SIAM J. COMPUT.
Vol. 30, No. 1, pp. 1–43

ON THE DYNAMIC FINGER CONJECTURE FOR SPLAY TREES. PART I: SPLAY SORTING log n-BLOCK SEQUENCES*

RICHARD COLE†, BUD MISHRA†, JEANETTE SCHMIDT‡, AND ALAN SIEGEL†

Abstract. A special case of the dynamic finger conjecture is proved; this special case introduces a number of useful techniques.

Key words. binary search tree, splay tree, amortized analysis, finger search tree

AMS subject classifications. 68P05, 68P10, 68Q20, 68Q25

PII. S0097539797326988

1. Introduction. The splay tree is a self-adjusting binary search tree devised by Sleator and Tarjan [ST85]. It supports the operations search, insert, and delete, collectively called *accesses*. The splay tree is simply a binary search tree; each access will cause some rotations to be performed on the tree. Sleator and Tarjan show that a sequence of m accesses performed on a splay tree takes time $O(m \log n)$, where n is the maximum size attained by the tree ($n \leq m$). They also show that in an amortized sense, up to a constant factor, on sufficiently long sequences of searches, the splay tree has as good a running time as the optimal weighted binary search tree. In addition, they conjecture that its performance is, in fact, essentially as good as that of any search tree. Before discussing these conjectures it will be helpful to review the operation of the splay tree and the analysis of its performance. The basic operation performed by the splay tree is the operation *splay*(x) applied to an item x in the splay tree. *splay*(x) repeats the following step until x becomes the root of the tree.

> SPLAY STEP.
> Let p and g be, respectively, the parent and grandparent (if any) of x.
> CASE 1. p is the root: make x the new root by rotating edge (x, p).
> CASE 2—THE ZIG-ZAG CASE. p is the left child of g and x is the right child of p, or vice-versa: rotate edge (x, p), making g the new parent of x; rotate edge (x, g).
> CASE 3—THE ZIG-ZIG CASE. Both x and p are left children, or both are right children: Rotate edge (p, g); rotate edge (x, p).

Henceforth, we refer to the rotation, single or double, performed by the splay step as a *rotation* of the access or splay operation. A rotation is the basic step for our analysis; the cost of one rotation is termed a *unit*; clearly, this is a constant.

*Received by the editors June 8, 1994; accepted for publication (in revised form) August 19, 1998; published electronically April 25, 2000. This work was supported in part by NSF grants CCR-8702271, CCR-8902221, CCR-8906949, CCR-9002819, CCR9202900, CCR-9503309, CCR-9800085, DMS-8703458, and IRI-9414862; by ONR grants N00014-85-K-0046 and N00014-89-J3042; and by a John Simon Guggenheim Memorial Foundation Fellowship. This work was made possible, in part, by the hospitality of the Laboratoire d'Informatique, Ecole Normale Supérieure.
http://www.siam.org/journals/sicomp/30-1/32698.html

†Computer Science Department, Courant Institute, New York University, 251 Mercer Street, New York, NY 10012-1185 (cole@cs.nyu.edu, mishra@cs.nyu.edu, siegel@cs.nyu.edu).

‡Incyte Pharmaceuticals, 3174 Porter Drive, Palo Alto, CA 94306, on leave from Polytechnic University, 6 Metrotech Center, Brooklyn, NY 11201 (jschmidt@incyte.com). This work was supported in part by NSF grant CCR-9305873.

Sleator and Tarjan use the following *centroid potential* to analyze the amortized performance of a splay operation. Node x is given weight, $wt(x)$, equal to the number of nodes in its subtree; they define the *centroid rank* of x, or simply the *rank* of x to be $\text{rank}(x) = \lfloor \log wt(x) \rfloor$ (our terminology). Each node is given a centroid potential equal, in units, to its centroid rank. Let δ denote the increase in centroid rank from x to g, if g is present; otherwise it denotes the increase in centroid rank from x to p. Sleator and Tarjan show that the amortized cost of the splay step if g is present is at most 3δ units, while if g is not present the cost is at most $\delta + 1$ units. Since the total increase in rank for the complete access is bounded by $\log n$, the amortized cost of an access is at most $3 \log n + 1$ units. More generally, this analysis can be applied to weighted trees, in exactly the same way. We call this the *centroid potential analysis*.

The operation $insert(x)$ is performed as follows: first, item x is inserted as in a binary search tree and then the operation $splay(x)$ is carried out. Clearly, the cost of an insertion is dominated by the cost of the corresponding access. So, subsequently, when analyzing the cost of insertions, we count only the cost of the splays themselves.

Now, we list the conjectures formulated by Sleator and Tarjan.

Dynamic Optimality Conjecture. Consider any sequence of successful accesses on an n-node binary search tree. Let A be any algorithm that carries out each access by traversing the path from the root to the node containing the accessed item, at a cost of one plus the depth of the node containing the item, and that between accesses performs an arbitrary number of rotations anywhere in the tree, at a cost of one per rotation. Then the total time to perform all the accesses by splaying is no more than $O(n)$ plus a constant times the time required by algorithm A.

Dynamic Finger Conjecture. The total time to perform m accesses on an arbitrary n-node splay tree is $O(m+n+\sum_{j=1}^{m} \log(d_j+1))$, where, for $1 \le i \le m$, the jth and $(j-1)$th accesses are performed on items whose ranks differ by d_j (ranks among the items stored in the splay tree). For $j = 0$, the jth item is interpreted to be the item originally at the root of the splay tree.

Traversal Conjecture. Let T_1 and T_2 be any two n-node binary search trees containing exactly the same items. Suppose the items in T_1 are accessed one after another using splaying, accessing them in the order they appear in T_2 in preorder (the item in the root of T_2 first, followed by the items in the left subtree of T_2 in preorder, followed by items in the right subtree of T_2 in preorder). Then the total access time is $O(n)$.

Sleator and Tarjan state that the dynamic optimality conjecture implies the other two conjectures. (The proof is nontrivial.)

There have been several works on, or related to, the optimality of splay trees [STT86, W86, T85, Su89, Luc88a, Luc88b]. [STT86] shows that the rotation distance between any two binary search trees is at most $2n-6$ and that this bound is tight; they also relate this to distinct triangulations of polygons; although connected to the splay tree conjectures, this result has no immediate application to them. [W86] provides two methods for obtaining lower bounds on the time for sequences of accesses to a binary search tree; while some specific tight bounds are obtained (such as accessing the bit reversal permutation takes time $\Theta(n \log n)$), no general results related to the above conjectures follow. [T85] proved the scanning theorem, a special case of the traversal conjecture (also a special case of the dynamic finger conjecture): accessing the items of an arbitrary splay tree, one by one, in symmetric order, takes time $O(n)$. Sundar [Su89] considered various classes of rotations on binary trees. Among other things,

this led to results concerning the deque conjecture; formulated by Tarjan [T85], it states that if a splay tree is used to implement a deque, in the natural way, then a sequence of m operations on a deque, initially of n items, take time $O(m+n)$; Sundar proved a bound of $O((m+n)\alpha(m+n))$.

In this paper we investigate a special instance of the splay sorting problem, which is related to the dynamic finger conjecture. Splay sorting is defined as follows. Consider sorting a sequence of n items by inserting them, one by one, into an initially empty splay tree; following the insertions, an inorder traversal of the splay tree yields the sorted order. We call this *splay sort*. A corollary of the dynamic finger conjecture is the following.

SPLAY SORT CONJECTURE. Let U be sequence of n items. Suppose the ith item in U is distance I_i in sorted order from the $(i-1)$th item in U, for $i > 1$. Then splay sort takes time $\mathrm{O}(n + \sum_{i=2}^{n} \log(I_i + 1))$.

Incidentally, an interesting corollary of the splay sort conjecture is the following.

SPLAY SORT INVERSION CONJECTURE. Let U be sequence of n items. Suppose the ith item in U has I_i inversions in U (counting inversions to both the left and right). Then splay sort takes time $\mathrm{O}(n + \sum_{i=1}^{n} \log(I_i + 1))$.

In the remainder of this paper we prove the splay sort conjecture for the following type of sequence. Suppose the sorted set of n items is partitioned into subsets of $\log n$ contiguous items, called *blocks*. Consider an arbitrary sequence in which the items in each block are contiguous and in sorted order. We call such a sequence a *$\log n$-block sequence*. We show an $\mathrm{O}(n)$ bound for splay sorting a $\log n$-block sequence. It is convenient to assume that the set being sorted comprises the integers $1 \ldots n$.

In brief, our analysis has the following form. The first insertion in each block is provided $\Theta(\log n)$ potential; it is called a *global* insertion. Every other insertion in the block is provided $\mathrm{O}(1)$ potential; these insertions are called *local* insertions. In section 2, we analyze the global insertions; then, in section 3, we modify the analysis to take account of local insertions.

Our work has a number of interesting features and introduces several new techniques for proving amortized results.

1. We introduce the notion of *lazy potential*; this notion can be viewed as a tool for designing potential functions. The lazy potential is a refinement of an initial potential function that avoids waste when potentials decrease. (For an example of waste, consider the following splay tree analyzed using the centroid potential. The tree is a path of n nodes, each of unit weight. The last node on the path is accessed. The resulting splay will have a real cost of $\Theta(n)$ but will reduce the potential by $\Theta(n \log n)$. The desired amortized cost of this operation is $\mathrm{O}(\log n)$, so essentially all the reduction in potential is wasted.) The idea of the lazy potential is to keep an old potential, ϕ_{old}, in situations in which the creation of a new (larger) potential, ϕ_{new}, may be followed by a return to the ϕ_{old} potential, with essentially $\phi_{new} - \phi_{old}$ potential being wasted. Not surprisingly, some care is needed in choosing which operations to treat as "lazy."

2. The potential depends on the access sequence; that is, identical trees, if created by distinct access sequences, may have different potentials. (Actually, this is a difference "in principle"; we have not constructed any examples to demonstrate it.) Nonetheless, the bounds we obtain do not depend on the input sequence.

These techniques are used later in the proof of the dynamic finger conjecture [C00]. Some of the techniques are slightly generalized so that they can be used as modules in this later paper.

2. Global insertions. We start with some definitions.

The *left path* of a tree comprises the nodes traversed in following left child pointers from the root, but excluding the root itself; the *right path* is defined analogously. Collectively, they are called *extreme paths*. Extreme paths of a subtree rooted at v are defined analogously; they are also called v's extreme paths. Node u is a *right ancestor* of node v if u is an ancestor of v that is to the right of v in symmetric order; note that v must be in u's left subtree. A *left ancestor* is defined analogously. A *right edge* is an edge to a right child; a *left edge* is defined analogously.

A *block* B is an interval $[i, j] \subseteq [1, n]$ of items; here a block always comprises $\log n$ items with $i = c \log n + 1$ and $j = (c + 1) \log n$ for some integer c. Any block $[i, j]$ induces a binary tree T_B, called the *block tree* of B, which comprises exactly those nodes of the splay tree, S, containing items i to j; they are called the *nodes* of B. Loosely speaking, the block tree is constructed by shrinking paths in S between nodes of B to single edges. More formally, the root, r, of T_B is the lowest common ancestor of nodes i and j in S. The left (resp., right) subtree of r is the tree induced by the set of items in S to the left (resp., right) of r, if nonempty; otherwise the subtree is empty.

The *root* of block B is the root of the corresponding block tree. The *global nodes* are exactly the block roots. The remaining nodes are called *local nodes*.

Every node on an extreme path of a block tree carries a potential of c units, c a constant to be specified later. Also, each node on an extreme path may carry a debit, either *small* or *large*, comprising sd and ld units, respectively; sd and ld are constants that are specified later. A node may not have both a small and a large debit. We note that for any block B the only nodes that may be visited henceforth are those presently on the extreme paths of its block tree, plus its root.

Each global node is given a centroid potential, called its *global potential*; it is defined on the splay tree using the following weights: each global node has weight 1 and each local node has weight 0. It is convenient to define a *global rank* for all nodes: this is the centroid rank in the splay tree under this weighting; $g\_\mathrm{rank}(v)$ denotes the global rank of v. Global nodes have a global potential equal to gp times their global rank, gp a constant to be specified later; the local nodes do not have a global potential.

To avoid difficulties created by the insertion of weighted nodes, for the purposes of the analysis we preinsert all the nodes into an ordinary binary search tree (without splays) in the order of their insertion into the splay tree. Then, in this same order, we access each node, in the binary search tree, by a splay operation. It is easy to see that the rotations performed on the binary search tree are identical to those performed on the splay tree. However, now each insertion can be treated as an access.

In the analysis of global insertions, whenever a rotation is performed (and paid for), another s spare rotations are also paid for, s a constant to be specified later. The spare rotations are needed subsequently to handle the effects of local insertions. We provide $(3 \log n + 1)gp$ units to pay for a global insertion. If a rotation increases the global rank of the item being inserted by Δ, then the rotation is paid for with $3\Delta gp$ of these units. Note that the rank of an inserted item increases by at most $\log n$ (from ≥ 0 to $\log n$). If the last rotation (the one involving the root of the splay tree) involves just two nodes, we call it an *incomplete rotation*; the remaining gp units are used as additional payment for the incomplete rotation, if any.

To avoid special cases it is convenient to redefine the access path for an insertion to exclude the splay tree root r in the event that r is involved in an incomplete

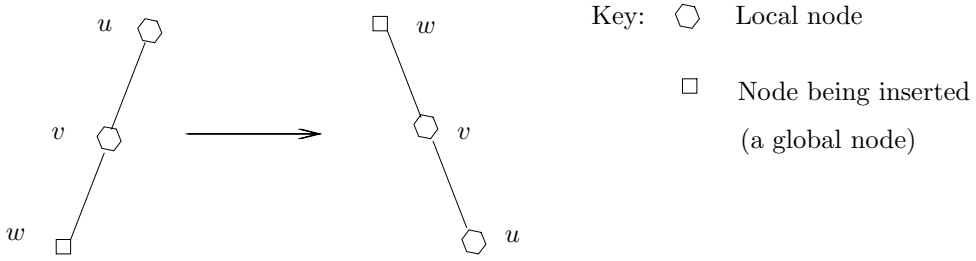


FIG. 1. *A couple comprising two local nodes.*

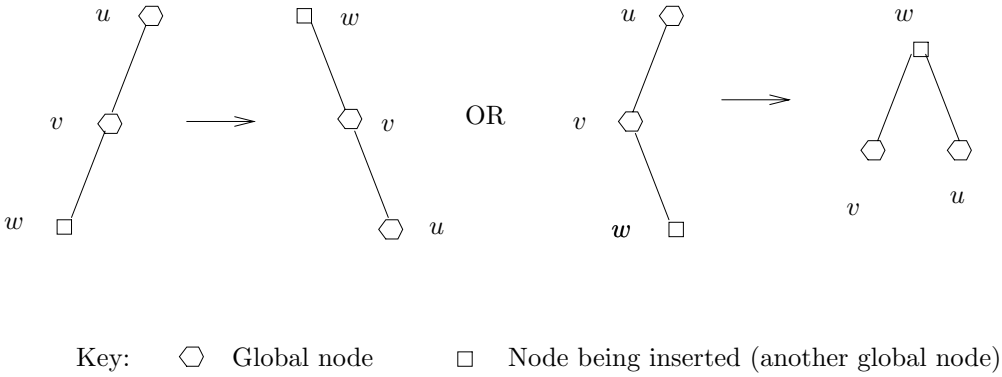


FIG. 2. *A couple comprising two global nodes.*

rotation. Now consider a rotation performed during the splay along the access path. Of the three nodes involved in the rotation, the top two are called the *coupled* nodes of the rotation, or a *couple* for short. The analysis focuses on the coupled nodes in a rotation. In order to provide the reader with some intuition we describe two simple cases.

Case 1. The coupled nodes are both local nodes in the same block (see Figure 1). Nodes u and v are on an extreme path of their block, without loss of generality the left path; v is the left child of u, and w, the node being inserted, is the left child of v. Since node u leaves the extreme path of its block it loses potential c. This pays for the removal of debits (small or large) from nodes u and v and in addition pays for $s+1$ rotations. Later, we see that we want $c \geq s+1+\max\{2sd, ld\}$.

Case 2. The coupled nodes are both global nodes (see Figure 2). Again, let w be the inserted node, let v be w's parent and u be v's parent. This case is analyzed using the centroid potential analysis of Sleator and Tarjan. The cost is at most 3 times the jump in global potential from w to u; to pay for $s+1$ rotations, it suffices to have $gp \geq s+1$; later, we see that in fact we want $gp \geq 6(s+1)+11sd$.

The full analysis is more involved. We begin by stating several invariants about the debits.

INVARIANT 1. *Only local nodes on an extreme path of their block can have debits.*

INVARIANT 2. *See Figure* 3. *Let v be a local node on an extreme path of block B. Suppose that in the splay tree v is the left (resp., right) child of its parent u. v can*

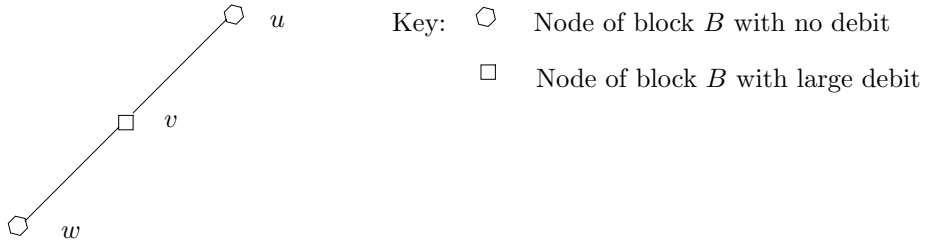


FIG. 3. *Large debits.*

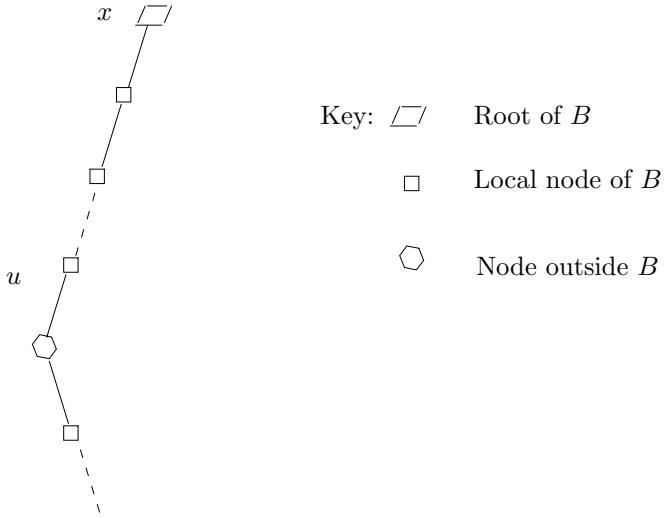


FIG. 4. *A small debit.*

have a large debit only if

(i) *u is a local node of block B,*

(ii) *v has a left (resp., right) child w which is a local node of block B, and*

(iii) *neither u nor w carry any debit,*

(iv) *either w is a local node of block B, or it is the item being accessed.*

INVARIANT 3. *Let u be the root of block B. Let v be a child of u. If v is in B, then v can have a small debit only if* $g\_\mathrm{rank}(v) < g\_\mathrm{rank}(u)$.

INVARIANT 4. *See Figure* 4. *Let x be the root of block B. Let P be an extreme path of the subtree (of the splay tree) rooted at x. Let u be the bottommost node on the path P that is in B. u can have a small debit only if* $g\_\mathrm{rank}(u) < g\_\mathrm{rank}(x)$.

For the purposes of the analysis the access path is partitioned into segments. Recall that each successive two nodes on the path form the coupled nodes of a rotation of the present splay operation. Each segment comprises an even number of nodes, or equivalently a collection of couples. The segments are created by a traversal of the access path from bottom to top; each segment is chosen to have the maximum length such that following the removal of its front (top) two nodes, the (truncated) segment satisfies the following conditions:

(i) The node being inserted has the same global rank throughout the rotations

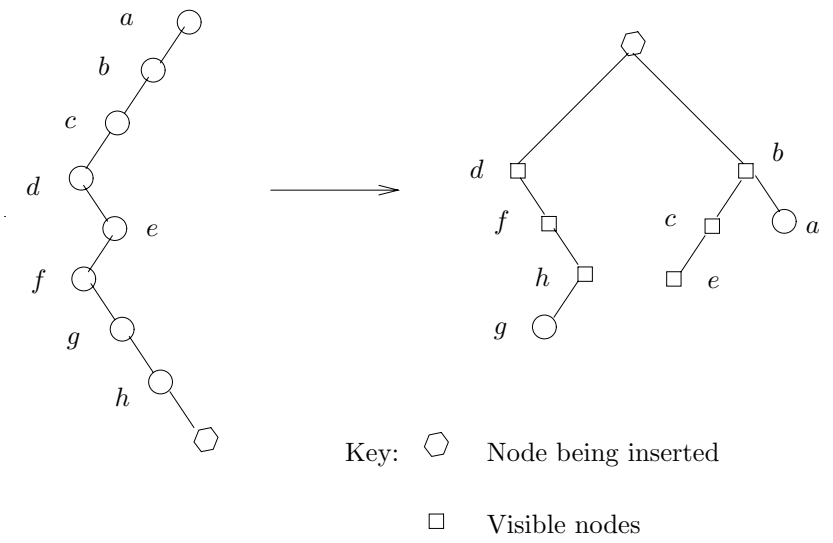


FIG. 5. *Visible nodes.*

involving the truncated segment.

(ii) Each global node on the truncated segment has the same rank following its rotation.

(iii) (This is implied by (i) and (ii).) Each couple in the truncated segment includes at least one local node (i.e., it does not comprise two global nodes).

(iv) See Figure 5. Define a node to be *visible* if it is involved in a zig-zag rotation or it is the lower node in a couple. (Intuitively, the visible nodes are those that remain on one of the traversed paths following the splay. Note that the splay, in general, creates two traversed paths.) For each block there are at most two visible nodes in the truncated segment that are local following the rotation.

The topmost segment is said to be *incomplete* if it satisfies conditions (i)–(iv) prior to truncation. We consider an incomplete segment to comprise a (trivially) truncated segment.

Next, we mark the following types of coupled nodes in each truncated segment. The rotations involving marked couples are self-paying, as is demonstrated later. For each type below, suppose the segment includes a couple, u, v, with u the parent of v.

Type 1. See Figure 6. Suppose u is the root of v's block; then both u and v are marked.

Type 2. See Figure 7. Suppose u is a local node, and let x be the root of u's block. Further suppose u is on the left (resp., right) path descending from x. Let v be the left (resp., right) child of u; if x is in the truncated segment and if v is global, then both u and v are marked.

Note that in a type 2 couple, the node being inserted must be a left (resp., right) child of v if v is a left (resp., right) child of u. Otherwise, the ranks of at least one of v and x would decrease following the rotations of their couples.

Type 3. Suppose u and v are both local nodes of the same block; then both u and v are marked.

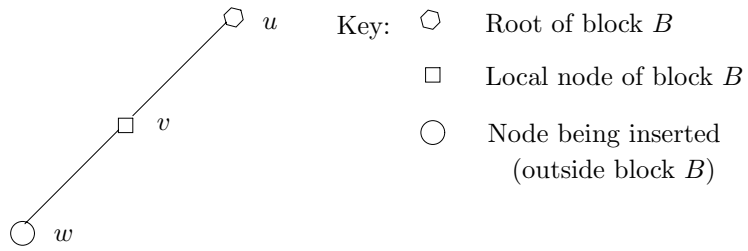


FIG. 6. *Type* 1 *couple.*

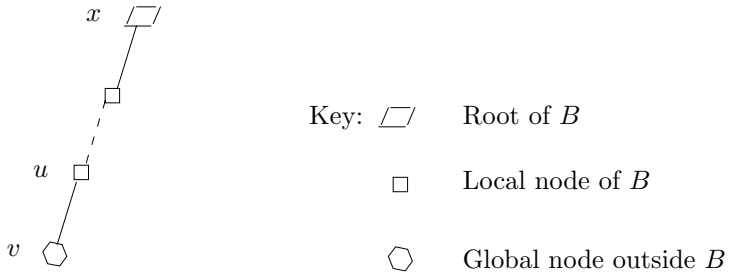


FIG. 7. *Type* 2 *couple.*

We can now prove a bound on the length of a truncated segment.

LEMMA 1. *A truncated segment comprises at most* 10 *unmarked nodes, of which at most* 7 *are local.*

Proof. Define the *left (resp., right) side* of the segment to comprise those nodes that are to the left (resp., right) of the item being inserted. From (ii) we deduce that one side, at least, contains no global nodes; without loss of generality suppose that this is the right side.

Observation 1. The right side contains at most two unmarked nodes. For as the right side contains no global nodes, its nodes must all come from the same block; so, by (iv), the right side comprises at most two sets of contiguous nodes, each one of at most two nodes. Then, because of type 3 markings, there are at most two unmarked nodes remaining on the right side.

The left side of the segment is partitioned in the obvious way into subsegments by the nodes on the right side. By Observation 1, the left side comprises at most three subsegments.

Observation 2. There are at most two couples that include both a node on the left side and a node on the right side. This follows from (iv) applied to the right side of the segment.

Observation 3. There is at most one unmarked couple within each subsegment; if present, this couple comprises the topmost global node and its parent, a local node. For the marking strategy marks all other couples within the subsegment. This is a total of at most three unmarked couples.

Thus there are a total of at most 5 unmarked couples, and at most 2 of these do not include a global node; so there are at most 7 unmarked local nodes. □

It is not hard to see that the marked nodes are all involved in zig-zig rotations.

Next, we show how to pay for the rotations (and associated spares) along a segment. Each marked couple pays for its rotation (and spares), as follows.

Type 1. See Figure 6. By Invariants 2 and 3, node v does not have a debit. The rotation is paid for by giving node u a small debit following the rotation. It is straightforward that Invariants 1–4 are unaffected. So it suffices that

$$sd \geq s + 1. \tag{1}$$

Type 2. See Figure 7. By Invariants 2 and 4, node u does not have a debit (for $g\_\mathrm{rank}(x) = g\_\mathrm{rank}(u)$ as x and u are both in the truncated segment). The rotation is paid for by giving u a small debit following the rotation. Again, it is straightforward that Invariants 1–4 are unaffected. Here too, (1) suffices.

Type 3. This is the Case 1 analyzed earlier. We need to pay for the rotation plus the removal of two small debits or one large debit. Again, it is straightforward to check that Invariants 1–4 are unaffected. So it suffices that

$$c \geq s + 1 + \max\{2sd, ld\}. \tag{2}$$

There remain up to 6 rotations to account for: the up to 5 unmarked couples and the first couple of the segment. These rotations are paid for by the first couple in the segment, which was removed in truncating the segment and which causes a violation of at least one of the conditions (i)–(iv), except in the case of an incomplete topmost segment, which is handled subsequently. The cost of these rotations is called the *segment charge*. Either the rotation involving the first couple causes a change in global potential (Case A) or it creates a sequence of three contiguous local nodes from the same block (Case B), or possibly both. Both cases involve three costs.

1. $Cost1$. The rotation and spares for each couple: $\leq 6(s + 1)$.
2. $Cost2$. Removal of small debits from nodes on the segment; $Cost2$ is analyzed below. (A node with a large debit will always be a marked node of type 3.)
3. $Cost3$. Removal of small debits for local nodes that now violate Invariant 3 or 4; $Cost3$ is analyzed below.

LEMMA 2. *For each segment* $Cost2 + Cost3 \leq 11sd$.

Proof. A definition is helpful here. Let B be a block and let v be the root of B. Consider an extreme path of block B and consider the topmost portion that is contiguous in the splay tree. If this topmost portion is incident on v in the splay tree, then it is called an *abutting extreme path portion* for v, or an *abutting portion* for short. (*Comment.* Node v of Invariant 3 is the top node of an abutting portion for u, while node u of Invariant 4 is the bottom node of an abutting portion for x.)

Next, we make some observations about $Cost3$. Note that none of the nodes from marked couples on the traversed path retain small debits; so these nodes do not contribute to $Cost3$. Contributions to $Cost3$ can arise in one of two ways:

(1) One way contributions to $Cost3$ can arise is through a traversed global node v having a reduction in rank. Then, on an already abutting portion (for v) which continues to be abutting (for v), if the portion was not traversed, we may need to remove up to two single debits in order to maintain Invariants 3 and 4. v can have at most one such abutting portion. This contributes up to $2sd$ to $Cost3$. This contribution can be a consequence of either a zig-zag rotation or a zig-zig rotation. We examine each in turn.
   (a) In a zig-zag rotation, there may be one such abutting portion for each global node in the couple. In this case the contribution to $Cost3$ can be as large as $4sd$.

(b) In a zig-zig rotation only the top node in the couple can retain such an abutting portion; so here the contribution to $Cost3$ is at most $2sd$.

(2) Contributions to $Cost3$ can also arise through the creation of an abutting portion, provided that this abutting portion is not traversed. This can only occur for the top node v of a couple and then only if both nodes in the couple are global and the rotation is a zig-zig rotation. Here, the contribution to $Cost3$ is at most $2sd$.

Next, we determine, in turn, the contribution to $Cost2 + Cost3$ of the truncated segment and of the first couple.

First, we consider the truncated segment. By Lemma 1, the contribution to $Cost2$ is at most $7sd$. For a contribution to $Cost3$, the discussion of the previous paragraph shows we need only consider possibility (2) since global nodes in the truncated segment do not have a change in global rank. Likewise, possibility (2) cannot arise for the couples of the truncated segment each have at most one global node. We conclude that the truncated segment contributes at most $7sd$ to $Cost2 + Cost3$.

We turn to the leading couple. The contribution to $Cost2$ is at most sd for each local node it contains. By the above discussion regarding contributions to $Cost3$, the leading couple may contribute up to $4sd$ to $Cost3$, but only if both nodes of the couple are global; if there is only one global node in the couple, then the contribution to $Cost3$ is at most $2sd$. In any event, the total contribution of the first couple to $Cost2 + Cost3$ is at most $4sd$. □

LEMMA 3. *A global insertion has amortized cost at most* $(3 \log n + 1)gp$ *units, where* $gp, ld = 17(s + 1)$, $c = 18(s + 1)$, *and* $sd = s + 1$.

Proof. In Case A, the segment charge is paid for either by the drop in global potential, which provides at least gp units, or, if there is an increase in global rank, by gp units of the at least $3gp$ units provided for this rotation. In Case B, the middle node in the sequence of three contiguous nodes is given a large debit, which pays for the segment charge. So it suffices to have

$$gp, ld \geq 6(s + 1) + 11sd. \tag{3}$$

Now, we show how to pay for the incomplete segment, if present. Recall that we provided $(3 \log n + 1)gp$ units to pay for the present insertion. The $+gp$ term is used to pay for the incomplete segment; in addition, this term is used to pay for the incomplete rotation, if any; however, the $+gp$ term does not need to account for any increase in rank on the part of the inserted item during the incomplete rotation, for this has already been accounted for. Clearly, the result of Lemma 2 applies here too (in fact, a tighter bound can be shown). Here too, (3) suffices.

The result follows by taking equalities in (1)–(3). □

Actually, we have shown the following.

LEMMA 4. *The path traversed in an access of an item can be partitioned into segments, such that for any segment* σ, *the amortized cost of the rotations for the couples of* σ *is at most* $gp + 2gp \cdot I_\sigma$, *where* I_σ *is the increase in rank undergone by the accessed item in the last rotation of segment* σ. *Furthermore,*

(i) $gp \cdot I_\sigma$ *units of the cost are used to raise the potential of the accessed item;*

(ii) $gp \cdot I_\sigma$ *units of the cost are used to raise the potential of the lower global item, if any, in the final couple of segment* σ*;*

(iii) gp *units are used to pay for all the rotations of* σ, *and all the associated spare rotations and debit removals.*

Remark 1. Later, the form of the blocks will be generalized to allow more complex

situations involving local nodes. Specifically, we will allow local nodes to carry other debits. For the above analysis to continue to apply, it will suffice that

(i) if a local node on its block's right (resp., left) path carries a new debit, then its parent and right child (resp., left child) are both local nodes of the block;

(ii) a zig-zig rotation in which the couple comprises two local nodes pays for the removal of all debits on the couple's nodes, plus $s + 1$ rotations, if the rank of the accessed item does not increase.

Indeed, in a subsequent paper, we will introduce a hierarchy of blocks. A block will comprise a contiguous set of items (with respect to the items' sorted order). A superblock will comprise a contiguous set of blocks. (In this section, the superblock is simply the whole splay tree.) As here, the root of each block is a global item and the other items are local items. Suppose each global item has a positive integer weight and each local item has nonnegative integer weight. Suppose further that there are small and large debits which obey Invariants 1–4; there may be additional types of debits also, but they must obey the following invariant.

INVARIANT 5. *Let v be a local node on an extreme path of block B. Suppose that in the splay tree v is the left (resp., right) child of its parent u. v can have an additional debit only if*

(i) *u is a local node of block B,*

(ii) *v has a left (resp., right) child w which is a local node of block B.*

Then the following lemma has been shown.

LEMMA 5. *Let v be a global node in a superblock B. Suppose that v is the accessed item. Let I be the increase in global rank undergone by v during some (arbitrary) portion P of the access. The amortized cost of the rotations in portion P of the access is bounded by $(3I + 1)gp$ units, where $gp, ld = 17(s + 1)$, $c = 18(s + 1)$, and $sd = s + 1$, provided the conditions of Remark 1 hold. The amortized cost of the traversal comprises a payment of $s+1$ rotations for each couple, the removal of debits on the path traversed and the removal of other debits needed to restore Invariants 3 and 4, minus the debits that are created.*

Proof. The only change compared to Lemma 3 is the introduction of additional debits as constrained by Remark 1. Let x be a node carrying such a debit and let w be its parent. Suppose x is the right child of its parent. Let y be the right child of x. By assumption, w, x, and y are all local nodes of their superblock B. Since the inserted node is global in superblock B, the rotation involving v and x must be a zig-zig rotation; the second node of the couple will be either w or y. The additional cost introduced by the removal of the additional debit from x is paid for by the zig-zig rotation involving node x, as assumed in Remark 1. □

3. Local insertions. We show how to analyze a sequence of local insertions. This leads to the introduction of *lazy trees* which forces a reanalysis of global and local insertions. Following this reanalysis, the overall analysis is readily concluded.

Recall that the insertion of each block comprises one global insertion followed by a sequence of $\log n - 1$ local insertions. Each local insertion traverses a left path up to the right child of the splay tree root (see Figure 8); we call this path the *local access path*. By providing $e(3 \log n + 1)gp$ units to the global insertion, we can treat the first $e - 1$ local insertions as if they were global insertions. Throughout these $e - 1$ local insertions, spare rotations will be accumulating (though, for reasons that will become clear later, it may be that no spares accumulate during the global insertion itself); more precisely, each couple receives s spares in each of these local insertions (see Lemma 7). These spares, plus the spares already accumulated by the nodes of

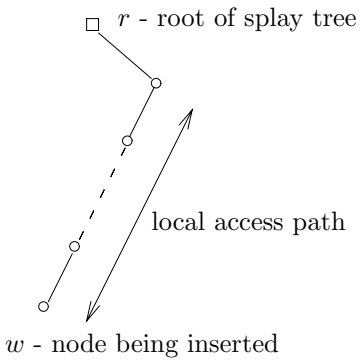


FIG. 8. *The local access path.*

the couple, are given to the node remaining on the local access path. So after $e-1$ local insertions, each node on the local access path, except the topmost, will have at least $(2^{e-1}-1)s$ spare rotations. The rotations in subsequent local insertions will be self-paying, apart from the rotation involving the splay tree root. These subsequent local insertions are called *true local insertions*.

In fact, we provide another $gp \log n$ units to the global insertion. Following the first $e-1$ local insertions, we provide a *reserve* potential to each global node on the local access path. The reserve for global node u is defined as follows: let v be the first proper global descendant of u on the local access path, if any; let $g\_\mathrm{rank}(v)$ be v's global rank if v is present; otherwise, let $g\_\mathrm{rank}(v) = 0$. Then $reserve(u) = gp(g\_\mathrm{rank}(u) - g\_\mathrm{rank}(v))$. The role of the reserve potential will become clear later.

We need one further constant, q, to be specified later. We choose e so that

$$(2^{e-1}-1)s \geq q + (s+1). \tag{4}$$

We note $e \geq 2$. In a true local insertion, each node v removed from the local access path is given potential q; $s+1$ will bound the cost of the rotation involving v. This is paid for by the $(2^{e-1}-1)s$ potential associated with the node removed from the local access path. At the end of the sequence of local insertions, the nodes remaining on the local access path, apart from the topmost node, are also given potential q (these nodes all have $(2^{e-1}-1)s$ potential at hand, which more than suffices).

Consider a true local insertion. The inserted item, e, is not provided a global potential; rather, following the final rotation of the insertion, e becomes the root of the splay tree and acquires the global potential associated with the old root of the splay tree. There are three types of rotation on the local access path. They all involve a couple u, v, with u the parent of v. Either both u and v are local, or one of u or v is local and the other is global, or both are global. Only in the last case is a global node removed from the local access path. The rotations, apart from those involving two global nodes, cost at most $s+1$ units, for there is no increase in global rank in such rotations and there are no debits to remove (for following the global insertion there are only large debits on the nodes of the local access path, which are removed by the first local insertion; as $e \geq 2$, the current insertion is not the first local insertion). In the case of a rotation involving two global nodes we swap their global potentials. The node remaining on the local access path still has the correct global potential, but the node u removed from the path may have too small a potential, which is called its *lazy*

potential. Again, the cost of this rotation is $s+1$ units; however, Invariant 4 may no longer be true for the block B whose root now has a lazy potential. (Invariant 3, which might be in question, remains true, for the local nodes of block B that are now children of B's root had been on the local access path and hence had their debits removed.) All these rotations, as noted in the previous paragraph, are paid for by the potential at hand.

LEMMA 6. *A true local insertion has amortized cost* $c+s+1$ *units.*

Proof. As discussed in the paragraph before the lemma, all couples comprising two nodes on the local access path are paid for by the potential associated with the nodes of the couple. So the only rotation to be paid for is the final rotation of the insertion; it costs $s+1$ units. In addition, the item displaced from the root of the splay tree is given c units of potential, for it has become an additional node on an extreme path of its block. □

Remark 2. We will need to generalize this analysis to take account of the presence of lazy trees on the local access path. The previous analysis will continue to apply if the following conditions hold:

(i) Each node on the local access path has potential $(2^{e-1}-1)s$.

(ii) Each node on the local access path does not carry any debit.

(iii) The rotation of each couple on the local access path costs at most $s+1$ units.

(i) and (ii) will be achieved as here. (iii) will be discussed later.

Every node removed from the access path in a true local insertion and remaining on an extreme path of its block is called a *lazy node*, whether or not it has a lazy potential. Those lazy nodes with a lazy potential are called *heavy* nodes; the other lazy nodes are called *light* nodes. When the insertion of the current block is completed we form *lazy trees*. Each global node v remaining on the local access path becomes the root of a new lazy tree. Its left subtree is empty; its right subtree comprises those lazy nodes created during the insertion of the current block that are in v's right subtree in the splay tree. It is straightforward to see that each new lazy node is contained in a new lazy tree. We call the lazy trees as defined above *initial lazy trees*. (We will be adding and removing a few nodes from the initial lazy tree in order to obtain other lazy trees, which are the lazy trees that are actually analyzed.)

The intuition behind the lazy tree is that if all the lazy nodes were restored to a left path, then the lazy potentials would be the actual global potentials (perhaps with some swapping). Actually, difficulties are caused by the fact that the left subtree of the path may increase in size through rotations between the lazy tree and the remainder of the splay tree. So, strictly speaking, the intuition may be incorrect. More precisely, the constant potentials q provided to the lazy nodes will pay for rotations among lazy tree nodes until these nodes have their global potentials restored (in general, this will not happen by recreating the left path from which these nodes originated).

Let us return to the cost of the insertions. A global insertion, as noted above, costs $e(3\log n+1)gp+gp\log n$ units. We see below that the presence of lazy trees adds a further $4gp\log n$ units to the cost of the global insertions, for a total of

$$e(3\log n+1)gp+5gp\log n \text{ units.} \tag{5}$$

In section 4 we show how to modify the analysis of global and local insertions to account for the presence of lazy trees.

4. The analysis of lazy trees. Most of the analysis focuses on a subtree of the initial lazy tree, called the *truncated lazy tree* or the *lazy tree*, for short. It is

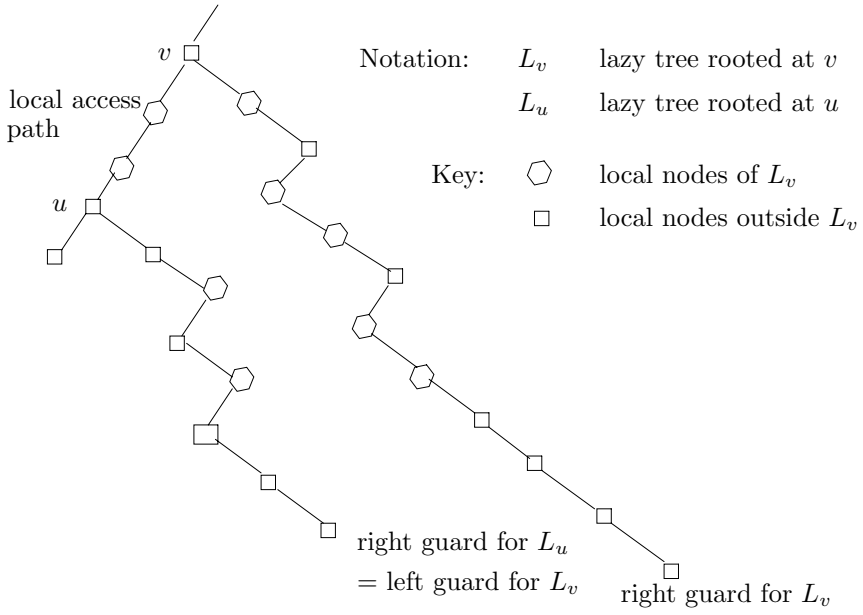


FIG. 9. *Left and right guards.*

defined as follows. Consider a new initial lazy tree, L, created by a sequence of local insertions. Consider the set of heavy nodes in L plus the root of L; the tree they induce is called the *initial lazy block tree*. We remove the rightmost node from the initial block tree; this defines the (truncated) lazy block tree for the (truncated) lazy tree. This rightmost node is called the *right guard* for the lazy tree (see Figure 9). The *left guard* is a global node, defined later, to the left of the nodes of the lazy block tree. The root of the (truncated) lazy block tree is called the root of the (truncated) lazy tree. We define the tree induced by the nodes of the (truncated) lazy block tree plus the left and right guards to form the *large lazy block tree*. Now we define the (truncated) lazy tree as follows.

We define the *left guard*, w, of the lazy tree as follows. Let L be a new lazy tree with root u. Let v be the first proper global descendant of u on the local access path, if any. Suppose v exists; if v is the root of another new lazy tree, let w be the right guard in this lazy tree, while if v is not the root of a new lazy tree, then let $v = w$. Otherwise, let w be the root of the splay tree. In general, a node may be a right guard in one lazy tree and a left guard in a second lazy tree.

The truncated lazy tree comprises the nodes of the (truncated) block tree together with the following light nodes. For each node v in the (truncated) block tree, we add the following nodes from v's block. Let w be any descendant of v in the large lazy block tree. Those nodes in v's block on the path from v to w in the splay tree are added to the (truncated) lazy tree. The light nodes added to the lazy tree form its *skeleton*.

Intuitively, a truncated lazy tree is a megablock comprising several of the blocks at hand. In its interactions with the remainder of the splay tree it will behave in the same way as a block. The root of the lazy tree corresponds to the block root, and the other nodes of the lazy tree, called *local* nodes of the lazy tree, correspond to the

local nodes of the block. The lazy tree local nodes may carry small and large debits according to the invariants specified for blocks; a further *lazy* debit may be carried as specified later.

The right guard of each new lazy tree has its global potential restored (this is paid for by adding its reserve potential to its lazy potential, which suffices, as is stated in Invariant 12, in section 4.3). In addition, Invariant 4 may need to be restored for the right guard; this requires the removal of at most two small debits. These are paid for by the potential $a + b$ (see the following paragraph) associated with the right guard prior to the restoration of its global potential. So it suffices to have

$$2sd \leq a + b. \tag{6}$$

Thus both the left and right guards of each lazy tree have their global potentials.

Next, we provide additional potentials to the nodes in the lazy tree (in addition to the potentials these nodes already carry). All light nodes on the lazy tree carry a potential of $2c'$ units, except for the light nodes from the leftmost block in the lazy tree, which carry a potential of only c' units; c' is a constant to be specified later. In addition, we give the following potential to the heavy nodes. Let v be a heavy node; suppose it has height h in the lazy block tree. Then v receives potential $a\sum_{i=1}^{h} i + b$, where a and b are constants to be specified later. This potential is provided by redistributing the potential q given to each lazy node in the initial lazy trees. Note that each initial lazy tree has the following form: consider the right path descending from its root; the left subtree of each node on this path, excluding the root, is a complete binary tree; in top to bottom order, these trees have strictly decreasing height. The q potentials are redistributed as follows. First, a potential of $a\sum_{i=1}^{k} i + b$ is given to each height k node in the initial lazy tree, other than the lazy tree root. The following argument shows that it suffices to provide each node with a potential of $12a + b$. Each node of height k in a complete binary tree begins by receiving a temporary potential of $\frac{3}{2}ak(k+1)+b$; it provides this by passing a charge of $\frac{3}{2}a[(k-1)k+4(k-1)]$ to each of its children; if it has a parent, in turn, it receives a charge of $\frac{3}{2}a[k(k+1)+4k]$ from the parent; adding its own initial potential of $12a+b$ provides exactly the required temporary potential (note that a leaf will pass a charge of 0 to its nonexistent children). The nodes on a right path acquire their potential from their left child. A node of height $k+1$ receives potential $k(k+1)a$ from its right child, which together with its own initial potential of $12a + b$ suffices to provide the required $[\frac{1}{2}(k + 1)(k + 2)]a + b$ potential; note that the right child still has potential $[\frac{1}{2}k(k + 1)]a + b$, as required. So it suffices to have

$$12a + b \leq q. \tag{7}$$

But the potentials in the initial lazy tree upper bound the potentials desired in the (truncated) lazy block tree. Finally, to provide the light nodes with their potential, it suffices to have

$$2c' \leq a + b. \tag{8}$$

The left (resp., right) extreme path of (truncated) lazy tree, L, is the path from the root of L to the leftmost (resp., rightmost) node in L; it is convenient to exclude the lazy tree root from the extreme paths. We will call these the left and right paths of L, for short.

Nodes on the extreme paths of the lazy tree may have a small or large debit; these debits satisfy Invariants 1–4, where lazy tree L replaces block B in the statement of

the invariants. Also, each light node in the lazy tree may have a *lazy* debit, which is *huge* and has value hd units, $hd \geq ld$, a constant to be defined later. We now state two invariants concerning the lazy debits.

INVARIANT 6. *Let L be a lazy tree. Suppose node v in L has a lazy debit. Then v is a light node of L. Also, v is not on the left path of L. Finally, v does not carry a small or large debit.*

INVARIANT 7. *Let L be a lazy tree. Let u be a local node of L. Suppose u has a lazy debit. Let v be the root of u's block.*

(i) *Suppose that v is not on the left path of L. Then if u is on the right path descending from v in the splay tree, both the parent and child of u on this path are local nodes in u's block.*

(ii) *If u is on the right path of L, then its parent and child in the splay tree are local nodes in u's block; this holds regardless of whether u is on the right path descending from v in the splay tree.*

Next, we show how to incorporate lazy trees into the analysis of global insertions. A global insertion can traverse a lazy tree in one of three ways:

(a) traverse the right extreme path of the lazy tree (or rather a topmost portion of it),

(b) traverse the left extreme path of the lazy tree (or rather a topmost portion of it), or

(c) traverse the interior of the lazy tree and thereby split the lazy tree.

Actually, it is convenient to classify a traversal of type (a) which is to the left of the right guard to be a split (a type (c) traversal); likewise a traversal of type (b) to the right of the left guard is defined to be a split. As we will see later, local insertions only involve traversals of type (a) or (b).

A traversal of type (c) will be paid for in two phases. First, in a preprocessing phase, the current lazy tree is partitioned into several lazy trees and/or ordinary blocks, so as to ensure that the actual splay (the second phase) comprises only traversals of types (a) and (b). (In fact, as we will see, we need a third phase in order to pay for some of the partitioning performed in the first phase.)

We start by considering the interactions between the lazy tree and the remainder of the splay tree (which may include other lazy trees). We treat the lazy tree, as delimited by its extreme paths, as a block. We note that Invariants 6 and 7 ensure that a node on an extreme path, carrying a lazy debit, satisfies condition (i) of Remark 1. In addition, we note that on creation, the nodes of the lazy tree have no debits, so Invariants 1–4 all hold at this point.

Next, we need to show that condition (ii) of Remark 1 is satisfied. That is, we have to show that couples involving two nodes on an extreme path of the lazy tree, which will be marked, can pay for the removal of their debits and $s + 1$ rotations. There are two classes of rotations: those involving at least one light node of the lazy tree and those involving two heavy nodes; they are treated in sections 4.1 and 4.2, respectively.

Segments are defined and paid for essentially as before. The one difference occurs if either node of the leading couple of the segment carries a lazy debit; then the couple pays for its own rotation and the removal of its debits as part of the lazy tree, as we see later; but this can only reduce the total remaining cost of the segment and so the bounds of the previous analysis are still valid.

We need to mention one detail about couples containing the root, u, of a lazy tree and another node, v in the same lazy tree. Following the rotation, v becomes the

root of the lazy tree; v and u interchange roles and potentials (so if v had been light, resp., heavy, u becomes light, resp., heavy).

In section 4.3 we will explain how a lazy tree is split.

4.1. Analysis of couples including one light node. Consider a couple comprising nodes u and v, where u is the parent of v, both u and v are on an extreme path of their lazy tree and one, at least, of u and v is a light node.

Case 1. u and v are both light. u ceases to be on the skeleton. u's c' potential pays for the rotation, s spares, and the removal of all debits on u and v. So it suffices to have

$$c' \geq s + 1 + 2hd. \tag{9}$$

Case 2. u is a light node and v is a heavy node. By Invariants 6 and 7, u and v do not have lazy debits. If u leaves the skeleton, the operation is paid for by u's c' potential, as in Case 1; (9) suffices. Otherwise, u is given a lazy debit; this then pays for the operation. The cost of the operation comprises the rotation, s spares, and the removal of small or large debits, if any, from u and v. So it suffices to have

$$hd \geq s + 1 + \max\{ld, 2sd\} = s + 1 + ld. \tag{10}$$

Case 3. v is a light node, u is the root of v's block (but is not the lazy block tree root). By Invariants 6 and 7, u and v do not have lazy debits. v becomes the block root. As in Case 2, if u leaves the skeleton, the operation is paid for by u's c' potential. Otherwise, u is given a lazy debit, which pays for the operation. The cost of the operation comprises the rotation, s spares, and the removal of small or large debits, if any, from u and v. Here too (9) and (10) suffice.

4.2. Analysis of couples comprising two heavy nodes. The potential associated with the lazy block tree is called the *lazy complete tree potential*.

Consider a couple comprising nodes u and v, where u is the parent of v and both u and v are heavy nodes on an extreme path of the lazy tree. We need to pay for the rotation, for s spares, and for the removal of small or large debits from u and v, if any. In addition, we may need to reestablish Invariant 7 for node u; this may require the removal of up to two lazy debits, which will also be paid for by the rotation. So the cost of this rotation is at most

$$s + 1 + ld + 2hd. \tag{11}$$

For the remainder of section 4.2 we focus on the lazy block tree. Hence when we refer to a node we mean a node in the lazy block tree; likewise a reference to a tree refers to the lazy block tree. The *depth* of a node in the tree is its distance from the root. The height of a node at the time of the creation of the lazy tree is called its *creation_height*; the creation_height does not change subsequently.

Consider how an extreme path traversal appears to the lazy block tree. A top contiguous portion of the nodes on this path are all *touched* (these are the nodes contained in couples of the splay tree). Some (arbitrary) subset of disjoint pairs of touched nodes form couples. Each touched node is provided with $\frac{s'}{2}$ spare units of potential, $s' \leq s$ (this is part of the node's portion of the s spares provided to its couple in the splay tree).[1] The spare potential of the touched nodes, together with

[1] In this paper we can take $s' = s$; the more general form will prove useful in a later paper.

any changes to the potentials of the touched couples will pay for the rotations of the touched couples.

At any time, certain nodes, called *active* nodes, are the nodes that pay for a traversal of the extreme path. Some nodes may pay more than other nodes. This is captured by the notion of *active layers*. A node of creation_height h has an associated *span* of layers $[1, h]$; each active node v of creation_height h has an *active span* of *active layers*, $(i, h]$, $0 \leq i \leq h$; for each l, $i < l \leq h$, we say v is *l-active*. If the active span is nonempty, we say the node is *active*.

Initially, only the nodes on the right path are active. An inactive node becomes active when it first reaches the right path. Once a node becomes active it remains active, whether or not it remains on the right path; also, the active span of a node can only grow.

Recall that a rotation of the splay operation involving a heavy node, v, and the root, r, of the lazy tree causes the two nodes to swap all their potentials. Also, r acquires v's creation_height and active span.

The following invariant states several properties of the active nodes. We prove the invariant later. In order to avoid special cases for the left path we state the invariant with respect to the normal form of the tree, defined as follows. It is obtained by performing the following series of single rotations: one by one, the left path nodes are moved to the right path; each such rotation, between node v and node r, the root of the lazy tree, makes v the root and places r on the right path. r acquires v's active span. The resulting tree is called the *normal tree*.

INVARIANT 8. *Let H be the maximum creation_height for the nodes, other than the root, present in the tree initially. Then, in the corresponding normal tree, the following hold:*

(i) *There is exactly one l-active node, $1 \leq l \leq H$.*
(ii) *Every node on the right path is active (this does not include the root).*
(iii) *Apart from the root, the ancestors of an active node are all active.*
(iv) *Let v be an l-active node. Let w be a j-active node. If $j < l$, then w is to the right of v in symmetric order, while if $j > l$, then w is to the left of v in symmetric order.*
(v) *Let inactive node v have creation_height l. Then its parent has creation_height greater than l.*

When a lazy tree is created the active spans for the nodes in the corresponding lazy block tree are initialized as follows. Let u be a node on the right path, of creation_height h; suppose it has a right child v of creation_height i (if there is no such node v, let $i = 0$). Then u is given active span $[i + 1, h]$. Clearly, the new tree obeys Invariant 8.

The layers of a node are further categorized as *black* or *white*; an active node, with active span $[j, h]$, can be black with respect to each of the layers $[1, j - 1]$. In general, an active node v, with active span $[j, h]$, is black with respect to all the layers in some range $[i, j-1]$, $i \geq 1$, called its *black span*; we say v is *l-black* for $i \leq l \leq j-1$. If the black span is nonempty we say the node is *black*. Nodes are initially white at all layers. A node becomes black as a consequence of a rotation with the root. The following invariant applies to black nodes.

INVARIANT 9.

(i) *All the nodes on the left path of a tree are fully black, i.e., a node with active span $[j, h]$ has black span $[1, j - 1]$.*
(ii) *If a node u is l-black, all u's left ancestors, apart from the root, in the corre-*

sponding normal tree are l-black.

We define the following distances for each node v, active at layer l. Its *right path l-distance*, $d_l(v)$, is the number of proper left ancestors of v on the right path, excluding all l-black nodes. (Recall that a left ancestor of v is an ancestor of v to the left of v in symmetric order.) Its *layer l interior right path distance*, $id_l(v)$, is the number of proper left ancestors of v below the first l-black node and below the right path.

For each l-active node we maintain an l-potential, which satisfies the following invariant.

INVARIANT 10. *The l-potential at node v is at least*

$$a \cdot \min\left\{\frac{d_l(v)}{d}, l\right\} + \frac{a}{d} \cdot id_l(v). \tag{12}$$

We note that initially a node has $a \cdot l$ units of potential for each layer l at which it could become active, so when a node first becomes l-active, Invariant 10 holds.

In addition, for each l-black node we maintain an l-black potential of $\frac{a}{2d}$ units.

In turn, we analyze traversals of the right and left extreme paths.

Immediately prior to a traversal of the right path, each node on that path is given $s'/2$ spare rotations (whether or not the node is part of a couple comprising two heavy nodes of the same lazy tree). Each node's spares are subsequently provided by the rotation which involves that node. Suppose node v remains on the right path following the rotation. Let layer l be the largest layer at which v is presently active. Suppose node u, the other node in v's couple, is not l-black. If $d_l(v) > d \cdot l$, then the spares, called the *l-spares*, for the nodes at depths $(d(l-1), d \cdot l]$ on the right path, are used to pay for v's rotation. Otherwise, a/d units of v's l-potential are used to pay for this rotation. Note that Invariants 8–10 continue to hold following this rotation.

If node w is l-active but is not on the right path, then $id_l(w)$ can increase; thus its l-potential can increase, but only if $d_l(w) > d \cdot l$. In this case, the unit increase in potential is paid for by the l-spares as in the previous paragraph.

A rotation between an l-black node u and its child v, where v is l-active, is paid for by the $\frac{a}{2d}$ units of l-black potential at u; u reduces its black span accordingly, as do all the nodes in u's new right subtree (the subtree following the rotation). Again, it is clear that Invariants 8 and 9 hold following the rotation. To verify Invariant 10 we argue as follows. (See Figure 10.) First, for j-active node w, $j < l$, in v's old right subtree, $d_j(w)$ is unchanged or reduced, since u ceases to be an ancestor of w; in addition, $id_j(w)$ is unchanged. Second, for j-active node w, $j > l$, in v's old left subtree, as in the previous paragraph, if v's j-potential increases, it is paid for by the j-spares. Third, for j-active node w, $j > l$, in u's left subtree, $d_j(w)$ and $id_j(w)$ are unchanged since the locations of w's left ancestors are unchanged, as is their j-black status. Reference to Invariant 8(iv) shows that these are the only possible cases.

A black node is created in a rotation with the root. Node v becomes the new root. Node u, the old root, acquires all of v's potentials; v acquires the root's global potential. If u now has minimum active layer l, it acquires black span $[1, l-1]$. The new portion of its black span, $[1, k-1]$, for some $k \le l$, is paid for as follows. Let w be the j-active node for some $j < k$. If $d_j(w) \le j$ before u's rotation, as $d_j(w)$ is reduced by at least one by u's rotation, $a/2d$ units of w's j-potential provide u's j-black potential. If $d_j(w) > j$, then let w' be the node at depth j. w' transfers a second portion of $\frac{a}{2d}$ units of potential. Note that by Invariant 8(iv) each active layer j' of w' satisfies $j' > j$, and so $d_{j'}(w') < j'$; thus the j'-potential of w' is reduced by

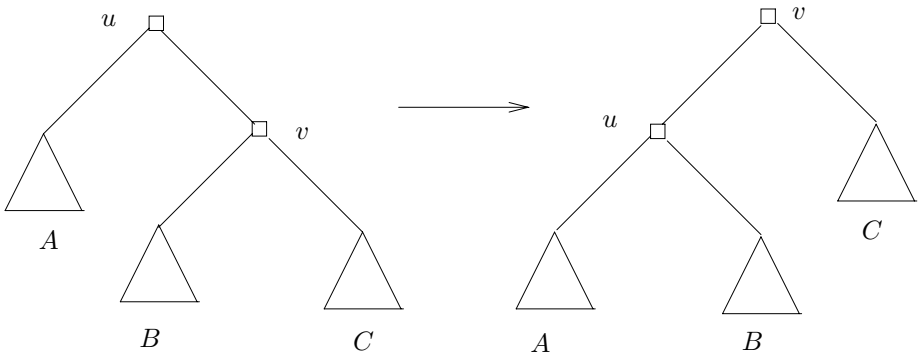
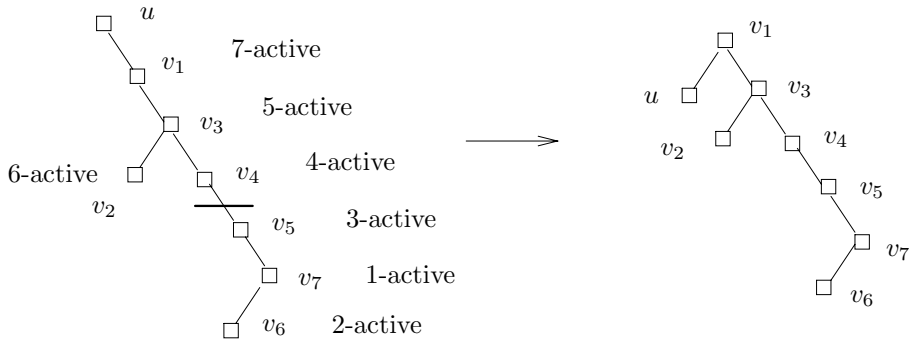


FIG. 10. *Rotations with black nodes.*



v_3 pays for the 5-black potential and the 2-black potential

FIG. 11. *Creating a black node.*

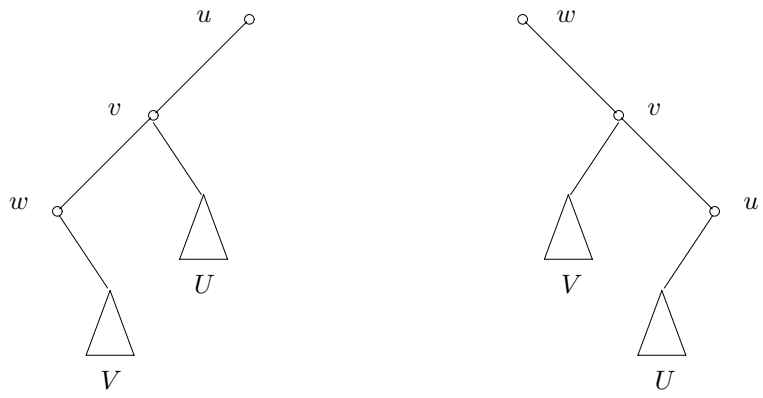
$\frac{a}{d}$, owing to the creation of the black node, and w' can afford to transfer two sets of $\frac{a}{2d}$ units of j'-potential to the new black node (one set for itself and one set for node w). (See Figure 11 for an example.)

We summarize the above discussion. The rotation of a couple costs $coup = s + 1 + ld + 2hd$ units. Suppose the bottom node of the couple is l-active; then these *coup* units are paid for by one of $\frac{a}{d}$ units of l-potential, $\frac{a}{2d}$ units of l-black potential, or the $\frac{d \cdot s'}{2}$ l-spares. While if the l-active node was not involved in a rotation with another global node of the lazy tree, the $\frac{d \cdot s'}{2}$ l-spares may have to provide it with $\frac{a}{d}$ units of potential. So it suffices to have

$$\frac{d \cdot s'}{2} \geq \max\left\{(s + 1 + ld + 2hd), \frac{a}{d}\right\}, \tag{13}$$

$$\frac{a}{2d} \geq s + 1 + ld + 2hd. \tag{14}$$

We turn to the analysis of left path traversals.



Actual situation v, u rotated onto right path

FIG. 12. *Left path traversals.*

Consider a couple, w, v, on the left path, with v the parent of w; let u be the parent of v (see Figure 12). If nodes u and v were on the right path, then subtree U (resp., V) would be the left subtree of u (resp., v). We treat the rotation of couple w, v on the left path as if it were the rotation of couple u, v on the right path. We pay for the rotation between v and w using w's present $\frac{a}{2d}$ units of l-black potential, where w is l-active (note that the present potentials of v and w correspond respectively to those v and u would have if on the right path). Then w and v swap potentials; v and all the nodes in its right subtree reduce their black spans accordingly. Invariants 8–10 continue to hold, as can be seen by arguments similar to those used for the right path traversal.

Remark 3. A left path traversal itself does not cause the spending of any spares, for all the rotations on the left path involving two heavy nodes of the lazy tree (which are the only rotations to spend spares) are paid for by black potentials.

We can conclude (as asserted in the second paragraph of section 3) the following.

LEMMA 7. *For each of the first $e-1$ local insertions in a block's insertion, for each couple, s spares can be provided to the couple.*

In a later paper we will again use lazy trees generalized in various ways. However, the nodes of the lazy trees in the later paper will still have small, large, and lazy debits which obey the present invariants, namely Invariants 1–7. Other additional debits may be present. They will satisfy the following invariant.

INVARIANT 11. *Let light node v carry an additional debit. Then*

(i) *v is not on the left path of the lazy tree;*

(ii) *if v is on the right path of the lazy tree, then so are its parent and right child in the splay tree.*

This invariant meets condition (i) of Remark 1; so long as condition (ii) is also met, the analysis of the traversal of extreme paths of the lazy trees is unaffected. But the presence of new debits, of value ed, say, merely raises the cost of a couple by at most $2ed$. It is convenient to let

$$md = \max\{ed, hd\}. \tag{15}$$

The new debits can be handled in the present analysis by modifying (9), (10), (13), and (14) so as to pay for the removal of up to two debits of value ed from each couple being analyzed, yielding

$$c' \geq s + 1 + 2md, \tag{16}$$

$$hd \geq s + 1 + \max\{ld, 2sd, 2ed\}, \tag{17}$$

$$\frac{d \cdot s'}{2} \geq \max\left\{(s + 1 + \max\{ld, 2ed\} + 2hd), \frac{a}{d}\right\}, \tag{18}$$

$$\frac{a}{2d} \geq s + 1 + \max\{ld, 2ed\} + 2hd. \tag{19}$$

4.3. Splitting the lazy tree. A split of the lazy tree occurs when the inserted item lies in value between the left guard and the right guard in the lazy tree. A split can occur only during a global insertion, as we show next, for a split causes a zig-zag rotation within the lazy tree. However, in a local insertion the only zig-zag rotation involves the splay tree root; but the root is in the same block as the inserted item and hence is not yet part of any lazy tree.

For this section, we assume that each node belongs to at most one lazy tree, apart, possibly, from the sharing of guards. In section 4.5, we analyze the general case.

The split is analyzed in three parts. First, we show that if the lazy tree satisfies Invariant 12, below, then, by promoting appropriate nodes (i.e., restoring their global potentials), the lazy tree is split into several new lazy trees which all also satisfy Invariant 12. In addition, the promotions cost at most $4gp \cdot logn$ units of potential. Second, we show that the cost of removing debits so as to restore all the invariants concerning debits can be charged to the promoted nodes at no further cost. Third, we show how to create the lazy tree potential for each new lazy tree from the corresponding potential for the lazy tree prior to the split.

In order to permit more general accesses in Part II of this paper, we show how to analyze a split when a local item in a block is accessed as well as when a global item is accessed. The goal, when a local item e is accessed, is to remove e's block from the lazy tree so that it becomes a normal block.

4.3.1. The promotions. At this point it is helpful to mention a few properties of the lazy and reserve potentials. Consider the heavy nodes in a new lazy tree. Let SL be the set of global nodes contained strictly between the guards of a lazy tree. For each heavy node, u, in the lazy tree, define its right neighbor $r\_n(u)$ to be the heavy node immediately to its right in the large lazy block tree, and define $SL(u)$ to comprise the subset of SL strictly to the left of u. Finally, define the lazy rank of heavy node v to be $\frac{1}{gp}$ times its lazy potential.

INVARIANT 12. *Let v be a heavy node in the normal form of the lazy block tree for lazy tree L. Then*

(i) $lazy\_\text{rank}(v) \geq \lfloor \log(wt(SL(v)) \rfloor$;

(ii) *if, in the normal form of the large lazy block tree, v has no heavy node or guard in its right subtree, $lazy\_\text{rank}(v) + 1/gp \cdot reserve(v) \geq g\_\text{rank}(v)$, while if v does have a heavy node or guard in its right subtree, then $lazy\_\text{rank}(v) + 1/gp \cdot reserve(v) \geq lazy\_\text{rank}(r\_n(v))$.*

Invariant 12 can be seen to hold when the lazy tree is created by considering node v at the point at which it becomes lazy. Also, it is readily seen that this invariant continues to hold following traversals of the extreme paths (for they leave the lazy rank of nodes in the normal form of the lazy tree unchanged).

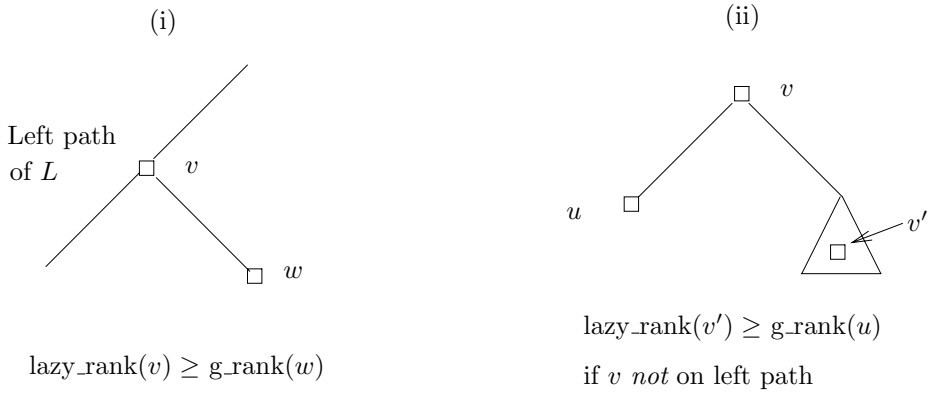


FIG. 13. *Heavy nodes.*

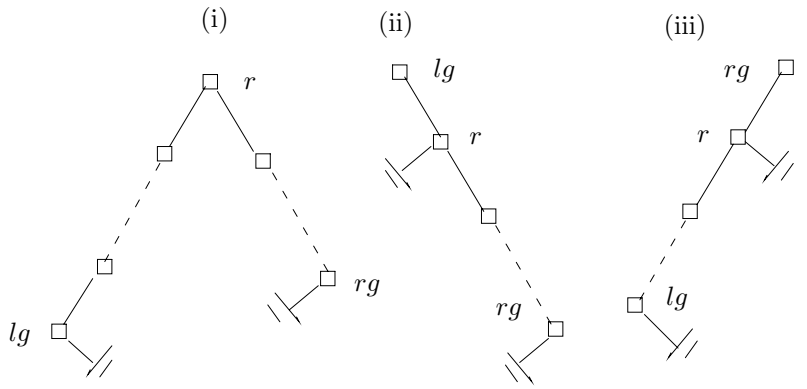


FIG. 14. *The large lazy block tree.*

COROLLARY 8. *See Figure* 13. *Let v be a heavy node other than the root in lazy tree L. Let w be v's right child and u be v's left child.*

(i) *Suppose that v is on the left path of L; then* $lazy\_\mathrm{rank}(v) \geq g\_\mathrm{rank}(w)$.

(ii) *Suppose that v is not on the left path of L. Let v' be a right descendant of v, not necessarily proper. If v' is a heavy node,* $lazy\_\mathrm{rank}(v') \geq g\_\mathrm{rank}(u)$.

It is convenient to define v's *lazy weight* to be $wt(SL(v))$.

The analysis is simplified by, on occasion, modifying the guards of a lazy tree following a traversal of one of its extreme paths. Specifically, we maintain the following invariant.

INVARIANT 13. *See Figure* 14.

Consider a lazy tree L. Let r denote the root of the truncated lazy block tree, and let lg and rg denote, respectively, the left and right guards of L. The large lazy block tree has one of the following three forms.

(i) *r is the root, and the left subtree of rg is empty, as is the right subtree of lg.*

(ii) *lg is the root of the large lazy block tree. Then r is the right child of lg, and r has an empty left subtree, as does rg.*

(iii) *rg is the root of the large lazy block tree. Then r is the left child of rg, and*

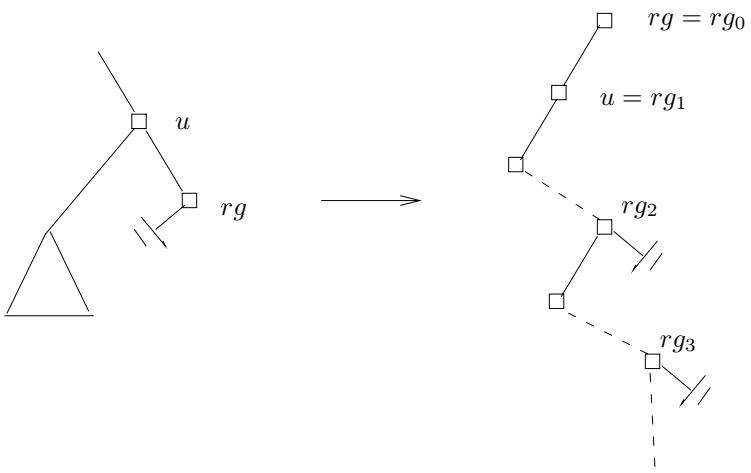


FIG. 15. *The right guard restoration process.*

r has an empty right subtree, as does lg.

Now, we show how to restore Invariant 13 following a traversal of an extreme path. Invariant 13 can cease to hold only if there is a couple containing a guard or the root of the lazy tree and a heavy node. So first consider a couple comprising a heavy node u on the right path and rg (see Figure 15). Following the rotation, let $rg_0 = rg$, and let rg_{i+1} be the rightmost node in the left subtree of rg_i in the large lazy block tree, if this subtree is not empty. This defines a nonempty sequence of nodes $rg_1, \ldots, rg_k$; all these nodes are promoted (i.e., they have their global potential restored) by adding their reserve potential to their lazy potential (this suffices by Invariant 12(ii)). rg_k becomes the new right guard for the remaining lazy tree rooted at r. This is called the *right guard restoration process* applied to rg.

Second, consider a rotation between lg and heavy node u on the left path. Let v be the parent of u in the large lazy block tree. Following the rotation, $lazy\_\text{rank}(u) \geq g\_\text{rank}(u)$ (for by Invariant 12(i), $lazy\_\text{rank}(u) \geq \lfloor \log(wt(SL(v))) \rfloor$—remember the lemma applies to the normal form of the lazy tree). In addition, following the rotation, u has an empty left subtree in the large lazy block tree (since lg had an empty right subtree before the rotation). So u is made the root of a new lazy tree, with left guard lg; temporarily, its right guard, rg_{new}, is the rightmost node in its right subtree in the large lazy block tree; rg_{new} acquires its global potential by adding its reserve potential to its lazy potential (by Invariant 12(ii), this suffices). See Figure 16. rg_{new} also becomes the left guard for the remainder of the lazy tree rooted at r. Then the right guard restoration process is applied to rg_{new} to obtain the actual new right guard for the new lazy tree rooted at u. There is one special case: u has an empty right subtree; then u simply becomes the root of a normal block and r acquires u as its new left guard. This is called the *left guard restoration process* applied to u.

Third, consider case (ii) of Invariant 13; suppose there is a rotation between r and its right child, u, a heavy node (see Figure 17). Then, following the rotation, the left guard restoration process is applied to r, which is now the left child of u, the new root of the lazy tree. Fourth, consider case (iii) of Invariant 13; suppose there is a rotation between r and its left child, u, a heavy node (see Figure 18). Then, following

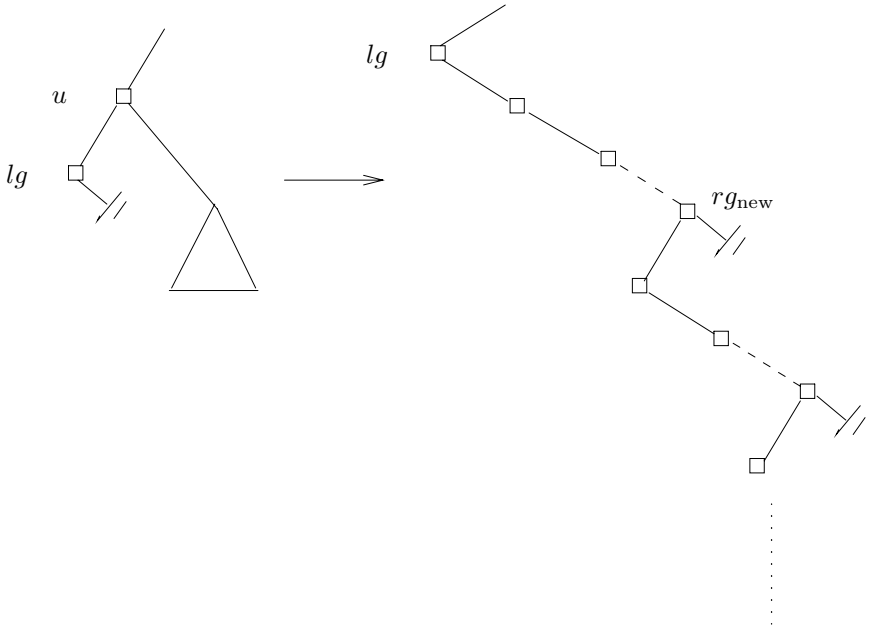


FIG. 16. *The left guard restoration process.*

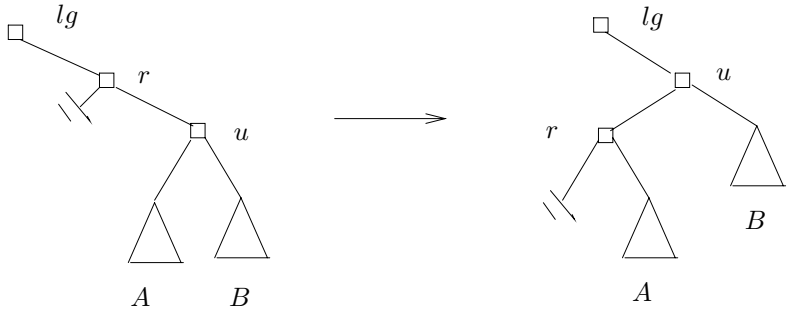


FIG. 17. *Guard restoration for a root and right child couple.*

the rotation, the right guard restoration process is applied to rg.

It remains to discuss how to maintain Invariants 1–10. We explain how to do this later in the section, when we show how to maintain these invariants following a split operation.

We view a split as occurring in three phases. In phase 1 no rotations are performed, but certain heavy nodes are marked as *promoted* (a node is promoted by increasing its lazy potential to its global potential). In fact the nodes are not promoted yet, but the splay will proceed as if they had been promoted. The effect of the promotions is to partition the original lazy tree into several new lazy trees. Phase 3 pays for these promotions. This ensures that in phase 2, the actual splay, only extreme paths of lazy trees are traversed. However, there will be one difference in paying for phase 2 as compared to the previous traversals. Any (apparently) promoted node

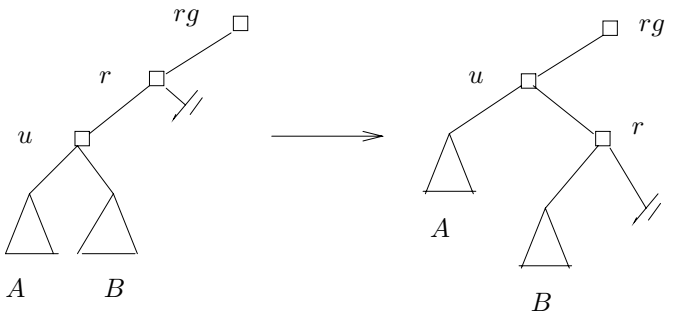


FIG. 18. *Guard restoration for a root and left child couple.*

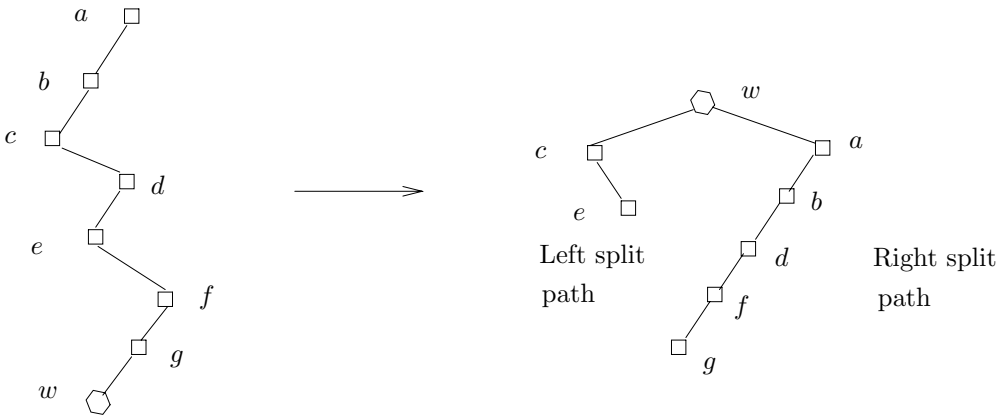


FIG. 19. *The split paths.*

whose global rank drops during the splay does not use gp units of its global potential to pay for the associated segment, for it does not yet have its global potential; instead paying for the segment becomes a charge to be paid for by the promotion (a charge of at most gp units). It is called the *segment charge*; the segment charge is paid for directly by the promoted vertex. Phase 3 pays for the remaining costs of the promotions, at most $4gp \log n$ units.

Phase 3 uses the following *imaginary* tree. Consider performing all the zig-zag operations of the insertion but replacing each of the zig-zig operations by two single rotations performed in bottom to top order. This creates two paths, called *split paths*, descending from the inserted item, the root of the imaginary tree; one path, the *left split path*, to the left of the root, descends to the right, and the other path, the *right split path*, to the right of the root, descends to the left (see Figure 19). The items on the split paths are exactly the items that will be traversed in the splay operation. We provide each global node on the split paths with an *imaginary* global rank, namely the global rank it has in the imaginary tree. The global ranks of the other global nodes are the same in the imaginary tree and the actual tree. Each of the split paths is traversed from bottom to top; for each global node at which there is a jump in imaginary global rank, denoted $jump(v)$, the following potential is provided:

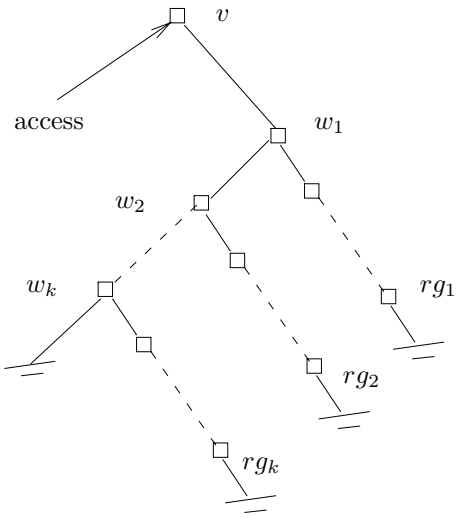


FIG. 20. *Right offset promotions for v.*

$2gp \cdot jump(v)$. The following corollary is helpful.

COROLLARY 9. *Let v be a heavy node in lazy tree L. Suppose that L is split.*

(i) *Also, suppose that v is on the left path of L and is also on the left split path. Then $lazy\_\text{rank}(v) + jump(v) \geq g\_\text{rank}(v)$.*

(ii) *Alternatively, suppose that v is not on the left path of L and is on the right split path. Let v' be a heavy node which is a right descendant of v, not necessarily proper. Then $lazy\_\text{rank}(v') + jump(v) \geq g\_\text{rank}(v)$.*

Proof. We prove (i); the proof of (ii) is very similar. Let w be v's right child in L (if w is not present, define $g\_\text{rank}(w) = 0$). Also, let w' be v's right child in the imaginary tree (again, if w' is not present, define $g\_\text{rank}(w') = 0$). Then, by Corollary 8(i), $lazy\_\text{rank}(v) \geq g\_\text{rank}(w)$. Now w' must be a descendant of w, so $g\_\text{rank}(w) \geq g\_\text{rank}(w')$. Finally, $jump(v) = g\_\text{rank}(v) - g\_\text{rank}(w')$. So $lazy\_\text{rank}(v) + jump(v) \geq g\_\text{rank}(w') + jump(v) = g\_\text{rank}(v)$. □

We now discuss which nodes are promoted in a split and how this is paid for. There are a number of cases.

Case 1. v is a heavy node in the lazy block tree, which is not on the left extreme path. In addition, v is accessed from its left child, u.

Then v is promoted. v's promotion costs at most $gp \cdot jump(v)$ (apply Corollary 9(ii)).

Let w_1 be v's right child, if any, in the lazy block tree, and let $w_2, w_3, \ldots, w_k$ be the maximal left path descending from w_1 in the lazy block tree (see Figure 20). $w_1, w_2, \ldots, w_k$ are also promoted. Between them, these promotions cost at most $gp \cdot jump(v)$. For, by Corollary 8(ii), $lazy\_\text{rank}(w_i) \geq g\_\text{rank}(w_{i+1})$, for $1 \leq i < k$. In addition, $lazy\_\text{rank}(w_k) + jump(v) \geq g\_\text{rank}(v)$ (by Corollary 9(ii)), and $g\_\text{rank}(v) \geq g\_\text{rank}(w_1)$. The nodes w_i become the roots of new lazy trees. Let rg_i be the rightmost node in the subtree of the old lazy block tree rooted at w_i for $1 \leq i < k$. Temporarily, rg_i is made the right guard for w_i. rg_i is promoted; this is paid for by its reserve potential (for note that rg_i has an empty right subtree in the large lazy block tree and apply Invariant 12(ii)). Then the right guard restoration

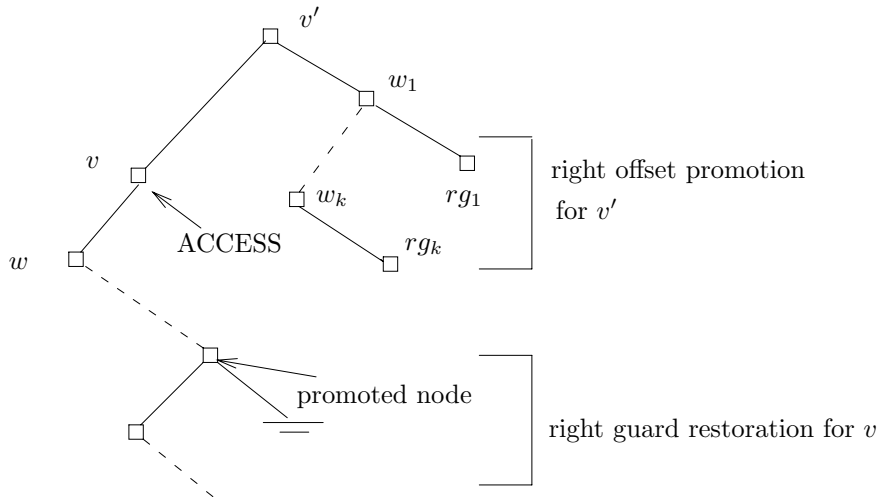


FIG. 21. *Promotions in Case* 2.

process is applied to rg_i for $1 \leq i \leq k$. Also, rg_{i+1} becomes the left guard for the new lazy tree rooted at w_i for $1 \leq i < k$; the tree rooted at w_k uses v as its left guard. If $rg_i = w_i$, for some i, the new lazy tree rooted at w_i comprises only one block; so it is treated as an ordinary block henceforth. We call the promotions described in this paragraph the *right offset promotions* for v.

So Case 1 occasions charges of $2gp \cdot jump(v)$. Recall that v is a node on the right split path.

Case 1.1. v' is a light node which is not on the left extreme path. In addition, v' is accessed from its left child, which is not in the lazy tree.

The right offset promotions for v' are performed (where w_1 is now defined to be the nearest heavy descendant of v'). Let v be the root of the block containing v'. As we will see, v is promoted in Case 2 or Case 3. Thus the cost of Case 1.1 is at most $gp \cdot jump(v)$.

Case 2. v is a heavy node on the left extreme path and it is accessed from its right child, u.

Then v is promoted. v becomes the root of a new lazy tree. v's promotion costs at most $gp \cdot jump(v)$ (apply Corollary 9(i)).

We need to provide a new left guard for the remaining portion of the old lazy tree (see Figure 21.) Let v' be v's parent in the lazy block tree. We perform the right offset promotions for v'. In addition, if v' is not the root of the lazy tree, v' is promoted. v' becomes the root of an ordinary block. If v' is not the root of the lazy tree, rg_1 becomes the left guard for the remainder of the old lazy tree, while if v' was the root, then rg_1 did not need promoting as it was already the right guard of the lazy tree. In the imaginary tree, v''s lazy weight is at least its normal weight; so its promotion comes for free. As in Case 1, $gp \cdot jump(v')$ suffices to pay for the right offset promotions.

The promotion of v' and the associated right offset promotions are also performed if just the left guard, lg, is separated from the remainder of the lazy tree (through being accessed from its right child), and then v' is defined to be the leftmost node in the lazy block tree (note v' will be traversed in the access). This applies even if lg is

the root of the large lazy tree.

We need to provide the new lazy tree rooted at v with a right guard. So let x be the rightmost heavy node in v's lazy tree which is a right descendant of v in the imaginary tree (possibly $x = v$). There are two possibilities to consider.

(i) $x \neq v$. Temporarily, x becomes the right guard for the new lazy tree rooted at v. In the imaginary tree, x's right subtree contains no nodes from its large lazy block tree. So by Invariant 12(ii), the lazy potential for x plus its reserve potential is at least its global potential; thus the promotion can be paid for at no extra charge. Now, the right guard restoration process is applied to x. This has zero amortized cost.

(ii) $x = v$. Let w be v's left child in its new lazy block tree (if v does not have such a child, then the new lazy tree rooted at v comprises only one node and can be treated as an ordinary block henceforth). w is promoted, becoming the root of a new lazy tree while v becomes its right guard. Next, if w's right subtree in the large lazy block tree is nonempty, the right guard restoration process is applied to v. w's promotion is paid for as follows. w adds its reserve potential to its lazy potential; w's modified lazy potential is at least gp times the global rank of v's right child in the imaginary tree (for by Invariant 12(ii), this modified lazy potential is at least $gp \cdot lazy\_\mathrm{rank}(r\_n(v))$ and $r\_n(v)$ must be to the right of the accessed item (as $x = v$); so $lazy\_weight(r\_n(v))$ includes the weight of all of v's right subtree in the imaginary tree; now recall that $lazy\_\mathrm{rank}(r\_n(v)) \geq \log lazy\_weight(r\_n(v))$). Hence $gp \cdot jump(v)$ suffices to pay for this promotion.

Overall, Case 2 occasions a charge of $2gp \cdot jump(v) + gp \cdot jump(v')$. Note that v is on the left split path and v' is on the right split path.

Case 2.1. $\tilde{v}$ is a light node on the left extreme path; it is accessed from its right child, which is not in the lazy tree.

Let v' be the nearest heavy ancestor of v in the lazy tree. v' is treated as in Case 2. Let v be the nearest heavy descendant of $\tilde{v}$ on the left extreme path. v is promoted. v's promotion costs at most $gp \cdot jump(\tilde{v})$ (strictly speaking $jump(\tilde{v})$ was not defined; it is the jump in g_rank on accessing node $\tilde{v}$), which is at least $g\_\mathrm{rank}(v)$. As in Case 2, we need to provide the new lazy tree rooted at v with a right guard. This is handled as in Case 2.

Hence, overall, Case 2.1 occasions a charge of $2gp \cdot jump(\tilde{v}) + gp \cdot jump(v')$. Note that v is on the left split path and v' is on the right split path.

Case 3. The inserted item is to the right of the lazy tree root, r.

Then we need to provide the new lazy tree rooted at r with a new right guard. The method followed is identical to that used in Case 2 for providing the new lazy tree rooted at v with a right guard. So Case 3 occasions a charge of $gp \cdot jump(r)$. Note that r is on the left split path. These promotions are performed even if only the right guard, rg, in the lazy tree is separated from the rest of the lazy tree (through being accessed from its left child). This applies even if rg is the root of the large lazy tree.

LEMMA 10. *The promotions in a global insertion cost at most* $4gp \log n$ *units, where for each split lazy tree there is a charge of* $2gp$ *times the increase in global rank along the right split path and of* $2gp$ *times the increase in global rank along the left split path.*

Proof. Charges of up to $2gp \cdot jump(v)$ arise for nodes on the split paths in Cases 1–3 above. Specifically, in Case 1, there is a charge of $2gp \cdot jump(v)$ for node v on the right split path. In Case 1.1, there is a charge of $gp \cdot jump(v)$ for node v on the left split path. In Case 2 there is a charge of $2gp \cdot jump(v)$ for a node v on the left split

path and a charge of $gp \cdot jump(v')$ for a node v' on the right split path; further v' is not charged in Case 1. v may be the node from Case 1.1, but then Case 2(ii) applies and v is only charged $gp \cdot jump(v)$ for Case 2(ii), giving a total charge of $2gp \cdot jump(v)$ for node v. In Case 2.1 there is a charge of $2gp \cdot jump(\tilde{v})$ for $\tilde{v}$, a light node on the left access path; $\tilde{v}$ is not charged in any other case. Finally, in Case 3, there is a charge of $gp \cdot jump(r)$. These are the only charges that occur. □

The following properties of the new lazy trees are helpful; they are readily confirmed by inspection of the split procedure.

PROPERTY 1. *The left path of each lazy tree created by the split has one of the following two forms:*

(i) *It is a subpath (possibly complete) of the left path of the lazy tree before the split.*

(ii) *It comprises light nodes from just one block.*

PROPERTY 2. *The number of promoted nodes from each old lazy tree L is at least the number of lazy trees into which L is split.*

4.3.2. Debits and their invariants. We need to consider the effect of the split and (from the beginning of section 4.3) of the updates to the guards on Invariants 1–4 and 6–7. We show how to reestablish these invariants by removing debits on the extreme paths of the new lazy trees and on the left and right paths of blocks whose roots cease to be part of the lazy tree.

INVARIANT 1. *The small or large debit, if any, is removed from each promoted node. This is charged to the promoted node. Also, each light node that ceases to be an extreme path node, and hence also ceases to be on a lazy tree, pays for the removal of its small or large debit, if any.*

INVARIANT 2. *For each promoted node, the large debits, if any, are removed from its parent and child (if any) on the extreme path. This is charged to the promoted node. (Note that the reestablishment of Invariant* 1 *has already removed the large debit, if any, from itself.)*

INVARIANT 3. *The small debit, if any, is removed from each extreme path node. This is charged to the promoted node.*

INVARIANT 4. *For each promoted node u, the small debits, if any, are removed from the corresponding nodes w, if present. This is charged to the promoted node.*

Note that reestablishing Invariants 1–4 requires each promoted node to pay for the removal of at most two large debits and three small debits, or one large and four small debits, or six small debits. The maximum charge is achieved with two large and three small debits.

INVARIANT 6. *The lazy debit, if any, is removed from each light node whose block is no longer part of a lazy tree. This includes those light nodes whose block roots become guards for a new lazy tree. We call such blocks* ordinary blocks. *This is charged to the node itself. For each block that becomes the leftmost block in a new lazy tree we reduce the potential of each light node from $2c'$ to c'. This pays for the removal of lazy debits from these nodes. Any other light node on a new left path must have been on the old left path and so already satisfied the invariant.*

INVARIANT 7. *Any node violating Invariant* 7 *must be on a new right path. There are three ways for a light node x on a new right path to still violate Invariant* 7.

Case 1. See Figure 22. x is adjacent to node y in block, B_y, where B_y either has become the leftmost block in a new lazy tree or has become an ordinary block. The removal of x's lazy debit is charged to node y.

Case 2. x's right child is in the same lazy tree but is heavy. Let z be the root

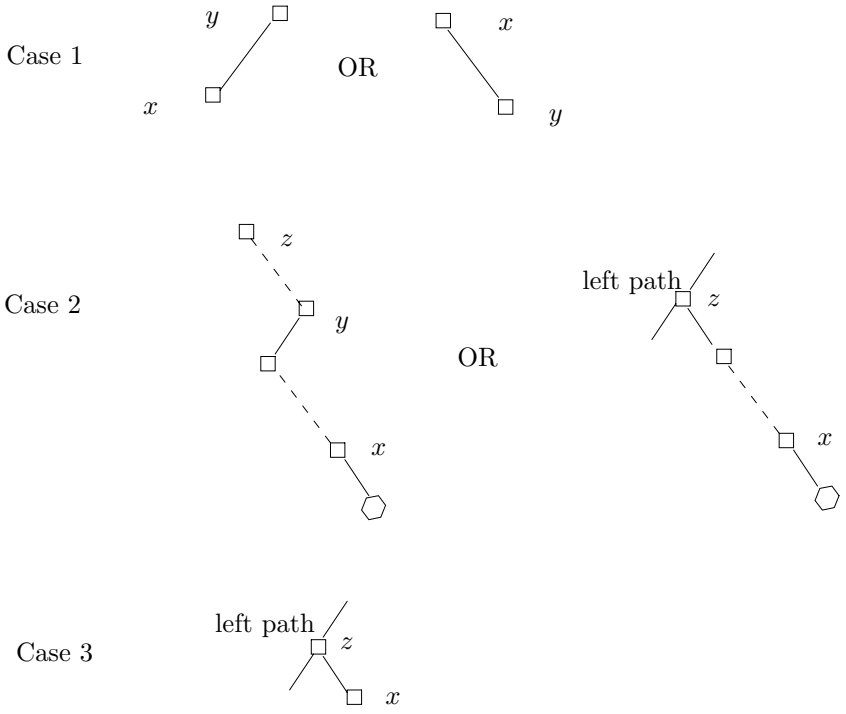


FIG. 22. *Invariant* 7.

of x's block (z is a heavy node also). Let y be the nearest right ancestor of x. If y is between x and z, y must be in a block, B_y, where B_y either has become the leftmost block in a new lazy tree or has become an ordinary block (this follows from Property 1). In addition, for each such node y, there is at most one node x of Case 2, namely the first node, on the right path descending from y's left child, to have a heavy child. The removal of x's lazy debit is charged to node y.

If z had not been on the left extreme path of the lazy tree prior to the split, then there is such a node y, for if not, prior to the split, x would have been on the right path descending from z, and by Invariant 7, would not be carrying a lazy debit.

So if there was no such node y, then z had been on the left path of the lazy tree prior to the split and x was on the right path descending from z in the splay tree. Also, z must be the promoted root of x's new lazy tree. In this case, the removal of x's lazy debit is charged to z.

Note that a promoted node may be charged for the removal of three lazy debits (as node y) or one lazy debit (as node z) but not both.

Case 3. x is the right child of its parent z, which was on the left path of the lazy tree prior to the split. Then z pays for the removal of the lazy debit from x. So z may have to pay for the removal of up to two lazy debits (the first such debit was removed in Case 2).

LEMMA 11. *Following a split, Invariants* 1–4 *and* 6–7 *concerning the debits can be restored provided the following two equations hold:*

$$c' \geq \max\{4hd, ld + 3hd\} = 4hd, \tag{20}$$

$$b \geq 3hd + 2ld + 3sd + gp. \tag{21}$$

Proof. Each light node in a new leftmost or ordinary block is charged for the removal of at most four lazy debits (from itself and the nodes for which it restored Invariant 7) or one small or large debit and three lazy debits (removed from the same nodes). This is charged to the reduction of at least c' in the node's potential. (Recall that each light node in a leftmost block has potential c' and all other light nodes have potential $2c'$.) So it suffices to have (20) hold.

Each promoted heavy node is charged for the removal of at most three lazy debits (from the nodes for which it restored Invariant 7), three small debits and two large debits; it may also have to pay a segment charge. (For the split is the result of a global insertion. Also see (3) and the paragraph preceding (3).) These costs are charged to the promoted node's potential of b units (which it received as a heavy node when its lazy tree was created). So it suffices to have (21) hold. □

4.3.3. Recreating the lazy complete tree potential. We continue by showing how to reestablish Invariants 8–10. To restore Invariant 8, we proceed essentially as in the original creation of active layers. Let h be the maximum creation_height of any node in the normal form of the new lazy tree rooted at u, apart from its root (the creation_heights are those defined with respect to the original tree; they are not redefined with respect to the new lazy tree). Suppose there is no l-active node for some $l \leq h$. Then, in the corresponding normal tree, the lowest node v on the right path whose span includes l becomes l-active (these new active layers are then translated back into the lazy tree at hand). Below, we show that Invariants 8–10 hold once more (incidentally, this implicitly shows that the rule for creating new l-active nodes is well defined).

Clearly Invariant 10 still holds, for if $d_l(v)$ increases, $id_l(v)$ is reduced by an equal amount. Next we consider Invariant 9. In each new lazy tree the black status of the nodes is unchanged; also, by Property 1, the heavy nodes on the new left paths were also previously on left paths. So Invariant 9 still holds.

Next, we show Invariant 8 is reestablished by the creation of new active layers. First, we consider the situation prior to the creation of new active layers. Clearly, Invariant 8 (iii) and (v) still hold; (iv) holds likewise, since the symmetric order of the nodes in the normal form of each new lazy tree is unchanged. We note that each new lazy tree has a span of active layers, possibly empty, of the form $(i, h]$, $0 \leq i \leq h$, where h is the largest creation_height (as provided initially) of any node in the normal form of the new lazy tree, apart from the root. By Invariant 8 (v), (iii), and (iv), the right path in each new normal lazy tree, from top to bottom, contains a sequence, possibly empty, of active nodes with decreasing creation_heights. It is now readily seen that the rule for creating new active layers restores Invariant 8(i)–(ii).

4.3.4. The invariants defined for the split operation. We note that the split operation has been defined so as to maintain Invariant 13. It remains to consider the effect of the split on Invariant 12; but the only effect of a split is to reduce the size and hence weight of a lazy tree; so this Invariant also continues to hold.

4.3.5. Summary and generalization. We have shown (see Lemma 10 and (5)) the following lemma.

LEMMA 12. *The cost of a global access is at most* $e \cdot (3 \log n + 1)gp + 5 \log n \cdot gp$ *units.*

In a later paper we will again use lazy trees which will have heavy and light nodes. The heavy nodes will have the same lazy and reserve potentials as here. It is useful

to summarize and slightly generalize the results of this section.

As already noted, the nodes of the lazy trees in the later paper will still have small, large, and lazy debits which obey the present invariants, namely Invariants 1–4 and 6–7. Other additional debits satisfying Invariant 11 may be present. We need to show how to maintain this invariant following a lazy tree split. It is readily seen that the method for restoring the invariants concerning lazy debits suffice to restore this invariant also, so it suffices to replace hd by md in (20) and (21), yielding

$$c' \geq \max\{4md, ld + 3md\} = 4md, \tag{22}$$

$$b \geq 3md + 2ld + 3sd + gp. \tag{23}$$

The debits may be restricted beyond the invariants of this paper, so long as these new constraints satisfy the following property.

PROPERTY 3. *The restoration of any further invariant concerning the debits requires the removal of at most α additional debits from each lazy tree created by the split, α a constant.*

By Property 2 there are at least as many promoted nodes as there are lazy trees created by the split. Thus the sets of α additional debits can be charged α to each promoted node. Modifying (23) as follows,

$$b \geq 3md + 2ld + 3sd + gp + \alpha \cdot md, \tag{24}$$

suffices to ensure all the invariants can be restored.

We have shown the following lemma.

LEMMA 13. *If a lazy tree comprises light and heavy nodes where the heavy nodes carry lazy potentials as specified in Invariant* 12*, then each lazy tree resulting from a split satisfies the following:*

(i) *Invariant* 12;

(ii) *If the nodes on its left path are new to a left path, then they are all light nodes in the same block; and*

(iii) *Assuming the debits obey Invariants* 1–4 *and* 6–7*, and* 11*, and assuming* (22) *and* (24) *hold, then these invariants can be restored following a split.*

Clearly, we also need the following.

PROPERTY 4. *The potential must be partitionable following a split; i.e., each lazy tree created by the split must be provided with an appropriate potential (e.g., a lazy complete tree potential).*

4.4. Lazy trees and local accesses. We need to reconsider the analysis of local accesses to take the presence of lazy trees into account (see Remark 2). Note that in such an access only the left paths of lazy trees can be traversed. In fact, only two modifications are needed. First, we need to consider couples comprising two heavy nodes. But these are analyzed as in a global insertion: They have $0 \leq s + 1$ amortized cost, for as noted in Remark 3, no spares are spent on paying for changes to the j-potentials and/or j-black potentials. Second, we may need to modify the guards of some lazy trees so as to maintain Invariant 13. This would result in the left guard restoration process being applied, which, as in section 4.3, does not occasion a change to the analysis of the insert operation. So the results of Lemma 6 continue to apply.

4.5. Multiple level lazy trees. It is convenient to refer to the lazy trees encountered so far in the paper as *right lazy trees*. Analogous lazy trees, called *left*

lazy trees, in which the roles of left and right are interchanged are considered in this section. While they are not needed to prove the result of this paper, they lead to a more general result in this section, without adding significantly to the complexity of the current section.

Because a local access path may include nodes of a current lazy tree we may seek to make the root of a lazy tree a heavy node carrying a lazy potential in a new lazy tree. We therefore generalize the form of the lazy trees. Now, a "block," or rather metablock, in a lazy tree may itself be another lazy tree.

In order to distinguish the ages of the different lazy trees, we number the blocks, in insertion order, by $1, 2, \ldots$. A lazy tree is labeled by the number of its corresponding creating block. When a lazy tree is split its parts keep the same age label.

Let L be a lazy tree; its skeleton plus its lazy block tree comprise the nodes on L. The nodes on L are also said to *belong* to L. A node may be on an extreme path of several lazy trees. For each lazy tree to which a node belongs it carries a separate potential. However, a node may carry only one debit, as before. Invariants 1–4 and 6–7 should be interpreted with respect to the newest lazy tree to which the node carrying the debit belongs.

We define a new lazy tree, L_{new}, to *contain* an old lazy tree, L_{old}, if the root, r, of L_{old} is on L_{new}. We write $L_{old} \subset L_{new}$. If there is no tree L_{mid} with $L_{old} \subset L_{mid} \subset L_{new}$, then L_{old} is treated as a metablock of L_{new}. The root of L_{old} is treated as the root of this metablock; the root of L_{old} is a heavy node on L_{new}. All other nodes on L_{old} are light nodes of this metablock, unless they are not even on L_{new}. This matters when defining the potentials for nodes on L_{new}. Suppose that L_{old} is contained in L_{new}; then, apart perhaps for its root, any node on L_{old} that is also on L_{new} must be a light node on L_{new}; in fact, only the root and nodes on an extreme path of L_{old} can be on L_{new}, but it need not be the case that even all these nodes are on L_{new}.

In addition, each guard of a lazy tree may be the root of another lazy tree, rather than being the root of a block. More precisely, suppose that g is the right (resp., left) guard for lazy tree L on creation of L, and L_{old} is the newest lazy tree rooted at g at this time (L_{old} is older than L). Then the root of L_{old} remains the right (resp., left) guard for L until one or both of L and L_{old} are split. Note that the root of L_{old} may at some point become the root of a newer lazy tree, L_{new}; however, the root of L_{new} does not take over the guard role for L.

In order to carry over the previous analysis of an extreme path traversal, we require the following properties concerning lazy trees to apply.

Requirement 1. Suppose u is traversed in the current insertion. Let L be the newest lazy tree to which u belongs.

(i) Let v be an ancestor of u on L. Then L is also the newest lazy tree to which v belongs.

(ii) u is either the root of L, or a heavy or light node of L; if a heavy node it carries a lazy potential defined with respect to L.

(iii) If u is a heavy node of an older lazy tree, then it is a light node of L.

Items (i) and (ii) enable the traversal of node u to be treated as part of the traversal of an extreme path of L; the previous analysis continues to apply. However, a new issue arises because u may also belong to older lazy trees. We maintain u's potentials with respect to each such lazy tree. Maintaining the l-potentials and l-black potentials might appear problematic, for it may involve the spending of k-spares, $k \geq 1$. But we note that each node, u, is heavy in at most one lazy tree,

and this is the only lazy tree with respect to which u carries l-potentials and l-black potentials and to whose heavy nodes it transfers spares (for in all newer lazy trees that contain u, u is light). So there is only one lazy tree with whose l-potentials and l-black potentials u is concerned, and this is the only lazy tree to which u contributes spares or from which u draws spares. Thus the l-potentials and l-black potentials can be maintained as before. Otherwise, the analysis of an extreme path traversal is unchanged, apart possibly for the removal of debits following the maintaining of Invariant 13. We discuss this at the same time as we show how to remove debits following the new split operation.

The following invariant characterizes the overlap of lazy trees.

INVARIANT 14.

(i) *Let L_a and L_b be two lazy trees of the same age. Then the two open intervals defined, respectively, by the guards of L_a and of L_b are disjoint.*

(ii) *Let L_{old} be a lazy tree and let r be its root. Let L_{new} be another, newer, lazy tree. Then the following hold:*

  (a) *If r lies strictly between the guards of L_{new}, then the guards of L_{old} lie between the guards of L_{new}. In addition, let u and v be the items in the large lazy block tree for L_{new} straddling r (r may or may not be an item in this large lazy block tree). Then L_{old} plus its guards lies between u and v. (Recall that the large lazy block tree for lazy tree L comprises the lazy block tree for L plus the guards for L.)*

  (b) *If r is strictly outside the closed interval defined by the guards of L_{new}, then the open intervals defined by the guards of L_{old} and L_{new}, respectively, are disjoint.*

  (c) *Suppose that r is the right (resp., left) guard for L_{new}. Let d_{new} be the rightmost (resp., leftmost) item in the lazy block tree for L_{new}. Then the left (resp., right) guard of L_{old} is either equal to or to the right (resp., left) of d_{new}.*

Thus, in some sense, an older lazy tree is either contained in a newer lazy tree or is disjoint from it. In the next two lemmas we show several consequences of this invariant.

LEMMA 14. *Requirement* 1(i) *holds.*

Proof. Let u and v be as in Requirement 1(i). Suppose that v belongs to lazy tree L_{new}, newer than L. Then L and L_{new} obey Invariant 14(ii)(a). By assumption, only an extreme path of L_{new} is traversed; so the inserted item lies outside the range spanned by the guards of L_{new}; without loss of generality, suppose that the inserted item is to the right of the right guard of L_{new}. By Invariant 14(ii)(a), all of L is to the left of L_{new}'s right guard; so the only items of L that are traversed must also be on L_{new}. □

LEMMA 15. *Let w be on lazy tree L_{old}. Suppose w is also on a newer lazy tree L_{new}. Then w is on an extreme path of L_{old}.*

Proof. Invariant 14(ii)(a) applies to L_{new} and L_{old}. Let u and v be the items of L_{new} straddling the root of L_{old}, as in the statement of the invariant. Since w is on L_{new}, w is on the path from u to v in the range (u, v) (if $u < v$) or (v, u) (if $v < u$). If w does not lie on an extreme path of L_{old}, then some item of L_{old} must lie outside the range (u, v) (or (v, u)), which contradicts Invariant 14(ii)(a). □

INVARIANT 15. *Let L be a lazy tree.*

(i) *The nodes of L, apart from its root, are all heavy or light on L.*

(ii) *Apart possibly from its root, all of L's heavy nodes carry their lazy potential.*

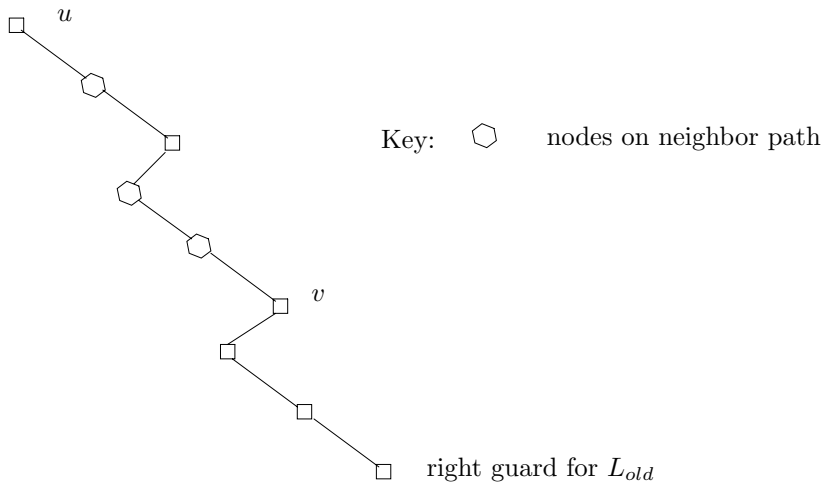


FIG. 23. *A (u, v)-neighbor path.*

(iii) *Each of L's light global nodes is a heavy node in some older lazy tree.*

(iv) *Each node, v, heavy on some lazy tree L, is a light node of all the newer lazy trees to which it belongs.*

Invariant 15 implies Requirement 1(ii) and 1(iii), above.

Before giving the next invariant, a few definitions are helpful. Let L be a lazy tree and let u be a node of its large lazy block tree. v is an *L-neighbor* of u if v is also a node of the large lazy block tree and u and v enclose no other node of the large lazy block tree. Suppose that u is an ancestor of its L-neighbor v; *the (u, v)-neighbor path* comprises those items on the path from u to v in the splay tree which are in the range (u, v), if $u < v$, or (v, u), if $v < u$.

INVARIANT 16. *See Figure* 23. *Let L be a lazy tree. Let u and v be L-neighbors, with u the ancestor of v. Let N denote the (u, v)-neighbor path. If N includes a global node, there is a lazy tree L_{old}, older than L, rooted at u, such that every node in N is on L_{old}. Further suppose that u is a left (resp., right) ancestor of v. Then the right (resp., left) guard for L_{old} is either v or a left (resp., right) descendant of v.*

By inspection plus induction, Invariants 14–16 are true on creation of a lazy tree L_{new}; also, they remain true as the extreme paths of the lazy trees are traversed.

We need to reconsider the analysis of splits. Again, in turn, we consider the promotions, the consequential removal of debits and the recreation of the complete lazy tree potential for the lazy trees created by the split. Actually, the arguments concerning Invariants 8–10 and 12–13 are unchanged from section 4.3 and so we will not discuss further the recreation of the complete lazy tree potential.

4.5.1. The promotions. For each split lazy tree, our goal in a split is to promote the same nodes as in section 4.3 (remember, this needs to be interpreted symmetrically for left lazy trees); however, this may prove too expensive because of the recursive containment of lazy trees. So, sometimes, instead of promoting a node v, heavy on lazy tree L, we will promote a node w which is light on L but heavy on an older lazy tree, and such that w is an ancestor of v.

We consider the promotions to be performed on one lazy tree at a time, oldest

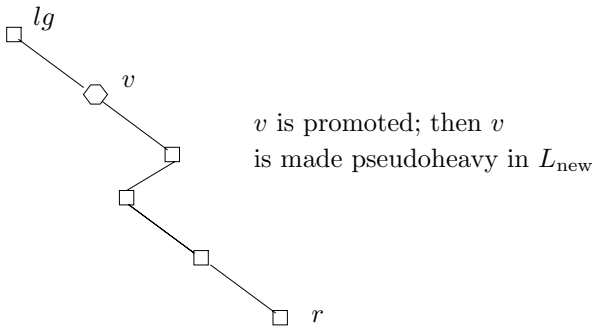


FIG. 24. *Pseudoheavy nodes.*

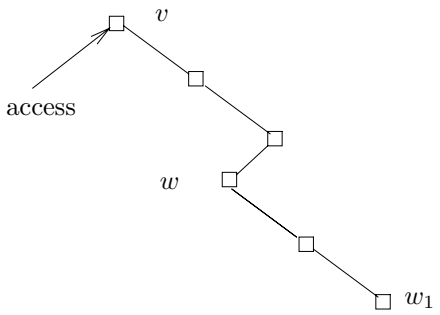


FIG. 25. *Case* 1′.

first. Invariants 14 and 15 are maintained but modified as follows.

Suppose the promotions have been applied to L_{old} but not to L_{new} (see Figure 24). All the promoted nodes which are on L_{new} are made into pseudoheavy nodes of L_{new}. In addition, let lg be the left guard of L_{new} and r its root. Suppose that lg is an ancestor of r. Let v be a node on the path from lg to r in the range (lg, r). If v is promoted, then v also becomes a pseudoheavy node of L_{new}. A similar rule applies with respect to the right guard of L_{new}.

INVARIANT 17. *Invariant* 14(ii)(a) *holds with respect to the heavy and pseudoheavy nodes of* L_{new}*, as do Invariants* 15(i) *and* 16.

Next, we explain how the previous promotion procedure is modified. Consider the promotions on right lazy tree L.

Case 1′. See Figure 25. v is a heavy node in the lazy block tree, which is not on the left extreme path. In addition, v is accessed from its left child, u.

v is promoted as in Case 1 of section 4.3. w_1 is defined as in this Case 1. The right offset promotions for v are performed exactly when no global node on the path from v to w_1, in the range (v, w_1) has been promoted. If such a node has been promoted, there will be one such node; let it be denoted w. Then w becomes a root of a lazy tree formed from L; its left subtree is empty and its right subtree contains exactly the heavy nodes of L in v's right subtree. v provides the left guard for this new lazy tree. The right guard for the lazy tree rooted at w is obtained in the same way as w_1's right guard in Case 1.

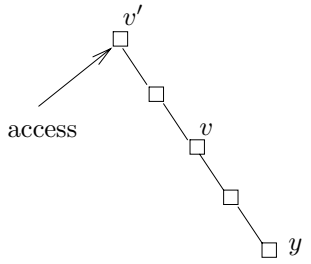


FIG. 26. *Case* 1.1′.

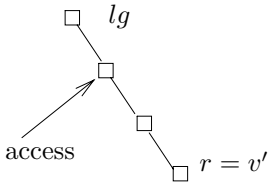


FIG. 27. *Separating the left guard.*

Case 1.1′. See Figure 26. v' is a light node of L, which is not on the left path of L, and v' is accessed from its left child, which is a node not on L.

Suppose v' is a heavy node on some older lazy tree, L_{old}; then v' is on the right path of L_{old} (by Lemma 15) and so v' was already promoted. Suppose that v' is straddled by L's heavy nodes x and y, where y is a descendant of v'. Let v be the promoted node nearest to y on the path from v' to y, in the range $[v', y)$, if any. If $v \neq v'$, v is made the root of a lazy tree formed from L; its left guard is the nearest node on the path from v' to v, in the range $[v', v)$ which has been promoted (in fact, this must be v'). Its right guard is handled as in Case 1. While if $v = v'$ or if v' was not heavy on any older lazy tree, then the right offset promotions for $v = v'$ are performed.

Case 2′. v is a heavy node on the left extreme path and it is accessed from its right child, u.

v' is defined to be the nearest heavy or pseudoheavy ancestor of v; v' must be on the left external path of L. v and v' are promoted as in Case 2 of section 4.3. As in Case 1′, the *right offset promotions* for v' are performed exactly when no global node on the path from v' to w_1, in the range (v', w_1) has been promoted. Again, if the right offset promotions are not performed, a new lazy tree is created exactly as in Case 1′.

If the left guard, lg, is separated from the remainder of L (through being accessed from its right child), and, v', the leftmost heavy node in L, is traversed, then the promotions described in the previous paragraph are performed; but if lg is separated from the remainder of L without v' being traversed (which can arise only if lg is the root of the large lazy block tree) then the new left guard for L is provided by the promoted node, if any, nearest to the root, r, of L, in the range (lg, r), on the path from lg to r. (See Figure 27.) If there is no such promoted node, as in Case 1.1′, the right offset promotions for r are performed.

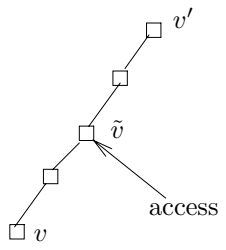


FIG. 28. *Case* 2.1′.

Again, we need to provide the new lazy tree rooted at v with a right guard. As in Case 2 of section 4.3, let x be the rightmost heavy descendant of v in the imaginary tree associated with its lazy block tree. If $x \neq v$, we proceed as in Case 2 of section 4.3. If $x = v$, we proceed as follows. Let w' be v's left child in the lazy block tree for L (if v does not have such a child then the lazy block tree rooted at v will comprise only one node and so can be ignored henceforth). Let w be the first node on the path from v to w', in the range (w', v), which has already been promoted, if any (there will be at most one promoted node on this path). If there is no such node, set $w = w'$ and promote it. w becomes the root of a new lazy tree formed from L with right guard v. Finally, if w has a nonempty right subtree in L, the right guard restoration process is applied to v.

Case 2.1′. $\tilde{v}$ is a light node on the left path of L and $\tilde{v}$ is accessed from its right child (a node which is not on L). See Figure 28.

v' is defined to be the nearest heavy or pseudoheavy ancestor of $\tilde{v}$; v' must be on the left external path of L. v' is handled as in Case 2′, above. Suppose $\tilde{v}$ is a heavy node on some older lazy tree, L_{old}. then $\tilde{v}$ must be on the left path of L_{old} (by Lemma 15), so $\tilde{v}$ was already promoted. $\tilde{v}$ will become the right guard of a new lazy tree formed from L. If a left descendant, w, of $\tilde{v}$, on the left path of L has already been promoted (there can be at most one such node) then w becomes the root of this new lazy tree. If not, w is defined to be the first heavy node on the left path of L which is a proper descendant of $\tilde{v}$; it is promoted. In addition, if w has a nonempty right subtree in L, the right guard restoration process is applied to $\tilde{v}$. Finally, suppose that $\tilde{v}$ was not heavy on any older lazy tree. Let v be the nearest descendant node on the left path of L which is heavy or pseudoheavy. v is promoted if it is not already promoted. It is then handled in the same way as node $\tilde{v}$ earlier in this case.

The charging for the promotions is analyzed shortly. The following lemmas are helpful.

LEMMA 16. *Let v be a node on lazy tree L accessed from its left child and promoted in Case* 1′ *or* 1.1′ *(as node v' in the latter case). Suppose that w' is a promoted node in v's right subtree, in addition, suppose that v and w' are on an older lazy tree L_{old}, where L_{old} is also split by the access. Let $w_1, \ldots, w_k$ be the nodes on L, if any, promoted by the right offset promotions. Then w' is a proper left descendant of w_k.*

Proof. Note that as w' is on L_{old}, an older lazy tree including v, w' must be in the range (v, w_k), by Invariant 14(ii)(a). Suppose, for a contradiction, that w' has an ancestor x on the path from v to w_1 in the range (v, w_1). See Figure 29. (If this is not the case, then w' is a proper left descendant of w_k.) x is global as both w' and

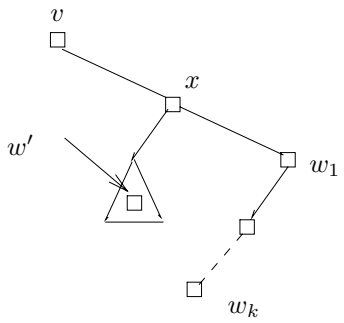


FIG. 29. *Proof of Lemma* 14.

w_1 are global. If x is on L_{old}, x is heavy on L_{old}, for x and its parent are separated, in left to right order, by global w'. Let x' be the first node on the path from v to x, in the range $(v, x]$, on L_{old}, which is heavy on L_{old}, or which is already promoted. Then unless already promoted, x' is promoted when L_{old} is split. (If Case 2(i) were to apply to L_{old}, then x' would not be promoted, as claimed; but, by assumption, w' is promoted; to achieve this promotion, Case 2(ii) needs to be applied, and then x' is indeed promoted.) But if x' is promoted, then w_1 would not be promoted, a contradiction. If x is not on L_{old}, then x is heavy on some tree L' sandwiched in age between L_{old} and L. Again, on replacing L_{old} by L' in the above argument, a contradiction is obtained. □

The next lemma is shown by the same argument as used in Case 1 in section 4.3.

LEMMA 17. *Let $w_1, \ldots, w_k$ be the chain of nodes promoted on lazy tree L by the right offset promotions for node v. In addition, let w' be a promoted node, if any, in v's right subtree on an older lazy tree L_{old}. $lazy\_\text{rank}(w_i) \geq g\_\text{rank}(w_{i+1})$ for $i = 1, \ldots, k-1$; similarly, $lazy\_\text{rank}(w_k) \geq g\_\text{rank}(w')$. Finally, $lazy\_\text{rank}(w_k) + jump(v) \geq g\_\text{rank}(v)$.*

In addition, we note the following.

LEMMA 18. *Let w be a node on lazy tree L promoted in Case 2′ or 2.1′. Let w' be a promoted node in v's left subtree on an older lazy tree L_{old} that is also split by the access. Then*

(i) *w' is a right descendant of w;*

(ii) *$lazy\_\text{rank}(w) \geq g\_\text{rank}(w')$;*

(iii) *$lazy\_\text{rank}(w) + \frac{1}{gp} reserve(w) + jump(v) \geq g\_\text{rank}(v)$.*

Proof. The proof of (i) is very similar to the proof of Lemma 16. (ii) follows from Corollary 8(i). The proof of (iii) uses the argument of Case 2 in section 4.3. □

Case 3′. The inserted item is to the right of the lazy tree root, r.

The new lazy tree rooted at r must be provided with a right guard. The method followed is identical to that used in Case 2′ for providing the new lazy tree rooted at v with a right guard. Lemma 18 applies here too.

It remains to analyze the cost of the split operations. Clearly, for the nodes on the right (resp., left) access path, the cost of the promotions is at most $gp \cdot \log n$, a total of $2gp \cdot \log n$ units for the two access paths. Lemmas 17 and 18 imply that for each promoted node on the right (resp., left) access path the associated promoted nodes form a chain descending to the left (resp., right); let $x_1, x_2, \ldots, x_k$ be a maximal path of such nodes, in descending order. Lemmas 17 and 18 show that $lazy\_\text{rank}(x_i) \geq$

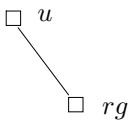


FIG. 30. *Guard restoration.*

$g\_\text{rank}(x_{i+1})$, for $i = 1, \ldots, k-1$ and $lazy\_\text{rank}(x_k) + \frac{1}{gp} reserve(x_k) + jump(v) \geq g\_\text{rank}(v) \geq g\_\text{rank}(x_1)$. (Note that the applications of Lemmas 17 and 18 will be interleaved if there is an interleaved sequence of successively older left and right lazy trees.) We conclude that a further $2gp \cdot \log n$ units of potential pay for the promotion of the nodes x_i.

So, as before, the cost of the split operation is at most $4gp \log n$ and Lemma 10 continues to hold.

LEMMA 10. *The promotions in a global insertion cost at most* $4gp \log n$ *units, where for each split lazy tree there is a charge of* $2gp$ *times the increase in global rank along the right split path and of* $2gp$ *times the increase in global rank along the left split path.*

It should be clear that Invariants 14–17 are maintained over the course of these promotions. When the promotions are complete there are no pseudoglobal nodes remaining, so at this point Invariants 14–16 will have been restored.

Finally, we need to consider the effect of the guard restoration process. We claim that if the guard restoration process is applied to lazy tree L, for each of the promoted nodes the newest lazy tree to which it belonged was L. Then it is easy to see that Invariants 14–16 continue to hold. To show the claim we demonstrate one case; the others are similar. Suppose that rg, the right guard of L, is in a couple with heavy node u of L (see Figure 30). Suppose that u is also on a newer lazy tree L_{new}. Then u is light on L_{new}. If rg is light on L_{new}, then following the rotation, u is no longer on L_{new}. If rg is heavy on L_{new}, then by Invariant 14(ii)(a) applied to L and L_{new}, there are no heavy nodes of L_{new} in rg's left subtree nor in u's left subtree; so, again, following the rotation, u is no longer on L_{new}. A similar argument applies if rg is the right guard of L_{new}.

4.5.2. Removing debits. We show how to restore Invariants 1–4 and 6–7. For each node that was on a newest lazy tree L but is not on a lazy tree derived from L, we charge the removal of its debit, if any, to the node's potential associated with lazy tree L. So we are only concerned with nodes for which the age of their newest lazy tree is unchanged. For such nodes, the invariants are all restored exactly as in section 4.3.

Restoring Invariants 1–4 result in the same maximum charge of three small debits and two large debits to a promoted node; this holds regardless of how many lazy trees the promoted node belonged to. Debit removal charged to the node relinquishing the debit is treated as before; this takes care of Invariant 6 as well. It remains to consider Invariant 7.

For a promoted node to be charged as node z of the argument in section 4.3, it must be a heavy node on the left (resp., right) extreme path of its right (resp., left) lazy tree prior to the split. Likewise, for a node to be charged as node y, it must be a heavy node on its right (resp., left) lazy tree prior to the split and not on the left (resp., right) extreme path. So as a promoted node, a node is charged no more than

before. The same node may receive additional charges, but as light nodes on other newer lazy trees; these charges are paid for by the potentials associated with the node as a light node in these newer lazy trees. So as before, (20) and (21) suffice to enable Invariants 1–4 and 6–7 to be restored.

Again, for the more general case, to restore Invariant 11 (22)–(24) suffice.

To satisfy any further restrictions obeying Property 3, we need to cover a charge of up to $\alpha \cdot md$ for each lazy tree created by the split. For each such lazy tree, the charge is given to its root. However, this cannot be applied solely to the charge made to promoted nodes, as several new lazy trees may share a common root. Instead, the charge is made to each node, either as a light node or for the oldest lazy tree, as a heavy node. This requires modifying (23) to (24) as before, and modifying (22) as follows:

$$c' \geq 4md + \alpha \cdot md. \tag{25}$$

4.5.3. Summary. The presence of multiple levels of lazy trees does not alter the previous analysis of a global insertion, stated in Lemma 12 (except that the constraint of (25) is introduced). The analysis of local insertions is also unaffected, for the discussion of section 4.4 carries over unchanged; so Lemma 6 also continues to hold. In summary, we have shown the following.

LEMMA 19. *The cost of a local access is* $c + s + 1$*, and the cost of a global access is* $e(3 \log n + 1)gp + 5gp \log n$ *units.*

In addition, we note that Lemma 13 continues to hold with (22) replaced by (25) (but now in Lemma 13(ii), "block" is interpreted to mean the newest lazy tree older than the lazy tree in question, to which nodes on the left path belong).

5. Concluding the analysis. We choose $s = 1$. Recall that $s' = s$ (see section 4.2). By Lemma 3 we have $sd = 2$, $gp = ld = 34$, $c = 36$. On taking equality in (7), (10), (13), (14), (20), and (21), we obtain $hd = 36$, $d = 432$, $b = 216$, $c' = 144$, $a = 2 \cdot 216^2 = 93,212$ (incidentally, these values satisfy (6), (8) and (9)) and $q + 4 \leq 2^{21}$; by (4) it suffices to set $e = 22$. For each block we need to pay for one global insertion and $\log n - e$ local insertions (note that the cost of the first $e - 1$ local insertions is covered by the charge for the global insertion). Equations (15)–(19) and (22)–(25) are not used here. From Lemma 19, we conclude the following.

THEOREM 20. *The number of rotations for sorting a* $\log n$*-block sequence is bounded by* $2500n + O(n/\log n)$*. (Recall that the number of rotations dominates the overall cost of the sort.)*

Acknowledgments. We thank Rajamani Sundar for his very careful reading of the paper which uncovered several serious omissions in an earlier draft. We also thank S. Muthukrishnan for his help and advice regarding figure drawing. Finally, we thank an anonymous referee for making a number of detailed suggestions that helped improve the paper's presentation.

REFERENCES

[C00] R. COLE, *On the dynamic finger conjecture for splay trees. Part* II: *The proof*, SIAM J. Comput., 30 (2000), pp. 44–85.

[Luc88a] J.M. LUCAS, *Arbitrary Splitting in Splay Trees*, Technical report DCS-TR-234, Computer Science Department, Rutgers University, Piscataway, NJ, 1988.

[Luc88b] J.M. Lucas, *Canonical forms for Competitive Binary Search Tree Algorithms*, Technical report DCS-TR-250, Computer Science Department, Rutgers University, Piscataway, NJ, 1988.

[ST85] D.D. Sleator and R.E. Tarjan, *Self-adjusting binary search trees*, J. ACM, 3 (1985), pp. 652–686.

[STT86] D.D. Sleator, R.E. Tarjan, and W.P. Thurston, *Rotation distance, triangulations, and hyperbolic geometry*, J. Amer. Math. Soc., 1 (1988), pp. 647–681.

[Su89] R. Sundar, *Twists, turns, cascades, deque conjecture, and scanning theorem*, in Proceedings of the 13th Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 555–559.

[T85] R.E. Tarjan, *Sequential access in splay trees takes linear time*, Combinatorica, 5 (1985), pp. 367–378.

[W86] R. Wilber, *Lower bounds for accessing binary search trees with rotations*, SIAM J. Comput., 18 (1989), pp. 56–67.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 44–85

ON THE DYNAMIC FINGER CONJECTURE FOR SPLAY TREES. PART II: THE PROOF*

RICHARD COLE†

Abstract. The following result is shown: On an n-node splay tree, the amortized cost of an access at distance d from the preceding access is $O(\log(d+1))$. In addition, there is an $O(n)$ initialization cost. The accesses include searches, insertions, and deletions.

Key words. binary search tree, splay tree, amortized analysis, finger search tree

AMS subject classifications. 68P05, 68P10, 68Q20, 68Q25

PII. S009753979732699X

1. Introduction.

The reader is advised that this paper quotes results from the companion Part I paper [CMSS00]; in addition, the Part I paper introduces a number of the techniques used here but in a somewhat less involved way.

The splay tree is a self-adjusting binary search tree devised by Sleator and Tarjan [ST85]. They showed that it is competitive with many of the balanced search tree schemes for maintaining a dictionary. Specifically, Sleator and Tarjan showed that a sequence of m accesses performed on a splay tree takes time $\mathrm{O}(m \log n)$, where n is the maximum size attained by the tree ($n \leq m$). They also showed that in an amortized sense, up to a constant factor, on sufficiently long sequences of searches, the splay tree has as good a running time as the optimal weighted binary search tree. In addition, they conjectured that its performance is, in fact, essentially as good as that of any search tree. Before discussing these conjectures, it will be helpful to review the operation of the splay tree and the analysis of its performance. The basic operation performed by the splay tree is the operation *splay*(x) applied to an item x in the splay tree. *splay*(x) repeats the following step until x becomes the root of the tree.

> SPLAY STEP.
> Let p and g be, respectively, the parent and grandparent (if any) of x.
> CASE 1. p is the root: Make x the new root by rotating edge (x, p).
> CASE 2—THE ZIG-ZAG CASE. p is the left child of g and x is the right child of p, or p is the right child of g and x is the left child of p: Rotate edge (x, p), making g the new parent of x; rotate edge (x, g).
> CASE 3—THE ZIG-ZIG CASE. Both x and p are left children, or both are right children: Rotate edge (p, g); rotate edge (x, p).

Henceforth, we refer to the rotation, single or double, performed by the splay step as a *rotation* of the access or splay operation. A rotation is the basic step for our analysis; the cost of one rotation is termed a *unit*; clearly, this is a constant.

*Received by the editors June 8, 1994; accepted for publication (in revised form) August 19, 1998; published electronically April 25, 2000. This work was supported in part by NSF grants CCR-8702271, CCR-8902221, CCR-8906949, CCR9202900, CCR-9503309, CCR-9800085, by ONR grant N00014-85-K-0046, and by a John Simon Guggenheim Memorial Foundation Fellowship. This work was made possible, in part, by the hospitality of the Laboratoire d'Informatique, Ecole Normale Supérieure.

http://www.siam.org/journals/sicomp/30-1/32699.html

†Computer Science Department, Courant Institute, New York University, 251 Mercer Street, New York, NY 10012-1185 (cole@cs.nyu.edu).

Sleator and Tarjan use the following *centroid potential* to analyze the amortized performance of a splay operation (our terminology). Node x is given weight, $wt(x)$, equal to the number of nodes in its subtree; they define the *centroid rank* of x, or simply the *rank* of x to be $\text{rank}(x) = \lfloor \log wt(x) \rfloor$ (our terminology). Each node is given a centroid potential equal, in units, to its centroid rank. Let δ denote $\text{rank}(g) - \text{rank}(x)$, if g is present; otherwise it denotes $\text{rank}(p) - \text{rank}(x)$. Sleator and Tarjan showed that the amortized cost of the splay step if g is present is at most 3δ units, while if g is not present the cost is at most $\delta + 1$ units (see [ST85, Lemma 1]). Since the total increase in rank for the complete access is bounded by $\log n$, the amortized cost of an access is at most $3 \log n + 1$ units. More generally, this analysis can be applied to weighted trees in exactly the same way. We call this the *centroid potential analysis*.

We now list the conjectures formulated by Sleator and Tarjan.

Dynamic Optimality Conjecture. Consider any sequence of successful accesses on an n-node binary search tree. Let A be any algorithm that carries out each access by traversing the path from the root to the node containing the accessed item, at a cost of one plus the depth of the node containing the item, and that between accesses performs an arbitrary number of rotations anywhere in the tree, at a cost of one per rotation. Then the total time to perform all the accesses by splaying is no more than $O(n)$ plus a constant times the time required by algorithm A.

Dynamic Finger Conjecture. The total time to perform m accesses on an arbitrary n-node splay tree is $O(m + n + \sum_{j=1}^{m} \log(d_j + 1))$, where, for $1 \le i \le m$, the jth and $(j-1)$th accesses are performed on items whose ranks differ by d_j (ranks among the items stored in the splay tree). For $j = 0$, the jth item is interpreted to be the item originally at the root of the splay tree.

Traversal Conjecture. Let T_1 and T_2 be any two n-node binary search trees containing exactly the same items. Suppose we access the items in T_1 one after another using splaying, accessing them in the order they appear in T_2 in preorder (the item in the root of T_2 first, followed by the items in the left subtree of T_2 in preorder, followed by items in the right subtree of T_2 in preorder). Then the total access time is $O(n)$.

Sleator and Tarjan state that the dynamic optimality conjecture implies the other two conjectures. (The proof is nontrivial.)

There have been several works on, or related to, the optimality of splay trees [STT86, W86, T85, Su89, Luc88a, Luc88b]. [STT86] showed that the rotation distance between any two binary search trees is at most $2n-6$ and that this bound is tight; they also related this to distinct triangulations of polygons; although connected to the splay tree conjectures, this result has no immediate application to them. [W86] provided two methods for obtaining lower bounds on the time for sequences of accesses to a binary search tree; while some specific tight bounds were obtained (such as accessing the bit reversal permutation takes time $\Theta(n \log n)$), no general results related to the above conjectures follow. [T85] proved the scanning theorem, a special case of the traversal conjecture (also a special case of the dynamic finger conjecture): accessing the items of an arbitrary splay tree, one by one, in symmetric order, takes time $O(n)$. Sundar [Su89] considered various classes of rotations on binary trees. Before discussing these we need some terminology. A right rotation is a rotation between a node u and its left child v. A k-right cascade comprises a sequence of k right rotations on the following path: the path is a descending sequence of $2k$ nodes, $v_1, v_2, \ldots, v_{2k}$, where v_{i+1} is the left child of v_i for $1 \le i < 2k$; the rotations are between nodes v_{2i-1} and v_{2i} for

$1 \leq i \leq k$. Sundar [Su89] shows the following bound on the maximum number of k-cascades in a sequence of right rotations on an n-node binary tree: $\Theta(n\alpha_{\lfloor k/2 \rfloor}(n))$, where α_k is the inverse of the kth level of the Ackerman hierarchy; in particular, for $k = 2\alpha(n)$, where α is the inverse Ackerman function, this is a $\Theta(n)$ bound. Our result will make use of this bound. Sundar uses this bound to obtain results concerning the deque conjecture; formulated by Tarjan [T85], it states that if a splay tree is used to implement a deque, in the natural way, then a sequence of m operations on a deque, initially of n items, takes time $O(m+n)$; Sundar proved a bound of $O((m+n)\alpha(m+n))$.

In this paper, we prove a generalized form of the dynamic finger conjecture, which includes insertions and deletions as well as searches among possible accesses (which is implicit in the above formulation of the conjecture). We note that our implementation of the delete operation is not the one described by Sleator and Tarjan [ST85]. The generalized form of the dynamic finger conjecture is as follows.

GENERALIZED DYNAMIC FINGER CONJECTURE. The total time to perform m searches, insertions, and deletions on an arbitrary splay tree, initially of size n, is $O(m+n+\sum_{j=1}^{m} \log(d_j+1))$, where, for $1 \leq i \leq m$, the jth and $(j-1)$th accesses are performed on items whose ranks differ by d_j (ranks among the items stored in the splay tree). For $j = 0$, the jth item is interpreted to be the item originally at the root of the splay tree.

Comment: The $+1$ in the log term is present to avoid problems with $\log 1$ and $\log 0$.

To measure the rank difference for an item deleted or inserted on the previous or the current operation, respectively, simply consider the immediate predecessor or successor of the item at the time of the deletion or insertion.

In the Part I paper a special case of splay sorting was examined. Splay sorting is closely related to the dynamic finger conjecture. It is defined as follows. Consider sorting a sequence of n items by inserting them, one by one, into an initially empty splay tree; following the insertions, an in-order traversal of the splay tree yields the sorted order. This is called *splay sorting*. A corollary of the dynamic finger conjecture is as follows.

SPLAY SORT CONJECTURE. Let S be sequence of n items. Suppose the ith item in S is distance I_i in sorted order from the $i-1$th item in S for $i > 1$. Then splay sort takes time $O(n + \sum_{i=2}^{n} \log(I_i + 1))$.

Incidentally, an interesting corollary of the splay sort conjecture is as follows.

SPLAY SORT INVERSION CONJECTURE. Let S be sequence of n items. Suppose the ith item in S has I_i inversions in S (counting inversions both to left and right). Then splay sort takes time $O(n + \sum_{i=1}^{n} \log(I_i + 1))$.

In the Part I paper we proved the splay sort conjecture for the following type of sequence. Suppose the sorted set of n items is partitioned into subsets of $\log n$ contiguous items, called *blocks*. Consider an arbitrary sequence in which the items in each block are contiguous and in sorted order. We call such a sequence a *$\log n$-block sequence*. We showed an $O(n)$ bound for splay sorting a $\log n$-block sequence.

The key contribution of the Part I paper was the notion of lazy potentials and their analysis. This notion can be viewed as a tool for designing potential functions. The lazy potential is a refinement of an initial potential function that avoids waste when potentials decrease. (For an example of waste, consider the following splay tree analyzed using the centroid potential. The tree is a path of n nodes, each of unit weight. The last node on the path is accessed. The resulting splay will have a real cost of $\Theta(n)$ but will reduce the potential by $\Theta(n \log n)$. The desired amortized cost

of this operation is $O(\log n)$, so essentially all the reduction in potential is wasted.) The idea of the lazy potential is to keep an old potential, ϕ_{old}, in situations in which the creation of a new (larger) potential, ϕ_{new}, may be followed by a return to the ϕ_{old} potential, with essentially $\phi_{new} - \phi_{old}$ potential being wasted. Not surprisingly, some care is needed in choosing which operations to treat as "lazy." In the Part I paper, collections of nodes all having lazy potentials formed *lazy trees*. The nodes in a lazy tree had originally been contiguous nodes on a left (or right) path. The lazy potentials were the "correct" potentials with respect to that original path. The key to the analysis was to provide additional potentials to the nodes of the lazy tree to pay for rotations that restored them towards their original left (or right) path configuration. As it happened, these potentials were linear in the size of the lazy tree and could be provided when the lazy tree was being created.

The lazy potential used here is similar to that used in the Part I paper. But, in contrast to the Part I paper, the additional potentials provided to the lazy trees may be superlinear in the size of the lazy tree. However, whenever we need to provide a superlinear potential, it is superlinear by only an inverse Ackerman function, and again we are able to provide this superlinear potential as the lazy trees are being created (the proof of the superlinear bound relies on the analysis of cascades provided by Sundar [Su89]). At this point it is convenient to define the version of the inverse Ackerman function, α, used in this paper. Define the following:

$$
\begin{aligned}
A_0(j) &= 2j \quad \text{for all } j \geq 1, \\
A_1(j) &= 2^j \quad \text{for all } j \geq 1, \\
A_i(j) &= \begin{cases} A_{i-1}(2) & \text{if } i \geq 2 \text{ and } j = 1, \\ A_{i-1}(A_i(j-1)) & \text{if } i \geq 2 \text{ and } j \geq 2, \end{cases} \\
\alpha(n) &= \min\{k \geq 1 | A_k(1) \geq n\} \quad \text{for all } n \geq 1.
\end{aligned}
$$

The other main feature of our analysis is a hierarchical partitioning of the items into blocks. A block comprises a contiguous set of items in the splay tree. The blocks at level 0 in the partition are called 0-blocks; at successively higher levels of the partition, the blocks are termed 1-blocks, 2-blocks, and so on. The highest level, vis, contains a single block which holds all the items in the tree. A separate potential function is associated with each block. The accesses are divided into sequences with respect to each level of blocks. A sequence, S_i, comprises a maximal series of successive accesses all to the same i-block. We call the first access of each sequence S_{vis-1}, a *global* access. Subsequent accesses of S_{vis-1} are called *local* accesses. We will prove an $O(\log n)$ bound on the amortized cost of global accesses and appropriately smaller bounds on the cost of local accesses.

In section 2, we analyze the global accesses. In section 3, we introduce and analyze the local accesses. We show a bound of $O(\log(d+1) + (\log\log n)\alpha(n))$ on the cost of a local access, where d is the distance, in items, between the current accessed item and the previously accessed item. In section 4, this bound is tightened to obtain the desired $O(\log(d+1))$ bound. The additional $O((\log\log n)\alpha(n))$ term arises because of the interaction between the different levels of blocks, as we will see later. In section 5, we extend the bound to allow insertions and deletions. Unfortunately, the constants in the bound are rather large (10^18). In part, this may be due to the fact that we have not sought to optimize the constants, but, to a larger extent, it is a result of the involved nature of the analysis. In section 6, we briefly discuss whether it might be possible to improve the constants in our bounds.

2. The block structure and global accesses. Before embarking on the analysis of global accesses, we provide some definitions which will be needed throughout the paper. Let S be a nonempty contiguous subset of the items in the splay tree. We define the tree, T_S, induced by S, as follows. Let u and v be, respectively, the leftmost and rightmost items in S. The root, r, of T_S, is the least common ancestor of u and v in the splay tree; note that r is a node of S. The left (resp., right) subtree of r is the tree induced by the set of items in S to the left (resp., right) of r, if nonempty; otherwise the subtree is empty. A node u is a *left descendant* of v if it is in v's left subtree or is v itself; we also say that v is a *right ancestor* of u. Right descendants and left ancestors are defined analogously. The right (resp., left) extreme path of a tree is the path from the root to the rightmost (resp., leftmost) item in the tree, excluding the root itself. The *access path* for an access comprises the traversed nodes other than the accessed node itself.

Next, we define blocks at level i, i-blocks, for $i = 1, 2, \ldots$. A 0-block comprises 1 item; a 1-block comprises c_1 items; an i-block comprises b_i $(i-1)$-blocks for $i > 1$; c_i is the number of items in an i-block for $i \geq 1$. For much of the paper, we choose $b_i = 2^{2^i}$, $c_i = b_{i+1}$ for $i \geq 1$. For each $i \geq 1$, we allow the rightmost i-block to be undersized. At the topmost level, level vis, a single block contains all the items in the tree. Henceforth, to simplify the discussion, we refer to an i-block as the block, an $(i+1)$-block as a superblock, an $(i-1)$-block as a subblock, and a j-block, $j < i-1$, as a miniblock. Sometimes, if confusion may arise, we refer to these blocks as i-superblocks, i-subblocks, and i-miniblocks, respectively.

We make a number of definitions with respect to the superblocks. The *superblock tree* for superblock B is the tree induced by the items in B. The *root* of the superblock tree for a superblock is called the *superblock root*. The block tree and block root for block B are defined analogously. Henceforth, when a node is identified as a block root, it is implicitly assumed not to be a superblock root. The block roots in the superblock and the superblock root itself are called *global nodes*; a global node other than the superblock root is called a *true global node*. All other nodes in the superblock are called *local nodes*. The *skeleton* of a superblock comprises the nodes on the extreme paths of its blocks. The *inner skeleton* of a superblock comprises the nodes on the skeleton other than nodes on an extreme path of the superblock. A node has *visibility* i if it belongs to the inner skeleton of an i-superblock. A node on an external path of the vis-block has visibility vis. On occasion, for precision, the skeleton (resp., inner skeleton) of an i-superblock is called the i-skeleton (resp., i-inner skeleton). Also, the global nodes in an i-superblock are said to be i-global.

Let P be a maximal contiguous path in the splay tree which is a portion of an extreme path for u's block, where u is the root of the block. If P is on the inner skeleton of u's superblock and if the top node on P is a child of u in the splay tree, then P is said to *abut* u. The *right* (resp., *left*) abutting path for u is the path to the right (resp., left) of u that abuts u, if any. Also, u is called the *parent* of P.

The nodes in each i-superblock are given centroid ranks, called the global ranks, or $g_$ranks for short, using the following weighting. Let $K_i = 2^{\lceil \log b_{i+1} \rceil}$ for $i > 0$, and $K_i = 2^{\lceil \log c_1 \rceil}$ for $i = 0$. The root is given weight $-K_i$, each true global node is given weight one, and each local node is given weight zero. The rank of v is given by $\lfloor \log wt(\text{subtree rooted at } v) \rfloor$, where we define $\log 0 = -1$. The two extreme nodes in the superblock are given additional weight K_i. We note that on an extreme path of a superblock tree, each node has the same global rank; also, this global rank is larger than that of any node on the inner skeleton. Each true global node is given a

potential, called its global potential, equal in units to gp times its global rank, gp a constant to be specified later.

Each local node on the skeleton is either *labeled* or *unlabeled*. Each labeled node has potential c, c a constant to be defined later. The labels and potentials are defined with respect to each level of block to which the node belongs. For each maximal path P of unlabeled nodes, the following invariant is maintained.

INVARIANT 1. *Let P be a maximal path of unlabeled local nodes on an extreme path of one block. Suppose that the nodes of P are contiguous in the splay tree. Then P has an associated* path potential *of $2c\lfloor\log(|P|+1)\rfloor$ units.*

Each of the terms just defined, if prefixed by i, will refer to a block at level i; alternatively we may use the prefixes "super," "sub," and "mini" to refer to the terms associated with superblocks, subblocks, and miniblocks. Implicitly, in the discussion that follows, all references are with respect to a fixed i-superblock.

A labeled node on the inner skeleton may carry a *small* or *large* debit. Small and large debits, are worth sd and ld units, respectively, sd and ld constants to be specified later. The following invariants apply to the labels and debits.

INVARIANT 2. *An unlabeled local node is always adjacent, in the splay tree, to two other local nodes which are on the skeleton and in the same block.*

INVARIANT 3. *Only a labeled local node on the inner skeleton can have a debit.*

INVARIANT 4. *Let v be a local node and let v belong to block B. Suppose that in the splay tree, v is the left (resp., right) child of its parent u. v can have a large debit only if*

(i) *u is a local node in block B;*

(ii) *v has a left (resp., right) child w in the splay tree which is a local node and in block B;*

(iii) *neither u nor w carry any debit.*

(Note that both u and w are on an extreme path of block B and hence on the inner skeleton.)

INVARIANT 5. *Let u be a true global node. Let P be a path abutting u, if any. Let v be the top node on P (v is a child of u). v can have a small debit only if $g\_\text{rank}(v) < g\_\text{rank}(u)$.*

INVARIANT 6. *Let u be the root of block B. Let P be a path abutting u, if any. Let w be the bottom node on P. w can have a small debit only if $g\_\text{rank}(w) < g\_\text{rank}(u)$.*

To avoid special cases it is convenient to redefine the access path for an access to exclude the splay tree root r in the event that r is involved in an incomplete rotation (Case 1 of the splay step). Now consider a rotation performed during the splay along the access path. Of the three nodes involved in the rotation, the top two are called the *coupled* nodes of the rotation, or a *couple* for short. The analysis focuses on the coupled nodes in a rotation.

For the purposes of the analysis the access path is partitioned into segments. Each segment comprises an even number of nodes, every two nodes on the path forming the coupled nodes of a rotation of the present splay operation. The segments are created by a traversal of the access path from bottom to top; each segment is chosen to have the maximum length such that, following the removal of its front (top) two nodes, the (truncated) segment satisfies the following conditions:

(i) The nodes on the truncated segment all belong to the inner skeleton of one superblock; in addition, the accessed item is part of the inner skeleton of this superblock immediately prior to the traversal of the current segment.

FIG. 1. *Type* 1 *nodes.*

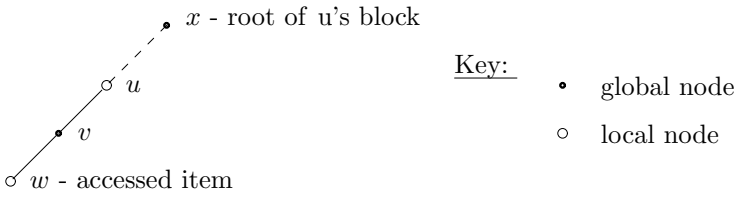


FIG. 2. *Type* 2 *nodes.*

(ii) The node being accessed has the same global rank throughout the rotations involving the truncated segment.

(iii) For each couple, the rotation does not change the total global potentials.

(iv) (This is implied by (ii) and (iii).) Each couple in the truncated segment includes at least one local node (i.e., it does not comprise two global nodes).

(v) Define a node to be *staying* if it is involved in a zig-zag rotation or it is the lower node in a couple. (Intuitively, the staying nodes are those that remain on one of the traversed paths following the splay. Note that the splay, in general, creates two traversed paths.) For each block there are at most two labeled staying local nodes in the truncated segment, following its traversal.

The topmost segment is said to be *incomplete* if it satisfies conditions (i)–(v) prior to truncation. We consider an incomplete segment to comprise a (trivially) truncated segment.

Next, we mark the following types of coupled nodes in each truncated segment. The rotations involving marked couples are self-paying, as is demonstrated later. Each marked couple is involved in a zig-zig rotation. For each type below, suppose the segment includes a couple, u, v, with u the parent of v.

Type 1. See Figure 1. Suppose that u is the root of v's block; then both u and v are marked.

Type 2. See Figure 2. Suppose that u is a local node, and let x be the root of u's block. Further, suppose that u is on the left (resp., right) abutting path for x. Let v be the left (resp., right) child of u; if u does not have a debit and if v is global, then both u and v are marked.

Type 3. Suppose u and v are both local nodes of the same block; then both u and v are marked.

We can now prove a bound on the length of a truncated segment.

LEMMA 1 (see [CMSS00, Lemma 1]). *A truncated segment comprises at most* 10 *unmarked nodes, of which at most* 7 *are local.*

Over the course of the access we maintain the following invariant.

INVARIANT 7. *See Figure* 3. *Let w be the accessed item. Let x be w's right (resp., left) child. Suppose that x is on the inner skeleton of its superblock. Let P*

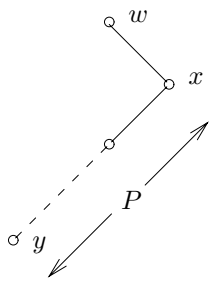


FIG. 3. *Invariant* 7.

be the maximal portion of the extreme left (resp., right) path in x's block descending from x, contiguous in the splay tree. Let y be the bottom node on P (x is the top node). Then neither x nor y carry any debits. w does not carry a debit either.

To initialize Invariant 7 at the start of the access may require the removal of up to five small debits or up to one large debit and three small debits; this costs at most

$$\max\{ld + 3sd, 5sd\}. \tag{1}$$

As we will see, the access will remove the debits from all the traversed nodes (though it may also cause large debits to be placed on some of the traversed nodes).

To understand the cost of individual rotations it is helpful to consider when abutting paths affect Invariants 5 and 6. We define an *untouched* abutting path to be one whose top node is not traversed in the current access and whose top node was not a child of the accessed item at the start of the access; any other abutting path is said to be *touched*. We define an abutting path to be *clean* if it obeys Invariants 5 and 6 with respect to its parent, and to be *dirty* otherwise. Because of Invariant 7, a node that receives a touched abutting path receives a clean abutting path.

LEMMA 2. *Let u be a true global node. Let P be an abutting path. If P is dirty, u must be on the inner skeleton of its superblock.*

Proof. By definition, P is on the inner skeleton. By construction, every node on an extreme path of a superblock has larger global rank than any node on the inner skeleton. So for P to be dirty, u must be on the inner skeleton of its superblock. □

LEMMA 3. *Let u be a node on the access path. Suppose that before the rotation involving u, u was j-global, and that after the rotation it is k-global, $k > j$. Then, after the rotation, u's abutting paths are clean.*

Proof. We note this must be a zig-zig rotation, and u must be the lower node in the couple. So of the at most two abutting paths u acquires, one is touched; the untouched one, if present, had been an abutting path for its parent, t (t and u are in the same couple). The only node added to this abutting path is t, which by assumption carries no debit. u acquires t's k-global rank, and so the untouched abutting path remains clean. □

LEMMA 4. *Let u be a node on the access path. Suppose that before the rotation involving u, u was j-global, and that after the rotation it is k-global, $k < j$. Then, after the rotation, u's abutting paths are clean.*

Proof. u remains on the j-skeleton and hence is not on the inner k-skeleton. The result follows from Lemma 2. □

LEMMA 5. *Let u be a node on the access path. Suppose that both before and after the rotation u is a true j-global node. Further suppose that u is on the inner i-skeleton before the rotation, where $i > j$. Then either*

(i) *u is on the inner j-skeleton after the rotation; then u's j-rank decreases;*

(ii) *u is not on the inner j-skeleton after the rotation; then u has no dirty abutting paths.*

Proof. The conclusion of (i) follows from the definition of the j-ranks. The conclusion of (ii) follows from Lemma 2. □

COROLLARY 1. *Let u be a node on the access path. Following the rotation, u can have a dirty abutting path only if*

(i) *u was a true j-global node both before and after the rotation; also, u is on the inner skeleton of its j-superblock after the rotation, and*

(ii) (a) *either u's j-global rank decreases, or* (b) *u receives a new untouched abutting path.*

Proof. Condition (i) summarizes Lemmas 2–4. There are only two ways in which u can acquire a dirty abutting path. First, an already abutting path becomes dirty; for this to happen, u's j-rank must decrease. Second, u receives a new abutting path; in order for the new abutting path to be dirty, it must be untouched (as already pointed out in the discussion following Invariant 7). □

Next, we investigate possibility (ii(b)) of Corollary 1 further.

LEMMA 6. *Consider the rotation involving couple u and v, where v is a child of u. Following the rotation, v does not have a new dirty untouched abutting path. u can receive a new dirty untouched abutting path only if both*

(i) *the rotation is a zig-zig rotation, and*

(ii) *v is in the same j-superblock as u but in a distinct j-block, where u is a true j-global node.*

Proof. In a zig-zag rotation the new abutting paths are all touched.

If v acquires a new dirty abutting path, v is a true j-global node both before and after the rotation. Also, for u to be in v's new abutting path, u must be in the same j-block as v; but then v was not a true j-global node before the rotation. So v does not have a new dirty untouched abutting path.

If u acquires a new dirty abutting path, u is a true j-global node both before and after the rotation. Again, if v is in the same j-block as u, u is not a true j-global node after the rotation. So v is in a distinct j-block. Suppose that u is on the inner i-skeleton before the rotation, where $i \geq j$. If $i > j$, by Lemma 5, u is on the inner j-skeleton after the rotation, and so v must be in the same j-superblock as u, while if $i = j$, as u was on the inner i-skeleton before the rotation, v must be in the same i-superblock as u. □

LEMMA 7. *Let u be a node on the access path. If u acquires a dirty abutting path, then u is in the leading couple of its segment. In addition, the visibility of w, the accessed item, is unchanged following the rotation involving u.*

Proof. If condition (ii(a)) of Corollary 1 applies, then clearly u is in the leading couple. So suppose that condition (ii(b)) applies. Consider Lemma 6; let v be the other node in u's couple (v is u's child). v is in the same j-superblock as u but in a distinct j-block. Also, the rotation involving u and v is zig-zig. There are two possibilities. First, v is j-global; then the rotation must reduce u's j-global rank or increase w's j-global rank; in either event u is in the leading couple. Second, v is not j-global; in order for v to be in a distinct j-block from u, w cannot be on the inner skeleton of B, u's j-superblock. So this rotation raises w's visibility, violating

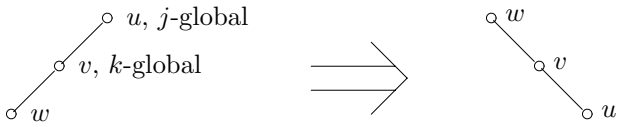


FIG. 4. *Type* 3 *nodes.*

condition (i) of the definition of a truncated segment; again, u must be in the leading couple. Finally, suppose w's visibility increases due to the rotation involving u. Then, u becomes a true k-global node, $k < j$, after the rotation; by Lemma 5, u does not acquire any dirty abutting paths. □

LEMMA 8. *Following the access, the accessed item's abutting paths are clean.*

Proof. This is an immediate consequence of Lemma 2, for the accessed item is not on the inner skeleton of any of its blocks. □

In the analysis of global accesses, whenever a rotation is performed (and paid for) another s spare rotations are also paid for, s a constant to be specified later. The spare rotations are needed subsequently to handle the effects of local accesses.

In order to pay for the rotations we provide the following potential. In each rotation, for each unit increase in rank on the part of the accessed item, w, we provide $3gp$ units of potential. It is used as follows. Consider a rotation involving couple u, v, where u is the parent of v. Suppose that w is currently the root of its block. Suppose that this rotation increases the global rank of w by I. Each of w and v (if global and in the same superblock as w) may increase their global potentials by up to $gp \cdot I$ units; this is a total of at most $2gp \cdot I$ units of potential. If $I > 0$, the remaining at least gp units of potential at hand will pay for the segment that ends at couple u, v (as we will see, gp units always suffice). When w becomes the root of its superblock, it interchanges potentials with the old root of the superblock (it may be that in one double rotation w in turn swaps potentials with v and then with u).

Further potential will also be provided; a total of $gp + vis \cdot vp + (ld + 3sd)$ units, vp being a constant to be specified (the role of the latter potential will become clear towards the end of this section).

Next, we show how to pay for the rotations (and associated spares) along a segment. First, each marked couple pays for its rotation (and spares), as follows. Recall that the marked couples are all involved in zig-zig rotations.

CASE 1. (Type 3.) The coupled nodes are both local nodes in the same i-superblock (see Figure 4). Nodes u and v are on the inner skeleton in the same superblock; without loss of generality, v is the left child of u, and w, the node being accessed, is the left child of v. Node u is j-global, and node v is k-global, $j, k < i$. There are several subcases depending on the relative values of j and k.

Case 1.1. $j > k$. After the rotation, v becomes j-global, and u becomes k-global. u and v exchange potentials. Since v is adjacent to a j-global node, it must be h-labeled, for $k < h \leq j$; it transfers the h-labels and associated potentials to node u. u is given a small j-debit; as it is now a j-local node on an inner j-skeleton, this obeys Invariant 3. By Lemma 2 and Corollary 1, Invariants 5 and 6 continue to hold. The small debit pays for the rotation and s spares. Thus it suffices that

$$sd \geq s + 1. \tag{2}$$

Next, we remove the debits present at u and v, if any. If both nodes are i-labeled,

then u spends its associated potential c; this pays for the removal of debits from both nodes. If only one node is i-labeled, as we show later, the traversal of a portion, or all, of the i-unlabeled path associated with the i-unlabeled node results in the i-labeling of that i-unlabeled node. So we proceed in the same way in this case. If neither node is i-labeled, then they do not carry debits. So it suffices that

$$c \geq \max\{2sd, ld\}. \tag{3}$$

Case 1.2. $j < k$. Then u and v are in the same k-superblock but are in distinct k-blocks. Here, v remains k-global and u j-global. u is given a small k-debit; again, as u is now a k-local node on an inner k-skeleton, Invariant 3 is satisfied. Again, by Lemma 2 and Corollary 1, Invariants 5 and 6 continue to hold. Here too, (2) suffices. The removal of debits is handled as in the previous case.

Case 1.3. $j = k$. The j-rank of u decreases, and this pays for the removal of small or large debits from u and v and of up to two small debits from nodes which now violate Invariants 5 and 6. So it suffices that

$$gp \geq \max\{2sd, ld\} + 2sd. \tag{4}$$

Case 2. (Type 1.) u is i-global and v is j-global, $j < i$ (see Figure 1). This is similar to Case 1.1, above. However, here v cannot have a debit before the rotation, by Invariant 5, since $g\_\text{rank}(u) = g\_\text{rank}(v)$ (this follows from the fact that the global rank of the accessed item does not change). Here u is given a small i-debit, for it remains on the inner i-skeleton. Note that following the rotation, $g\_\text{rank}(u) < g\_\text{rank}(v)$. Invariants 3, 5, and 6 continue to hold.

Case 3. (Type 2.) v is i-global and u is j-global, $j < i$ (see Figure 2). This is similar to Case 1.2, above. But here, u does not have a small debit by assumption (since the couple was marked). It is given a small debit following the rotation. As before, Invariants 3, 5, and 6 continue to hold.

The remaining rotations are paid for by the first couple in the segment, which was removed in truncating the segment and which causes a violation of at least one of the conditions (i)–(v), except in the case of an incomplete topmost segment, which is handled subsequently. The rotation involving the first couple falls into at least one of the following cases.

Case A. The rotation reduces the total global potential of the nodes in the first couple, or it increases the global rank of the accessed node.

Case B. The rotations along the segment create a sequence of three contiguous local nodes from the same block.

Case C. The rotation increases the visibility of the accessed item.

Each case involves three costs.

*Cost*1. The rotation and spares for each unmarked couple, including the leading couple: $\leq 6(s+1)$.

*Cost*2. Removal of small debits from the unmarked couples; *Cost*2 is analyzed below.

*Cost*3. Removal of small debits for local nodes that now violate Invariant 5 or 6; *Cost*3 is analyzed below.

We remind the reader that we remove all debits from the traversed nodes (though we may add new debits).

Lemma 9. (i) *For each segment in which the visibility of the accessed item is unchanged,* $Cost2 + Cost3 \leq 11sd$.

(ii) *For all other segments,* $cost2 + Cost3 \leq 7sd + \max\{2sd, ld\}$.

Proof. Lemma 9(i) is proven in [CMSS00], where it is Lemma 2. To prove (ii), we note that by Lemma 1, the truncated segment contributes at most $7sd$ to $Cost2$. By Lemma 7 it contributes nothing to $Cost3$. The removal of debits from the leading couple costs at most $\max\{2sd, ld\}$. Again, by Lemma 7 there is no contribution to $Cost3$. □

LEMMA 10. *Invariant* 7 *remains true throughout the access.*

Proof. This follows immediately from the fact that the traversed nodes have no debits following their traversal. Thus w's children do not carry debits (the node x of Invariant 7). Consider node y; this node changes only when the new node x has a larger visibility than the old node x; but then the new node y is the new node x. □

LEMMA 11. *A global access costs at most* $(3\sum_{i=1}^{vis} k_i + 1)gp + vp \cdot vis + 2c \cdot vis + (ld+3sd)$ *units, where* $sd = s+1$, $ld, c = 17(s+1)$, $gp = 19(s+1)$, $vp = 30(s+1)$, *vis is the number of levels of blocks, and* k_i *is the* i*-rank of nodes on an* $(i+1)$*-extreme path. (Note* $\sum_{i=1}^{vis}(k_i+1) \le \log n + 3vis + 1$*.)*

Proof. In case A, the cost of the rotation is paid for either by the drop in global potential, which provides at least gp units, or, if there is an increase in global rank for the accessed item, by gp units of the at least $3gp$ units provided for this rotation. In case B, there are at least three labeled staying local nodes; the middle node among these three labeled nodes is given a large debit, which pays for the rotation (note that the three labeled nodes need not be contiguous but they all lie on a contiguous path on the skeleton; the middle node obeys Invariant 4). In Case C, the cost of the rotation is charged to the new visibility level. (vp units are provided per level for a global access to pay for each instance of Case C.)

Thus it suffices to have

$$gp, ld \ge 6(s+1) + 11sd, \tag{5}$$

$$vp \ge 6(s+1) + 7sd + \max\{2sd, ld\}. \tag{6}$$

Recall that an i-superblock comprises b_{i+1} i-blocks; let k_{i+1} denote the i-rank for nodes on an $(i+1)$-extreme path. Clearly, $\sum_{i=1}^{vis} k_i \le \log n + 2vis + 1$; with $b_i = 2^{2^i}$, we get $\sum_{i=1}^{vis} k_i \le \log n + vis + 1 \le 2\log n$. We use $3(k_{i+1}+1)gp + vp$ units of potential to pay for rotations in which the accessed item traverses the inner skeleton of its i-superblock and for which, on completion, the accessed node has reached an extreme path of its i-superblock. The $3(k_{i+1}+1)gp$ units pay for rotations in which the i-global rank of the accessed item increases, as explained earlier. The vp units pay for the rotation in which the accessed item reaches an extreme path of its i-superblock. Overall, we need

$$\sum_{i=1}^{vis}(3(k_i+1)gp + vp) + \max\{ld + 3sd, 5sd\} \tag{7}$$

units to pay for global accesses so far (see (1)).

Now, we show how to pay for the incomplete segment, if present. We provide an additional gp units to pay for this segment; in addition, this term is used to pay for the incomplete rotation, if any; however, the additional gp term does not need to account for any increase in global rank, on the part of the accessed item, during the incomplete rotation, for this has already been accounted for. The result of Lemma 9 applies here too (in fact, a tighter bound can be shown). Here too, (5) suffices.

We have yet to explain how to maintain the potential of an unlabeled path and the invariants concerning labels, Invariants 1 and 2. If a maximal unlabeled path is traversed in its entirety and, in addition, the child on the extreme path of this unlabeled path is also traversed, then the two old endpoints of the path are labeled (or rather, if an old endpoint is in a couple with another unlabeled node, then the new endpoint is given the label) and the remainder of the path is essentially halved in length; the sufficiency of the potential is readily verified (for a path of length at most 4, the potential suffices to label each node on the path; for a path of length $2k+2$, $k>2$, we have $2\lfloor\log(2k+3)\rfloor \geq 2+2\lfloor\log(k+1)\rfloor$, and for a path of length $2k+3$, $k\geq 2$, we have $2\lfloor\log(2k+4)\rfloor \geq 2+2\lfloor\log(k+2)\rfloor$).

If only a portion of an unlabeled path is traversed, then the path was accessed through an increase in the visibility of the accessed item. Consider an unlabeled path that is part of a left extreme path of its block (the case of a right extreme path is analogous). At most two unlabeled nodes need to be labeled to maintain Invariants 1 and 2, namely the new top of the unlabeled path and the old root of the block. This costs at most $2c$ units. This charge is levied once for each unit increase in visibility on the part of the accessed item. Over the whole access this is a cost of $2c\cdot vis$ units.

On taking equalities in (2)–(6), the lemma follows from (7). □

Unfortunately, the potential, as described, may be of size $\Theta(n\log\log n)$ initially. (For example, consider a zig-zag path descending from the root.) To reduce this to linear size, we modify the rules for creating unlabeled paths and modify the analysis accordingly.

For each block, each of its extreme paths is partitioned into chains. A chain for block B, at level i, comprises a maximal sequence of nodes, $v_1, v_2, \ldots, v_k$, from B, such that the path P in the splay tree from v_1 to v_k does not include any j-global node, $j\geq i$. Each chain, C, is given a path potential of size $2c\lfloor\log(|C|+1)\rfloor+c$ units and a reserve path potential of $2\cdot 2c\lfloor\log|C|\rfloor$. The role of the reserve path potential will become clear in section 4. In addition, each node adjacent on P to a node that is not part of the chain is given an additional *zig-zag* potential of c units; this potential is provided once, and not for each block to which the node belongs.

LEMMA 12. *The cost of the path and reserve path potentials is at most*

$$3\sum_{j=1}^{vis}\left\lceil\frac{n}{c_j}\right\rceil 4c[\lfloor\log c_i\rfloor+3\sum_{i=1}^{j}4c\lfloor\log c_i\rfloor]+2c\cdot(j+1).$$

This is bounded by $144c\cdot n$ *if a* j*-block has size* $c_j=2^{2^{j+1}}$ *for* $j\geq 1$ *(with the possibility of one smaller block at each level* j*).*

Proof. The path potential for each chain, apart from the bottommost chain on each extreme path, is charged to the j-global node terminating the chain at the bottom. The remaining at most two chains in each block are charged to the block itself. Each j-global node is charged for at most two chains on the extreme paths of i-blocks, for each $i\leq j$. A chain of an i-block has length at most c_i-1, so the path potential for an i-chain is at most $2c\lfloor\log c_i\rfloor+c$. Hence the charge to a j-global node is bounded by $\sum_{i=1}^{j}12c\lfloor\log c_i\rfloor+2c$. Likewise, the charge to a j-block is bounded by $12c\lfloor\log c_j\rfloor+2c$. Summing over all j-global nodes, j-blocks, and all j gives a total charge of $\sum_{j=1}^{vis}\lceil\frac{n}{c_j}\rceil 12c[\lfloor\log c_j\rfloor+\sum_{i=1}^{j}\lfloor\log c_i\rfloor]+2c\cdot(j+1)$.

For $c_i=2^{2^{i+1}}$ this is bounded by $\sum_{i=1}^{vis}\lceil\frac{n}{c_j}\rceil 72c\cdot 2^j+2c\cdot(j+1)$ which is bounded by $72c\cdot(n+2\log n)+c\cdot(vis+1)(vis+2)\leq 144c\cdot n$, assuming $n\geq 16$ ($vis=\log\log n$).

For $n < 16$, $vis = 1$, so it suffices to give potential c to each node on an extreme path of the splay tree for a total of cn potential. □

LEMMA 13. *A global access costs at most* $(3\sum_{i=1}^{vis}(k_i+1)+1)gp+vp\cdot vis+2c\cdot vis+(ld+3sd)$ *units, where* $sd = s+1$, $ld, c = 17(s+1)$, $gp = 19(s+1)$, $vp = 30(s+1)$, vis *is the number of levels of blocks, and* k_i *is the* i*-rank of nodes on an* $(i+1)$*-extreme path. (Note* $\sum_{i=1}^{vis}(k_i+1) \leq \log n + 3vis + 1$.*) In particular, maintaining the chain potentials requires a charge of at most* $2c$ *for each unit increase in visibility on the part of the accessed item.*

Proof. It suffices to explain how to modify the analysis of Lemma 11 to account for chains. There are two ways a chain can be traversed: in its entirety or in part. We consider each in turn.

If a chain is traversed in its entirety, following the traversal it forms a path for which the associated path potential is sufficient to allow the previous analysis to be performed; indeed this path potential is too large by c units. These c units are used to label the old root of the block if the root of the block is changed by the traversal. All the couples which include just one node of the chain are paid for by the associated zig-zag potential of c units, or to put it in terms of marking, these couples are marked.

Next, consider a chain which is only partially traversed. Suppose, without loss of generality, that this chain is on the right extreme path of its block. Let v_j be the bottommost node on the chain to be traversed. If v_j is accessed from its left child, then the rotation involving v_j results in an increase in the visibility of the accessed item. This increase in visibility pays for the rotation involving node v_j. Again, the old root of the block may need to be labeled, which costs c units; this is also charged to the increase in visibility. The path potential is associated with the remainder of the chain on and below node v_j. The rotations along the traversed portion of the chain are paid for as in the traversal of the full chain. In addition, the nodes on the traversed portion cease to be on an extreme path of their block and so do not need an associated path potential on completion of the traversal. There may be several chains including v_j which are partially traversed, but the number of these chains is bounded by the increase in visibility. More precisely, let v_j be h-global, and suppose it is on an inner i-skeleton. Then every chain including v_j but not ending at v_j must be on an extreme path of an l-block, $h < l \leq i$; thus there are at most $i - h$ such chains. The increase in visibility for the accessed item, in its rotation with v_j is at least $i - h$. Hence each unit increase in visibility is charged c for the labeling of the root of one partially traversed chain of the type described in this paragraph.

Finally, suppose v_j is accessed from its right child. Then there is a node w on the path between v_j and v_{j+1} which is accessed from its right child. The rotation involving w results in an increase in the visibility of the accessed node. This increase in visibility is charged for the additional potential of c units needed to label the old root of v_j's block. Again, the rotations along the traversed portion of the chain are paid for as in the traversal of the full chain. The path potential continues to be associated with the remainder of the chain starting at node v_{j+1}. We note that there may be many chains including nodes v_j and v_{j+1}, each on an extreme path for a different level of block. But the level of node w must be smaller than that of these blocks. To be specific, let w be h-global. Then every chain including v_j and v_{j+1} must lie on an l-block extreme path, with $h < l$. Suppose further that v_j and v_{j+1} are on an inner i-skeleton. Then the increase in visibility of the accessed item in rotation with node w is at least $i - h$, and there are at most $i - h$ chains straddling w, each of which may require a charge of c to label its old root.

Thus an increase in visibility which places the accessed item on the inner skeleton of an i-superblock results in a charge of up to $2c$ to visibility level i. □

Remark 1. Later, the form of the blocks will be generalized to allow more complex situations involving local nodes. Specifically, we will allow local nodes to carry other debits. For the above analysis to continue to apply, it will suffice that

(i) if a local node on its block's right (resp., left) path carries a new debit, then its parent and right child (resp., left child) are both local nodes of the block,

(ii) a couple containing two local nodes must pay for the removal of all debits on the couple's nodes; the $s+1$ rotations will be paid for as before.

3. Local insertions. For each level i of blocks, the access sequence S is partitioned into maximal subsequences S_l; each subsequence comprises accesses in the same i-block. Recall that an $(i+1)$-block contains b_{i+1} i-blocks and c_{i+1} items. We allocate amortized time $O(\log b_{i+1} + |S_l|\alpha(c_{i+1}))$ to pay for the cost of the sequence S_l. Specifically, the first access to an i-block has amortized cost $O(\log b_{i+1})$ and subsequent accesses have amortized cost $O(1+\alpha(c_{i+1}))$. Note that an access occurs in vis sequences; thus the cost of an access comprises the sum of vis terms. So suppose that an access is to a new i-block, but to the same $(i+1)$-block, with respect to the previous access. Then the cost of the access is

$$O\left(\sum_{h=0}^{i} \log b_{h+1} + \sum_{h=i+1}^{vis} [1+\alpha(c_{h+1})]\right) = O(\log b_{i+1} + (\log\log n)\alpha(n)) \tag{8}$$

(assuming $b_i = 2^{2^i}$ which implies $vis = \lceil \log\log n \rceil$). In section 4, we show how to reduce the $O((\log\log n)\alpha(n))$ term to $O(1)$.

It is possible to have an access close to the previous access be in a different high level block (if both accesses are close to the boundary between the blocks). In section 3.7, we will see how to reduce this excess cost and obtain an amortized time for the access proportional to the logarithm of the distance between successive accesses (plus an additional $O(\log\log n\alpha(n))$ term). For now, we focus on the cost of the access at a single level, i. To be specific, we focus on the rotations during which the accessed item is on the inner $(i+1)$-skeleton.

In the present analysis spare rotations will be spent in several ways. We spend the spare rotations in portions of size s', s' a constant to be specified. Recall that s spares are provided per couple. As we will see in section 4, it suffices to have $s \geq 3s'$.

3.1. Forming lazy trees. As in the analysis of splay sorting $\log n$-block sequences, we construct lazy trees. However, their form is more involved here.

For each level of blocks a separate analysis is performed, as in the analysis of global accesses. Here we focus on the traversal of the inner skeleton of superblock B, an $(i+1)$-block. For each item, the analysis of its traversal of the inner skeleton of superblock B is performed as part of the analysis of a maximal sequence S of accesses to the same block B' of superblock B.

We start with a few definitions. Let b be the number of blocks in superblock B and let c' be the number of items in B. Following the first access of S, we define the *right access path* as follows: Let *right_B* denote the set of elements in superblock B to the right of block B'. Consider the tree induced by the nodes of *right_B*; its left extreme path is the *right access path* (see Figure 5). The *left access path* is defined analogously. In the splay tree, an access path may comprise more than one contiguous segment of nodes. Note that as the accesses of sequence S proceed, the access paths become shorter. Lazy trees are formed from the nodes originally on the access paths.

FIG. 5. *The right access path.*

Let $l\_child(v)$ denote the left child of v. For each global node, v, on the right access path, other than the bottom global node, define $jump(v) = g\_\text{rank}(v) - g\_\text{rank}(l\_child(v))$; for the bottom global node, define $jump(v) = g\_\text{rank}(v) + 1$. $jump(v)$ is defined analogously for global nodes on the left access path.

Consider an access in sequence S; there are three possibilities. First, no node on either access path is traversed, in which case no cost is incurred for the traversal of the inner skeleton of superblock B. Second, the right access path (or rather a top portion of it) is traversed. Third, a top portion of the left access path is traversed.

Our goal is to accumulate spares on the nodes of the access paths, so that following e' accesses we can assume that each node on the access paths, apart from the top node, has accumulated at least $2^{e'-1} - 1$ spares, e' a constant to be specified. We need to be careful, however, for higher portions of the access paths may have been traversed more frequently and thus have accumulated more spares. We proceed as follows.

It is convenient to ensure that on each traversal, for each couple, its two nodes have been traversed the same number of times. This need not be true. So we introduce *pseudotraversals*. Initially, each node on an access path has been pseudotraversed zero times. Each time a node is traversed, its pseudotraversal count is incremented, up to a maximum of e' pseudotraversals. In addition, immediately prior to each access of sequence S, apart from the first access, for each couple in the upcoming access, for each lower node in the couple, its pseudotraversal count is increased to the count of the higher node. We note that the pseudotraversal counts are nondecreasing from bottom to top of the access paths (for each time a node is traversed so are all its ancestors on its access path).

Immediately following the first access in S, we provide each global node, v, on an access path with *jump potential* $2gp \cdot e' jump(v)$. Summing over the two access paths, we obtain a cost of $4 \cdot e' gp(\log 2b + 1)$ units charged to the first access of sequence S; the first access itself costs a further $3gp(\log 2b + 1)$ units. As further traversals of the access paths occur, Invariants 8 and 9 below are maintained.

INVARIANT 8. *Each global node, v, on an access path, apart from the first node, carries a jump potential of $2gp \cdot (e' - t)jump(v)$ after having been pseudotraversed $t \le e'$ times.*

INVARIANT 9. *Each node on an access path, global or not, apart from the first node, carries a spare potential of $s' \cdot (2^{(t-1)} - 1)$ units after having been pseudotraversed $t \le e'$ times.*

Clearly the invariants are true following the first access of sequence S.

First, we explain how to maintain the invariants following the increase in pseudotraversal counts (immediately prior to an access). Clearly, Invariant 8 is unaffected

for the jump potentials can only decrease. To maintain Invariant 9, an additional spare potential of $s' \cdot (2^{q-1} - 2^{p-1})$ is provided to a node whose pseudotraversal count increases from p to q. This is a total of at most $s' \cdot 2^{e'-1}$ spare potential per access for the pseudotraversal counts are nondecreasing along the access path (from bottom to top).

Next, we show how to pay for an access and how to restore Invariants 8 and 9 following the access. Consider one couple of this access, where the couple contains nodes u and v, and u is the parent of v. Further suppose that at least one of u and v is on the access path.

The rotation is paid for as in a global access, except that the jump potentials provide the additional potential needed to pay for the segment associated with a rotation which increases the global rank of the accessed item. There is one special case: if the rotation increases the visibility of the accessed item, then the increase in visibility is charged for the segment associated with the rotation, and the increase in the global rank of the accessed item is not charged. In addition, we are no longer seeking to provide a global potential to the accessed item, w. For w eventually acquires the global potential of the old root, x, of w's block, and x then acquires the sub or mini potential of w. More precisely, it will suffice, at each global node, y, on the access path, to provide $2gp \cdot jump(y)$ units of potential, taken from y's jump potential.

It remains to explain how Invariants 8 and 9 are restored following the rotation. We start with Invariant 9. u gives v all its spare potential; in addition, v receives s' of the spares allocated to the couple u, v. This gives v spare potential $s'[2 \cdot (2^{(t-1)} - 1) + 1] = s'(2^t - 1)$, as needed, since t will be incremented.

Invariant 8 needs restoring only if both u and v are global nodes; then u gives v its remaining jump potential. To show this suffices we argue as follows. Without loss of generality, suppose that both u and v are on the right access path. First, we note that $g\_\text{rank}(l\_child(v))$ is unchanged following the rotation involving v (for w, the accessed item, is in block B' and so its left subtree contains no global items, while w's right subtree remains a subtree of v). Likewise, $new\_g\_\text{rank}(v)$, $g\_\text{rank}(v)$ following the rotation involving v, is exactly $g\_\text{rank}(u)$ before this rotation. So $new\_jump(v)$, the value of $jump(v)$ following the rotation, is given by $g\_\text{rank}(u) - g\_\text{rank}(l\_child(v))$, which is $[g\_\text{rank}(u) - g\_\text{rank}(v)] + [g\_\text{rank}(v) - g\_\text{rank}(l\_child(v))]$, or $jump(u) + jump(v)$, as claimed. Hence, following the rotation, v has jump potential $2gp \cdot (e' - (t+1))jump(v)$ (using the new value for $jump(v)$); as t is incremented following the rotation, Invariant 8 continues to hold.

Once a node on an access path, other than the top node, has been pseudotraversed e' times, and hence carries spare potential $s'(2^{e'-1}-1)$, when it is rotated off the access path it becomes a lazy node and is given q_1 units of potential, q_1 a constant to be specified. In addition, if both the nodes u and v in the couple are global, they swap their global potentials; the potential now carried by u, the node leaving the access path, is called its *lazy potential*. Each couple pays for its rotation, spare rotations, and associated segment charge, if any, by a charge of gp units to the spare potential of the node leaving the access path, instead of by the creation of debits. So it suffices to have

$$s'(2^{e'-1} - 1) \geq gp + q_1. \tag{9}$$

When a rotation removes the top node of an access path it too becomes a lazy node; however, it is given $a_3\alpha(c') + b_3$ potential, where a_3 and b_3 are constants to be specified. So following the first access of sequence S, the charge per access for the traversal of nodes on the inner skeleton of block B is bounded by $s' \cdot 2^{e'-1} + a_3\alpha(c') + b_3$.

We note that a lazy node, on creation, carries no debit. For the rotation creating the lazy node is paid for by the gp charge to the node's spare potential (see (9)).

In fact, we provide another $2gp \cdot (\log 2b + 1)$ units to the first access of sequence S. This is used as follows. For each global node u remaining on an access path, as soon as u acquires a proper global descendant v on the access path, which has been pseudotraversed e' times, we provide u a *reserve potential*, defined as follows. Define the *marker_rank* for a global node, v, on the access path to be the global rank of the node following exactly e' pseudotraversals. Define the *reserve_rank* for global node u to be: $reserve\_\text{rank}(u) = marker\_\text{rank}(u) - marker\_\text{rank}(v)$, and the reserve potential for u to be gp times its reserve_rank. The role of the reserve potential will become clear later.

As in the analysis of a global access, the cost of the segment that involves an increase in visibility on the part of the accessed item is at most $vp+2c$, per unit increase in visibility. The presence of lazy trees, as we will see, adds a further $4gp(\log 2b + 1)$ to the cost of the first access of sequence S. So, focusing on superblock B alone, the cost of the first access in S is bounded by

$$(4e' + 9)gp(\log 2b + 1) + vp + 2c, \tag{10}$$

and the cost of a subsequent access of S is bounded by

$$s' \cdot 2^{e'-1} + a_3\alpha(c') + b_3 + vp + 2c. \tag{11}$$

Also, for each access, there are fixed costs of (see (1))

$$ld + 3sd. \tag{12}$$

Each node removed from the access paths is called a *lazy node*, whether or not it has a lazy potential. Those lazy nodes with a lazy potential are called *heavy* nodes; the other lazy nodes are called *light* nodes. When the accesses of sequence S are completed, we form *lazy trees*. Each global node v remaining on the right access path becomes the root of a new *right lazy tree*. Its left subtree is empty; its right subtree comprises the lazy nodes, created during the accesses of sequence S, in v's right subtree in the splay tree. An analogous definition is made with respect to the left access path. It is straightforward that each new lazy node is contained in a new lazy tree. We call the lazy trees as defined above *initial lazy trees*. (We will be adding and removing a few nodes from the initial lazy tree in order to obtain other lazy trees, which are the lazy trees that are actually analyzed.) As we will see, the local nodes on the access paths, which have been pseudotraversed e' times, may be added to the new lazy trees; these nodes too are called light nodes of their lazy trees. We will need to ensure these nodes also carry no debits and have an associated potential of q_1 units. But this is ensured by requiring

$$s'(2^{e'-1} - 1) \geq q_1 + \max\{sd, ld, hd, ed\}, \tag{13}$$

where hd and ed are the values of other debits (huge and enormous debits) introduced later.

Remark 2. We will need to generalize this analysis to take account of the presence of lazy trees on the access paths. The present analysis, with (9) replaced by (14) below, will continue to apply if the following conditions hold.

(i) Each node on the access path, which has been pseudotraversed e' times, has spare potential $(2^{e'-1} - 1)s$.

(ii) A lazy tree node, on creation, carries no debit.

(iii) Following the first e' pseudotraversals of a node v on the access path, if v's couple includes just one node on the extreme path of a lazy tree, the cost of the rotation of this couple is the same as in the present analysis.

(iv) Following the first e' pseudotraversals of a node v on the access path, if v's couple includes two nodes on the extreme path of a lazy tree, the cost of the rotation of this couple is at most $\max\{hd, ed\}$ units. The additional cost of the couple is covered by the spare potential if

$$s'(2^{e'-1} - 1) \geq gp + q_1 + \max\{hd, ed\}. \tag{14}$$

(i) and (ii) will be achieved as here. (iii) will be seen to be true shortly. (iv) will be handled later, in Lemma 14.

Next, we describe the potential provided to the lazy trees and show how to include the lazy trees in the overall analysis.

3.2. The analysis of lazy trees. In the following subsections, unless we specify otherwise, the definitions given apply to right lazy trees. Analogous definitions hold for left lazy trees. We focus on the right lazy trees; we discuss the left lazy trees only where their presence affects the analysis (the only place this arises is in section 3.6).

Much of the analysis focuses on a subtree of the initial lazy tree, called the *truncated lazy tree* or the *lazy tree*, for short. It is defined as follows. Consider a new initial lazy tree, L, created by the sequence S of accesses. Consider the set of heavy nodes in L; the tree they induce is called the *initial lazy block tree*. We remove the rightmost node from the initial lazy block tree; this defines the (truncated) lazy block tree for the (truncated) lazy tree. This rightmost node is called the *right guard* for the lazy tree. The *left guard* is a global node, defined later, to the left of the nodes of the lazy block tree. The root of the (truncated) lazy block tree is called the (truncated) root of the (truncated) lazy tree. We define the tree induced by the nodes of the lazy block tree plus the left and right guards to form the *large lazy block tree*. Now we define the (truncated) lazy tree as follows. It comprises the nodes of the (truncated) lazy block tree together with the following light nodes. For each node v in the (truncated) lazy block tree we add the following nodes from v's block to the (truncated) lazy tree. Let w be any descendant of v in the large lazy block tree. Those nodes in v's block on the path from v to w in the splay tree are added to the (truncated) lazy tree. The light nodes added to the lazy tree form its *skeleton*.

We define the *left guard*, w, of the lazy tree as follows. Let L be a new lazy tree with root u. Let v be the first proper global descendant of u on the portion of the right access path which has been traversed at least e' times, if any. Suppose v exists; if v is the root of another new lazy tree, let w be the right guard in this lazy tree, while if v is not the root of a new lazy tree, then let $v = w$. Otherwise, let w be the root of the splay tree. In general, a node may be a right guard for one lazy tree and a left guard for a second lazy tree.

Intuitively, a truncated lazy tree is a megablock comprising several of the blocks at hand. The megablock is considered to be at level $i - 1$, the level of the blocks it comprises. In its interactions with the remainder of the splay tree the megablock will behave in the same way as a block. The root of the lazy tree corresponds to the block root and behaves like a global node; the other nodes of the lazy tree, called *local* nodes of the lazy tree, correspond to the local nodes from the block. The lazy tree local nodes may carry small and large debits according to the invariants specified for blocks; a further *lazy* debit may be carried as specified later.

The right guard of each new lazy tree has its global potential restored; this is paid for by adding its reserve potential to its lazy potential, which suffices by Invariant 24, in section 3.5. In addition, Invariants 5 and 6 may need to be restored; this requires the removal of at most two small debits. These are paid for by the potential $a_3 + b_3$ associated with each lazy node on paths P_1, P_2, P_3 (see sections 3.3.3 and 3.3.4); these paths always include the right guard. Thus both the left and right guards of each lazy tree carry their global potentials.

We need one more definition: a *rightist tree* is a tree in which the depths of the leaves, from left to right, are nondecreasing. Now, we describe the structure of a typical lazy tree. Consider a newly constructed initial right lazy tree, IL, and the associated lazy tree, L. The main difference between L and IL is that IL may have (many) light nodes which are not in L. The tree induced by the nodes of L comprises its left extreme path (a sequence of light nodes), its root, and a path, P, descending to the right, together with the left subtrees of the nodes on P; each of these left subtrees is a rightist tree (strictly speaking, if v is the root of such a left subtree, then the tree rooted at v in IL is a rightist tree). It is helpful to partition P, the left extreme path and the root into four portions, BL, P_1, P_2, and P_3. BL comprises the light nodes on the skeleton from the leftmost block in L (initially, BL comprises the light nodes on the left extreme path); P_1 is the maximal top contiguous portion of P incident on the root of the lazy tree, minus those nodes in BL, P_2 is the second highest contiguous portion of P below P_1, and P_3 is the remainder of P below $P2$. It is also helpful, for each subtree of P, a rightist tree, R, to partition R into its top right path RP, and the left subtrees of RP, also rightist subtrees (again, strictly speaking, it is the corresponding subtrees in IL which are rightist); let SR denote a typical such rightist subtree (*small rightist subtree*). Now, to analyze a lazy tree, we analyze rotations in each of its six types of components separately; the six types of components are

(i) BL, comprising nodes of type 1,
(ii) the path P_1, comprising nodes of type 2,
(iii) the path P_2, comprising nodes of type 3,
(iv) the path P_3, comprising nodes of type 4,
(v) each path RP, comprising nodes of type 5,
(vi) each rightist subtree SR, comprising nodes of type 6.

The following invariant describes the distribution of the six types of nodes.

INVARIANT 10. *Let L be a right lazy tree. Let v be a node in L. The following nodes all have type equal to or less than type(v):*

(a) *the nodes on the left extreme path of L, other than v's ancestors,*
(b) *the left ancestors of v in L, apart from the root of L.*

In particular, traversing the extreme paths of the lazy tree from the bottom of the right extreme path to the bottom of the left extreme path, including the root of the lazy tree, yields, in order,

(i) a path from a tree SR,
(ii) a portion of a path RP,
(iii) a portion of P_3,
(iv) a portion of P_2,
(v) a portion of P_1,
(vi) a portion of BL.

The portions need not be contiguous portions of the original paths, and some or all of the portions may be empty. There are only two ways in which the type of a node can change: first, a node can leave the lazy tree; second, when the lazy tree is split,

a node may move into a component BL.

The structure of a left lazy tree is entirely analogous.

Our analysis is presented in three parts. First, for each component of the lazy tree, we describe the potentials associated with its nodes. Second, we analyze a traversal of an extreme path of a lazy tree. Third, we analyze a split of a lazy tree. For the moment, this discussion will exclude the possibility of multiple lazy trees contained within one another; in section 3.6 we show how to accommodate this possibility.

Before entering into the details of the potentials it is helpful to introduce the normal form of a (right) lazy tree L. It is obtained by performing the following series of single rotations: one by one, the left path nodes are moved to the right path; each such rotation, between node v and node r, the root of the lazy tree, makes v the root and places r on the right path. r and v interchange potentials and all lazy tree properties. The resulting tree is called the *normal tree* for L.

3.3. The potentials for the components of the lazy tree.

3.3.1. The rightist subtrees SR. We use the potential defined for the lazy trees present in the result on splay sorting $\log n$-block sequences, where it is called the *lazy complete tree potential*. To define this potential we need to introduce the *component lazy block tree*. It is the tree induced by the heavy nodes in the component SR together with the root of the lazy tree. A node of height h in the component lazy block tree, other than the root, is given potential $\frac{1}{2}a_1 \sum_{i=1}^{h} i(i+1) + b_1$, a_1 and b_1 being constants to be specified, while a light node in the subtree SR is given potential c_1, c_1 being another constant to be specified.

Now, we verify that there is enough potential at hand to initialize the trees SR with their correct potential. Let ISR denote the subtree in the initial lazy tree corresponding to SR (the subtree with the same root as SR). We start by giving a node of height k in ISR potential $\frac{1}{2}a_1 \sum_{i=1}^{k} i(i+1) + b_1$, regardless of whether the node is heavy or light. Since, for each heavy node, its height in ISR is at least as large as its height in the component lazy block tree, the heavy nodes will receive sufficient potential. To ensure adequate potential for the light nodes (potential c_1) it suffices to ensure

$$c_1 \leq a_1 + b_1. \tag{15}$$

We explain how to provide the initial potential of $\frac{1}{2}a_1 \sum_{i=1}^{k} i(i+1)+b_1$ to the nodes of ISR. We exploit the fact that the path IRP (the right path in IL corresponding to RP) together with its left subtrees is also a rightist subtree. A nonleaf node, v transfers all but b_1 of the charge for its potential to the following node, w, to its right (see Figure 6). Suppose v's first right ancestor, u, is reached by following e edges to the left and then an edge to the right. Then w is defined to be the descendant of u reached from u's right child by following $e+1$ left edges (edges to left children). The charge, $\frac{1}{2}a_1 h(h+1)$, for a node v at height h, is transfered to a node w at *minimum height* at least $h-1$, where the minimum height of a node is the length, in vertices, of the path to its leftmost descendant. Thus, a node v at minimum height k receives a charge of at most $\frac{1}{2}a_1(k+1)(k+2)$. This charge can be distributed evenly among the nodes by passing a charge of $a_1 \cdot (\frac{1}{2}k^2 + \frac{5}{2}k - \frac{7}{2})$ to each of v's children, if v has minimum height greater than 1 (note that v will receive a similar charge of at most $a_1 \cdot (\frac{1}{2}(k+1)^2 + \frac{5}{2}(k+1) - \frac{7}{2})$ from its parent, for v's parent has minimum height at most $k+1$); also, v keeps a charge of $\frac{15}{2}a_1 + b_1$ locally. A node at minimum height 1 receives, at most, a transfered charge of $3a_1$, a charge of $\frac{7}{2}a_1$ from its parent, and a

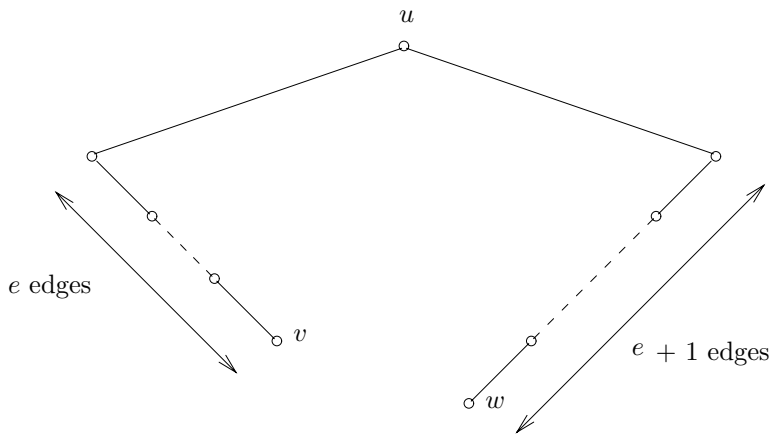


FIG. 6. *The node w receiving potential.*

direct charge of a_1+b_1; this is a total charge of at most $\frac{15}{2}a_1+b_1$. So we see that each node is charged at most $\frac{15}{2}a_1+b_1$. Finally, we note that the charges are transfered only among nodes in the trees ISR.

So, as the lazy tree is being constructed, it suffices to provide each node in a tree ISR with potential $\frac{15}{2}a_1+b_1$; this is guaranteed by having

$$q_1 \geq \frac{15}{2}a_1 + b_1. \tag{16}$$

3.3.2. The paths *RP*. Consider a path, RP. Initially, it comprises a contiguous sequence of nodes, descending to the right. As it is traversed, roughly speaking, it will become a binary tree, at which point the potential for rightist lazy trees becomes adequate. Thus our goal is to provide a potential that progressively converts to a rightist lazy tree potential as the path is repeatedly traversed. As we will see, an initial potential of a_2+b_2 for each node suffices, where a_2 and b_2 are constants to be specified. So it suffices to have

$$q_1 \geq a_2 + b_2. \tag{17}$$

3.3.3. The paths P_1 and P_2. They are treated in exactly the same way as the paths RP. To provide sufficient initial potential here it suffices to have

$$a_3\alpha(c_l) + b_3 \geq a_2 + b_2.$$

In turn, to satisfy this equation, the following suffices:

$$a_3 + b_3 \geq a_2 + b_2. \tag{18}$$

In fact, we modify the statement of section 3.1 that $a_3\alpha(c_l)+b_3$ potential is provided; instead only a_2+b_2 potential is provided, which still suffices. The reason for this modification will become clear in section 4.

3.3.4. The path P_3. This path cannot be treated in the same way as the paths RP for it does not necessarily comprise a contiguous sequence of nodes in the splay tree.

We maintain the following potential; each node on the current right extreme path of the tree induced by the root of the lazy tree plus P_3, with one class of exceptions, carries a potential of c_2, c_2 a constant to be specified. The exceptional nodes are formed by sequences of $4\alpha(c_l)$ contiguous nodes whose parent and right child in the splay tree are both from P_3; such nodes need not carry the c_2 potential. More precisely, each maximal contiguous sequence of nodes on the right extreme path of the tree induced by P_3, from bottom to top, has an initial segment of length one and a final segment of length at least one and at most $4\alpha(c_l)$ in which each node has a potential of c_2; the remainder of the sequence is partitioned into segments of length exactly $4\alpha(c_l)$, and in each such segment either all or none of the nodes carry a potential of c_2. Each node on the current extreme left path of the tree induced by the root of the lazy tree plus P_3 carries a potential of c_2, with no exceptions.

In addition, P_3 has an associated potential, $Pot(P_3)$; before specifying this potential we need to introduce some other definitions and results. A k-right cascade is a sequence of k single left rotations (a rotation between a node and its right child) on a set of k contiguous couples on a right path. A *right k-cascade sequence* is an intermixed sequence of k-right cascades and arbitrary left rotations. We recall that Sundar [Su89] showed that there can be at most $8m$ $2\alpha(m)$-right cascades in a right $2\alpha(m)$-cascade sequence on the normal form of an m-node binary tree. We define $Pot(P_3) = 4c_2\lambda\alpha(c_l)$, where λ denotes the maximum number of $2\alpha(c_l)$-right cascades in a right $2\alpha(c_l)$-cascade sequence that can be performed on the normal form of the tree induced by the nodes of P_3 and the root of the lazy tree, with the root of the lazy tree removed from the normal form (i.e., the cascades never include the root node); this tree comprises at most c_l nodes. The initial values of λ is at most $8|P_3|$, and so $Pot(P_3) \leq 32c_2\alpha(c_l)|P_3|$.

Finally, each node in P_3 carries an additional potential of $4c_2\alpha(c_l) + c_2$. Thus it suffices to provide each node in P_3, on creation, with potential $36c_2\alpha(c_l) + 2c_2$; so it suffices to have

$$a_3\alpha(c_l) + b_3 \geq 36c_2\alpha(c_l) + 2c_2.$$

That is, it suffices to have

$$a_3 \geq 36c_2, \tag{19}$$

$$b_3 \geq 2c_2. \tag{20}$$

3.3.5. *BL*, the skeleton nodes in the leftmost block. Each node in the leftmost block of the lazy tree carries an additional potential of c_3 units. Nodes in the other blocks carry an additional potential of $c_3 + c_4$ units. In order to provide this potential initially, we augment each of q_1, $a_3 + b_3$, and b_3 by $c_3 + c_4$ units. So it suffices to have

$$q_1 \geq \max\left\{\frac{15}{2}a_1 + b_1, a_2 + b_2\right\} + (c_3 + c_4), \tag{21}$$

$$a_3 + b_3 \geq a_2 + b_2 + (c_3 + c_4), \tag{22}$$

$$b_3 \geq 2c_2 + (c_3 + c_4). \tag{23}$$

3.3.6. Debits on the local nodes of the lazy tree. Nodes on the extreme paths of the lazy tree may have a small or large debit; these debits satisfy Invariants 3–6, where lazy tree L replaces block B in the statement of the invariants. Also, each node in the lazy tree can be considered to be labeled. A labeling potential is not needed here, however, for the removal of nodes from an external path of the lazy tree is always paid for in some other way, as we shall see. However, so that we can apply Invariants 3–6, we will consider the nodes of a lazy tree to be labeled. In addition, each light node in the lazy tree may have a *lazy* debit, which is *huge* and has value hd units, $hd \geq ld$, a constant to be defined later. Lazy debits satisfy the following invariants.

INVARIANT 11. *Let L be a right (resp., left) lazy tree. Suppose node v in L has a lazy debit. Then v is a light node of L. Also, v is not on the left (resp., right) extreme path of L. Finally, v does not carry a small or large debit.*

INVARIANT 12. *Let L be a right (resp., left) lazy tree. Let u be a light node of L. Suppose u has a lazy debit. Let v be the root of u's block.*

(i) *Suppose that v is not on the left (resp., right) extreme path of L. Then if u is on the right (resp., left) path descending from v in the splay tree, both the parent and child of u on this path are local nodes in u's block and are in the same component as u.*

(ii) *If u is on the right (resp., left) extreme path of L, then its parent and child in the splay tree are local nodes in u's block and are in the same component as u; this holds regardless of whether u is on the right (resp., left) path descending from v in the splay tree.*

INVARIANT 13. *BL and P_3 do not carry lazy debits.*

In addition, each node in the lazy tree may have a *border* debit, which is enormous and has value ed; ed is a constant to be defined later. Border debits satisfy the following invariants.

INVARIANT 14. *Let L be a right (resp., left) lazy tree. Suppose node v in L has a border debit. Then v is not the root of L and is not on the left (resp., right) extreme path of L. Further, if v is on the right (resp., left) extreme path of L, then both v's parent and right (resp., left) child in the splay tree are in the same component of L.*

INVARIANT 15. *A node has at most one debit.*

Next, we show how to incorporate lazy trees into the analysis of accesses. An access can traverse a lazy tree in one of three ways:

(a) traverse the right extreme path of the lazy tree (or rather a topmost portion of it),

(b) traverse the left extreme path of the lazy tree (or rather a topmost portion of it),

(c) traverse the interior of the lazy tree and thereby split the lazy tree.

Actually, it is convenient to classify a traversal of type (a) which is to the left of the block of the right guard to be a split (a type (c) traversal); likewise a traversal of type (b) to the right of the block of the left guard is defined to be a split.

A traversal of type (c) will be paid for in two phases. First, in a preprocessing phase, the current lazy tree is partitioned into several lazy trees and/or ordinary blocks, so as to ensure that the actual splay (the second phase) comprises only traversals of types (a) and (b). (In fact, as we will see, we need a third phase in order to pay for some of the partitioning performed in the first phase.)

We start by considering the interactions between the lazy tree and the remainder of the splay tree (which may include other lazy trees). We treat the lazy tree, as

delimited by its extreme paths, as a block. We note that Invariant 12 ensures that a node on an extreme path, carrying a lazy debit, satisfies condition (i) of Remark 1. In addition, we note that on creation, the nodes of the lazy tree have no debits; so Invariants 3–6 all hold at this point.

Next, we need to show that condition (ii) of Remark 1 is satisfied. That is we have to show that couples involving two nodes on an extreme path of the lazy tree, which will be marked, can pay for the removal of their debits and $s+1$ rotations. This is done in section 3.4.

Segments are defined and paid for essentially as before. The one difference occurs if either node of the leading couple of the segment carries a lazy debit; then the couple pays for its own rotation and the removal of its debits as part of the lazy tree, as we see later; but this can only reduce the total remaining cost of the segment and so the bounds of the previous analysis are still valid.

We need to mention one detail about couples containing the root u of a lazy tree and another node v in the same lazy tree. Following the rotation, v becomes the root of the lazy tree; v and u interchange roles and potentials (so if v had been light, resp., heavy, u becomes light, resp., heavy).

In section 3.5 we explain how a lazy tree is split.

3.4. Extreme path traversal. We consider all rotations involving two nodes on an extreme path. So consider a couple comprising nodes u and v, where u is the parent of v. We distinguish the following types of couples.

Case 1. u and v are both nodes of P_3. If both u and v carry c_2 potentials, then the potential associated with u, which ceases to be on an extreme path of P_3, is used to pay for the rotation, s spares, and the removal of any debits, which do not include lazy debits (see Invariant 13). So it suffices to have

$$c_2 \geq (s+1) + \max\{2ed, ld\}. \tag{24}$$

If one or both of u and v do not carry a c_2 potential then they are part of a contiguous sequence of $4\alpha(c_l)$ nodes lacking this potential. This sequence undergoes a $2\alpha(c_l)$-right cascade (or perhaps this sequence of nodes shifted by one). The cascade provides c_2 potential to each of these $4\alpha(c_l)$ nodes; we can then handle the rotation between u and v as before. The $4c_2\alpha(c_l)$ cost of the cascade is charged to P_3 (recall that P_3 had been given sufficient potential to pay for the maximum possible number of such cascades).

Case 2. u and v are both nodes of BL. The c_3 potential associated with u, which ceases to be on BL, is used to pay for the rotation, s spares and the removal of any debits, which do not include lazy debits (see Invariant 13). So it suffices to have

$$c_3 \geq (s+1) + \max\{2ed, ld\}. \tag{25}$$

Case 3. u and v are in distinct components. Then u is given a border debit. This pays for the rotation, s spares, and the removal of debits from u and v (note that by Invariants 14 and 12 u and v carry neither lazy nor border debits). So it suffices to have

$$ed \geq (s+1) + ld. \tag{26}$$

Case 4. u and v are in the same tree SR, a *rightist lazy tree*. Again, without loss of generality, we are only considering right lazy trees. The analysis is similar to

that in splay sorting $\log n$-block sequences but the possible presence of border debits changes the cost of the various rotations.

Case 4.1. u and v are both light nodes. u ceases to be on the skeleton. u's c_1 potential pays for the rotation, s spares, and the removal of debits on u and v. So it suffices to have

$$c_1 \geq s + 1 + \max\{2hd, 2ed, ld\} = s + 1 + \max\{2hd, 2ed\}. \tag{27}$$

Case 4.2. u is a light node and v is a heavy node. By Invariant 12, u does not have a lazy debit. If u leaves the skeleton, the operation is paid for by u's c_1 potential, as in Case 4.1; (27) suffices. Otherwise, u is given a lazy debit; this then pays for the operation. The cost of the operation comprises the rotation, s spares, and the removal of the small, large or border debits, if any, from u and v. So it suffices to have

$$hd \geq s + 1 + \max\{ld, 2ed\} = s + 1 + 2ed. \tag{28}$$

Case 4.3. v is a light node, u is the root of v's subblock but is not the lazy tree root. By Invariants 11 and 12, u and v do not have lazy debits. v becomes the block root. As in Case 4.2, if u leaves the skeleton, the operation is paid for by u's c_1 potential. Otherwise, u is given a lazy debit, which pays for the operation. The cost of the operation comprises the rotation, s spares, and the removal of the path or border debits, if any, from u and v. Here too (27) and (28) suffice.

Case 4.4. u and v are both heavy nodes. We need to pay for the rotation, for s spares, and for the removal of path or border debits from u and v, if any. In addition, we may need to reestablish Invariant 12 for the nodes on the path newly abutting u; this may require the removal of up to two lazy debits, which will also be paid for by the rotation. So the cost of this rotation is bounded by

$$s + 1 + 2hd + 2ed. \tag{29}$$

The analysis of this case is identical to that of the corresponding case in the splay sorting of $\log n$-block sequences [CMSS00]; it is omitted in part. However, in order to understand the analysis of Case 5, it is helpful to repeat it in part.

For the remainder of this case, until we indicate otherwise, when we refer to a tree we intend the component lazy block tree. So, for instance, when we refer to a node we mean a node in the component lazy block tree.

We start with some definitions. The *depth* of a node is its distance from the root. The height of a node at the time of the creation of the lazy tree is called its *creation-height* (a leaf has creation-height 1); the creation-height does not change subsequently.

Consider how an extreme path traversal appears to the tree. A top contiguous portion of the nodes on this path are all *touched* (these are the nodes contained in couples of the splay tree). Some (arbitrary) subset of disjoint pairs of touched nodes form couples. Each touched node is provided with $\frac{s'}{2}$ spare units of potential (this is the node's share of the portion of s' spares provided to its couple in the splay tree). The spare potential of the touched nodes, together with any changes to the potentials of the touched couples will pay for the rotations of the touched couples.

At any time, certain nodes, called *active* nodes, are the nodes that pay for a traversal of an extreme path. Some nodes may pay more than other nodes. This is captured by the notion of *active layers*. A node of creation-height h has an associated

span of layers $[1, h]$; each active node v of creation-height h has an *active span* of *active layers*, $(g, h]$, $0 \leq g \leq h$; for each k, $g \leq k \leq h$, we say v is *k-active*. If the active span is nonempty, we say the node is *active*.

Initially, only the nodes on the right path are active. An inactive node becomes active when it first reaches the right path. Once a node becomes active it remains active, whether or not it remains on the right path; also, the active span of a node can only grow.

A rotation of the splay operation involving a heavy node, v, and the root, r, of the lazy tree causes the two nodes to swap all their potentials. Also r acquires v's creation-height and active span.

The following invariant states several properties of the active nodes. We prove the invariant later. In order to avoid special cases for the left path we state our invariants with respect to the normal form of the tree, where the tree comprises the root of the lazy tree together with the component lazy block tree.

INVARIANT 16. *Let H be the maximum creation-height for the nodes, other than the root, present in the tree initially. Then, in the corresponding normal tree, the following hold:*

(i) *There is exactly one k-active node, $1 \leq k \leq H$.*

(ii) *Every node on the right path is active (this does not include the root).*

(iii) *Apart from the root, the ancestors of an active node are all active.*

(iv) *Let v be a k-active node. Let w be a j-active node. If $j < k$, then w is to the right of v in symmetric order, while if $j > k$, then w is to the left of v in symmetric order.*

(v) *Let inactive node v have creation-height k. Then its parent has creation-height greater than k.*

When a lazy tree is created the active spans for the nodes in the corresponding component lazy block tree are initialized as follows. Let u be a node on the right path, of creation-height h; suppose it has a right child v of creation-height g (if there is no such node v let $g = 0$). Then u is given active span $[g + 1, h]$. Clearly, the new tree obeys Invariant 16.

The layers of a node are further categorized as *black* or *white*; an active node, with active span $[g, h]$, can be black with respect to each of the layers $[1, g - 1]$. In general, an active node v, with active span $[g, h]$, is black with respect to all the layers in some range $[f, g - 1]$, $f \geq 1$, called its *black span*; we say v is *k-black*, for $f \leq k \leq g - 1$. If the black span is nonempty we say the node is *black*. Nodes are initially white at all layers. A node becomes black as a consequence of a rotation with the root. The following invariant applies to black nodes.

INVARIANT 17.

(i) *All the nodes on the left path of a tree are fully black, i.e., a node with active span $[g, h]$ has black span $[1, g - 1]$.*

(ii) *If a node u is k-black all of u's left ancestors, apart from the root, in the corresponding normal tree are k-black.*

We define the following distances for each node v, active at layer k. Its *right path k-distance*, $d_k(v)$, is the number of proper left ancestors of v on the right path, excluding all k-black nodes. Its *interior right path k-distance*, $id_k(v)$, is the number of proper left ancestors of v below the first k-black node and below the right path.

For each k-active node we maintain a k-potential, which satisfies the following invariant (d_1 is a constant to be specified).

INVARIANT 18. *The k-potential at k-active node v is at least*

$$a_1 \cdot \min\{d_k(v)/d_1, k\} + a_1 i d_k(v)/d_1.$$

We note that initially a node has $a_1 \cdot k$ units of potential for each layer k at which it could become active, so when a node first becomes k-active, Invariant 18 holds.

For each k-black node we maintain a k-black potential of $\frac{a_1}{2d_1}$ units.

We conclude by stating the sufficient conditions that are imposed by the analysis of [CMSS00] (in the previous analysis a and d replace the present a_1 and d_1, respectively).

$$d_1 s'/2 \geq \max\{s + 1 + 2hd + 2ed, a_1/d_1\}. \tag{30}$$

$$\frac{1}{2} a_1/d_1 \geq s + 1 + 2hd + 2ed. \tag{31}$$

There is one matter related to this analysis which needs detailing. Immediately prior to a traversal of the right path, each node on that path is given $s'/2$ spare rotations (whether or not the node is part of a couple comprising two heavy nodes of the same lazy tree component SR). Each node's spares are subsequently provided by the rotation which involves that node. The $s'/2$ spares associated with each node are redistributed in order to pay for those rotations which are not self paying. The details can be found in [CMSS00].

Case 5. u and v are both in a component RP (or are both in P_1 or P_2).

We begin by describing the potential associated with the nodes of component RP.

The nodes originally forming the path RP are partitioned into three sets:

(i) those nodes on a shortened path, RP', called *pure path nodes*,

(ii) *rare black nodes*,

(iii) *rightist nodes*.

The following invariants apply to the path RP.

INVARIANT 19. *The nodes in RP' form a contiguous path in the splay tree.*

INVARIANT 20. *After t traversals, a node on RP' has potential $a_2 \sum_{j=1}^{t+1} j + b_2$, $a_2 \geq a_1$ and $b_2 \geq b_1$ being constants to be specified.*

INVARIANT 21. *Let N be the normal form of the lazy tree L. Let v be a node in RP, a part of L.*

(i) *Suppose that v is a rare black node. Each of v's left ancestors in N which is also in RP is also a rare black node.*

(ii) *Suppose that v is in RP'. Then each of v's left ancestors in N which is also in RP is either in RP' or is a rare black node.*

INVARIANT 22. $|RP'| \neq 1$.

Each rightist node has a creation-height and corresponding potentials satisfying Invariants 16–18. Each rare black node also has a creation-height; these black nodes have the *rareness property*, namely there are no other nodes in this component RP with the same creation-height. The rare black nodes carry k-black potentials and k-potentials; they too satisfy Invariants 16–18. Owing to the rareness property we are able to reduce their initial values somewhat. However, it is easier to defer the precise specification of these potentials till Cases 5.2 and 5.3, where we explain how the potentials change as the rotations analyzed in these cases are performed. The nodes in RP' acquire creation-heights and appropriate rightist lazy tree potentials as and when they leave RP', either to become rare black nodes or rightist nodes.

Initially, $RP = RP'$. So an initial potential of a_2+b_2 for each node in RP suffices, as claimed in (17).

Case 5.1. Neither u nor v is in RP'. This is treated as a rotation in a rightist lazy tree. It is handled exactly as in Case 4.

Case 5.2. u and v are pure path nodes (i.e., both are on RP'). We require that the cost of the rotation be at most $s+1+ld$. This follows by ensuring that the pure path nodes carry no border or lazy debits. So we add the following invariant.

INVARIANT 23. *Let u be a pure path node in a lazy tree. Then u does not carry a border or lazy debit.* We note that Invariant 23 can be maintained by trimming the ends from RP' whenever an end is part of a couple including a node outside RP'. So if, in couple (u, v), with u the parent of v, v is the top node on RP', v is converted to a rare black node (by the method of Case 5.3, below); while if u is the bottom node on RP', u is converted to a rightist node (by the method of Case 5.4, below).

Suppose that this is the $(t+1)$th traversal of u and v. Then their potentials are adjusted as follows. u, which leaves RP', receives potential $a_1 \sum_{j=1}^{t+1} j + b_1$ and becomes a rightist node with creation-height $t+1$ (at this point we further reduce u's potential if it is light—see section 3.3.1). v remains on RP' and so its potential increases to $a_2 \sum_{j=1}^{t+2} j + b_2$ (see Invariant 20); the rotation itself costs at most $s+1+ld$ units. So it suffices to have

$$a_1 \sum_{j=1}^{t+1} j + b_1 + a_2 \sum_{j=1}^{t+2} j + b_2 + (s+1+ld) \leq 2a_2 \sum_{j=1}^{t+1} j + 2b_2. \tag{32}$$

This equation is satisfied by setting

$$a_2 \geq 2a_1, \tag{33}$$

$$b_2 \geq b_1 + (s+1+ld) + \frac{3}{2}a_2. \tag{34}$$

If v is now the only node on RP', it is converted to a rightist node, with creation-height $t+2$, by reducing its potential to $a_1 \sum_{j=1}^{t+2} j + b_1$ (note Invariant 22).

Note that the creation-heights for rightist nodes obey Invariant 16(v).

Case 5.3. v is on RP', but u is not on RP'. We provide v with appropriate j-potential and j-black potential; v becomes a node of the complete rightist lazy tree to be formed from the initial path RP. In particular, v becomes a rare black node (which behaves in the same way as a black node in a rightist lazy tree). The rotation between u and v is paid for as in a rightist lazy tree rotation (see Case 4 above).

It remains to explain how to provide v with sufficient potential. Suppose the path, RP', immediately before the traversal has length $2^l \leq |RP'| < 2^{l+1}$; further suppose that the nodes on RP' have been traversed t times so far. Then v receives creation-height $h(v) = 2l+t+1$. v requires $\frac{a_1}{2d_1}(h(v)-1) = \frac{a_1}{2d_1}(2l+t)$ units of potential for its k-black potentials; it requires a constant potential of b_1 units; finally, it requires $a_1 \sum_{j=1}^{2l+t} \min\{t, j\}$ units of j-potential. (At first sight, one would expect a quantity $\frac{1}{2}a_1 h(v)(h(v)+1)$. But, in fact, we can reduce this total for two reasons.

(i) The $h(v)$-potential can be set to zero for node v, since $ld_{h(v)}(v) = 0$.

(ii) Only $\min\{t, j\}$ units of j-potential are required for the other values of j, since v can have at most t left ancestors in its component in the normal form of the lazy tree when and if it becomes j-active.

Note that t increases by 1 each traversal, while l decreases by at least 1 each traversal. So the creation-heights of the rare black nodes ordered by time of creation are strictly decreasing; also, they are larger than the creation-heights of all the nonrare rightist nodes that are eventually created—the nonrare rightist node with largest creation-height is the node which is eventually alone on the path RP'; it has creation-height at most $l+t+1$.) As the rare black nodes are also ordered left to right by their creation time, as well as being to the left of all nonrare nodes, Invariant 16(v) is satisfied by the rare black nodes too.

Assuming that

$$d_1 \geq 1, \tag{35}$$

the total potential needed for v is at most $a_1(l + t/2) + \frac{1}{2}a_1t(t + 1) + 2a_1l \cdot t + b_1$, which is $a_1[l(2t + 1) + \frac{1}{2}(t^2 + 2t)] + b_1$. This potential is provided by the potential already at hand for node v plus a charge of $a_1(2t + 1)$ to each node that leaves the path RP'. v already has a potential of $\frac{1}{2}a_2(t + 1)(t + 2) + b_2$ at hand; this covers $\frac{1}{2}a_1(t^2+2t)+b_1+a_1(2t+1)$ units of v's future potential (since $a_2 \geq 2a_1$ and $b_2 \geq b_1$); For $l \geq 1$, $2^{l-1} \geq l$; as there are at least $2^{l-1} - 1$ nodes leaving RP', the charge to the leaving nodes covers the remaining $a_1(l - 1)(2t + 1)$ units of v's future potential. To cover this charge to the leaving nodes it suffices to modify (32) as follows:

$$a_1 \sum_{j=1}^{t+1} j + b_1 + a_2 \sum_{j=1}^{t+2} j + a_1(2t + 1) + b_2 + (s + 1 + ld) \leq 2a_2 \sum_{j=1}^{t+1} j + 2b_2.$$

This equation is satisfied by setting

$$a_2 \geq 2a_1, \tag{36}$$

$$b_2 \geq b_1 + (s + 1 + ld) + 7a_1. \tag{37}$$

If v is a light node, its k-potentials and k-black potentials are replaced by a potential of c_1; (15) guarantees that there is sufficient potential at hand.

Case 5.4. u is on the path RP', but v is not. Then u is made into a rightist node with creation-height $t+1$; the potential presently associated with u is larger than the potential it subsequently requires. If u is a light node its potential is reduced to c_1; (15) guarantees that there is sufficient potential at hand. The rotation is then treated as a rotation in a rightist lazy tree; this is handled by Case 4.

It order to justify the claim made in Remark 2 we need to report the amortized cost of the rotations occurring in the traversal of a lazy tree extreme path.

LEMMA 14. *The cost of each rotation of a couple comprising two nodes on the extreme path of a lazy tree is at most* $\max\{hd, ed\}$.

Proof. In section 3.4 we have considered all couples comprising two nodes on an extreme path. Cases 1, 2, 4.1, 4.4, and 5.2–5.4 are all self-paying (that is, they do not require the creation of debits). Case 5.1 is subsumed by Case 4. Case 3 has cost at most ed and Cases 4.2 and 4.3 have cost at most hd. □

3.4.1. Rotations with the lazy tree root. In a rotation with the lazy tree root, the root and the other node in the couple interchange lazy tree potentials. Debits are removed and created as for normal rotations on the inner skeleton of a block.

3.5. Splitting the lazy tree. A split of the lazy tree occurs when the accessed item lies strictly between the subblocks of the left guard and the right guard of the lazy tree. Thus a split must access a new i-block. So a split is the first access of a new sequence S_l. For this section, we assume that the open intervals spanned by the guards of each lazy tree are disjoint. In section 3.6, we analyze the general case. Without loss of generality, we suppose the lazy tree being split is a right lazy tree.

At this point it is helpful to mention a few properties of the lazy and reserve potentials. Consider the global nodes in a new lazy tree. Let SL be the set of nodes contained strictly between the blocks of the guards of a lazy tree. For each heavy node, u, in the lazy tree, define its right neighbor $r\_n(u)$ to be the heavy node immediately to its right in the large lazy block tree, and define $SL(u)$ to comprise the subset of SL strictly to the left of u's block. Finally, define the lazy rank of heavy node v to be $\frac{1}{gp}$ times its lazy potential.

INVARIANT 24. *Let v be a heavy node in the normal form of the lazy block tree for lazy tree L. Then the following hold:*

(i) $lazy\_\text{rank}(v) \geq \lfloor \log(wt(SL(v))) \rfloor$.

(ii) *If, in the normal form of the large lazy block tree, v has no heavy node or guard in its right subtree, $lazy\_\text{rank}(v)+1/gp \cdot reserve(v) \geq g\_\text{rank}(v)$. While if v does have a heavy node or guard in its right subtree, then $lazy\_\text{rank}(v) + 1/gp \cdot reserve(v) \geq lazy\_\text{rank}(r\_n(v))$.*

Invariant 24 can be seen to hold when the lazy tree is created by considering node v at the point at which it becomes lazy. Also, it is readily seen that this invariant continues to hold following traversals of the extreme paths (for they leave the lazy rank of nodes in the normal form of the lazy tree unchanged).

COROLLARY 2. *Let v be a heavy node other than the root in lazy tree L. Let w be v's right child and u be v's left child.*

(i) *Suppose that v is on the left path of L; then $lazy\_\text{rank}(v) \geq g\_\text{rank}(w)$.*

(ii) *Suppose that v is not on the left path of L. Let v' be a right descendant of v, not necessarily proper. If v' is a heavy node, $lazy\_\text{rank}(v') \geq g\_\text{rank}(u)$.*

It is convenient to define v's *lazy weight* to be $wt(SL(v))$.

At this point, we appeal to the construction in [CMSS00] to show that the split can be performed. To do this we review Invariant 11, Properties 3 and 4, and Lemma 13 of that paper.

INVARIANT 25 (Invariant 11 of [CMSS00]). *Consider a right lazy tree containing node v. Suppose v is a light node and carries an additional debit (this is referring implicitly to a border debit). Then*

(i) *v is not on the left path of the (right) lazy tree.*

(ii) *If v is on the right path of the (right) lazy tree, then so are its parent and right child in the splay tree.*

An analogous invariant applies to left lazy trees.

PROPERTY 1 (Property 3 of [CMSS00]). *The restoration of any further invariants concerning debits (this refers implicitly to Invariants* 12 *and* 14*) requires the removal of at most α additional debits from each lazy tree created by the split, α a constant.*

LEMMA 15 (Lemma 13 of [CMSS00]). *If a lazy tree comprises light and heavy nodes where the heavy nodes carry lazy and reserve potentials satisfying Invariant* 24*, then each lazy tree resulting from the split described in* [CMSS00] *satisfies the following:*

(i) *If the nodes on its left path are new to a left path, then they are all light nodes in the same block.*

(ii) *Assume the debits obey Invariants* 3–6 *and* 13*, and the invariants implicit in Property* 1*, namely Invariants* 12 *and* 14*. If the following equations hold, then these invariants can be restored following a split:*

$$c' \geq \max\{4md, ld + 3md\} = 4md, \tag{38}$$

$$b \geq 3md + 2ld + 2sd + gp + \alpha \cdot md, \tag{39}$$

$$\text{where } md = \max\{ed, hd\}. \tag{40}$$

Note that $b = c' = \min\{c_3, c_4\}$*; they are the additional potentials associated with heavy and light nodes, respectively, to be used to cover the costs of removing these nodes from the lazy tree.*

PROPERTY 2 (Property 4 of [CMSS00]). *The potential must be partitionable following a split; i.e., each (component of each) lazy tree created by the split must be provided with an appropriate potential (e.g., a lazy complete tree potential).*

Clearly Invariant 25 is true of the border debits (see Invariant 14). We use the splitting method from [CMSS00]. To justify its use we need to bound α and prove Property 2. Both results rely on Lemma 15(i). Indeed, Property 2 is already known for the lazy complete tree potential. It remains to consider the other types of potential and to determine a bound on α.

The Path RP. The nodes not on path RP' are treated as ordinary rightist lazy tree nodes. The nodes on RP', are separated into two paths, RP'_1, RP'_2, by the split (one of these paths may be empty). If either path RP'_i is of length 1, then it is converted to a rightist node (that is, its potential is reduced from $a_2 \sum_{j=1}^{i+1} j + b_2$ to $a_1 \sum_{j=1}^{i+1} j + b_1$ for some i). We note that this restores Invariant 22. There is no other change to the potentials of the nodes on the paths RP'_i. It is clear that Invariants 19–21 and 23 continue to hold.

The paths P_1 and P_2 are treated in the same way.

The Path P_3. Following the partitioning we need to provide c_2 potential to each node which is newly on a right extreme path (unless it is part of a contiguous sequence of nodes of length at least $4\alpha(c_l)$). We note that there are no nodes newly on a left extreme path and in a P_3 component (for such nodes become part of a BL component). Each node, v, which had been in component P_3, and which has been promoted or is now in an extreme left subblock of a new lazy tree or is now in a normal subblock (a subblock that is no longer part of a lazy tree), may spend up to $c_2 \cdot (4\alpha(c_l) + 1)$ potential in order to provide c_2 potential for each of the up to $4\alpha(c_l) + 1$ following nodes: those nodes in the contiguous portion of the new right extreme path, if any, immediately below and to the left of v in the splay tree, selected so as to restore the proper distribution of c_2 potentials (see section 3.3.4); this is a portion of a right extreme path in component P_3 for a neighboring new lazy tree (the new component P_3 is a portion of the old component P_3). As node v is no longer in a P_3 component, it can afford to spend potential $4c_2\alpha(c_l) + c_2$.

Also, when the tree formed from P_3 is split, the potential associated with P_3 is partitioned in proportion to how many $2\alpha(c_l)$-right cascades can be performed on each tree formed from P_3 (by Sundar's bound [Su89], there is sufficient potential at hand). To see this, consider the tree induced by the original component P_3 plus the root of the splay tree in its normal form. We simulate the actual rotations as follows. Only traversals of extreme paths of current P_3 components are simulated, and then only on couples comprising two nodes of P_3. The effect is to maintain each current P_3 component in its normal form, so all extreme path traversals are

simulated as right cascade sequences. So the effect of a split is merely to reduce the possible sequences of right cascades; hence Sundar's bound applies, or in other words, the potential associated with P_3 suffices to provide each of its partitions with their associated potential.

Bounding α. The additional constraints on the debits are given in Invariants 12 and 14, in the requirement that neighbors of a node with a lazy or border debit be in the same component and not simply in the lazy tree.

For each lazy tree created by the split, for each component, we remove the lazy or border debits, if any, from the leading and tail nodes of the components on the extreme paths. Clearly, there are at most twelve such lazy debits for each new lazy tree. So we can choose $\alpha = 12$.

By Lemma 10 of [CMSS00] the cost of the promotions is bounded as follows.

LEMMA 16 (Lemma 10 of [CMSS00]). *The promotions in a split have the following cost: $2gp$ times the increase in global rank along the right split path plus $2gp$ times the increase in global rank along the left split path.*

COROLLARY 3. *Consider a single access A and the consequential promotions of nodes in lazy trees on the skeleton of superblock B. Let b be the number of blocks in superblock B. The promotions of access A cost at most $4gp(\log 2b + 2)$ units, there being a charge of $2gp(\log 2b + 1)$ to the right split path and of $2gp(\log 2b + 1)$ to the left split path.*

3.6. Multiple level lazy trees. Because the access paths for a sequence, S_i, of accesses to an i-block may include nodes of a current lazy tree, we may seek to make the root of a lazy tree a global node carrying a lazy potential in a new lazy tree. We therefore generalize the form of the lazy trees. Now, a "block" in a lazy tree may itself be another lazy tree. The generalization is exactly as in [CMSS00]; the analysis proceeds exactly as described there. Lemma 16 continues to hold. The only detail is that (38) needs to be replaced by (see (25) of [CMSS00])

$$c' \geq 4md + \alpha \cdot md = 16md. \tag{41}$$

Recall that $c' = \min\{c_3, c_4\}$ and $\alpha = 12$.

3.7. The full result. We pull together the various equations from the previous subsections in order to bound the cost of an access. First, we define an access to be of type i if the previous access is to a distinct i-block but to the same $(i+1)$-block. We show a bound of the form $f_1 \log b_{i+1} + f_2(\log\log n)^2 + f_3 \log\log n\alpha(n) + f_4$ on the amortized cost of a type i access, where f_i, $i = 1, 2, 3, 4$ are constants. (Recall that we are choosing $b_i = 2^{2^i}$.) Then we show how to replace the $\log b_{i+1}$ term by a $4 \log d$ term, where d is the distance between the currently accessed item and the previously accessed item.

To avoid tedious computations, we will assume that $n \geq 16$.

Recall the statement of Lemma 13; we chose $sd = s + 1$, $ld, c = 17(s + 1)$, $gp = 19(s + 1)$, and $vp = 30(s + 1)$. We need to satisfy the constraints present in (15)–(41). By choosing equalities in some of these constraints we satisfy them all, as we demonstrate. Equalities in (26), (28), and (40) yield $ed = 18(s + 1)$, $md = hd = 37(s + 1)$. Equality in (27) yields $c_1 = 75(s + 1)$, and equality in (39) yields $c_3 = c_4 = 610(s + 1)$ (this subsumes (25), (38), and (41)). Equality in (24) yields $c_2 = 37(s + 1)$. Now, we choose $s = 1$ and hence $s' = \frac{1}{3}$. With equalities in (30) and (31) we obtain $d_1 = 1332$ (which subsumes (35)), and $a_1 = 1332 \times 222(s + 1) = 295,704(s + 1)$ (this subsumes (15)). Next, we choose $b_1 = 0$.

With equalities in the following equations we obtain: (36) (which is identical to (33)) $a_2 = 591,408(s+1)$; (37), $b_2 = 2,069,946(s+1)$ (this subsumes (32) and (34)); (22), $a_3 + b_3 = 2,662,574(s+1)$, and on choosing $a_3 = b_3$, this subsumes (18)–(20) and (23); (21), $q_1 = 2,662,574(s+1)$ (this subsumes (16) and (17)). Finally, from (9), (13), and (14), we deduce that $e' = 24$ suffices.

Substituting these values into (10), (11), and (12), we obtain that the amortized cost for the portion of the traversal in which the accessed item has visibility i for the first access to an i-block is bounded by $4,000 \log 2b_{i+1} + 4,200$, and the amortized cost for a subsequent access is bounded by $[6,000,000 + 3,000,000\alpha(c_{i+1})]$. Summing over all levels, we obtain a bound for the first access in S_i of $8,000 \log b_{i+1} + \lceil \log \log n \rceil [6,000,000 + 3,000,000\alpha(n)]$ (for recall that $b_i = 2^{2^i}$).

Now, we show how to avoid the dependence on block boundaries, which so far has meant that a search a short distance from the previous access may be treated as if it were far away (if a boundary between two high level blocks is crossed). The idea is to average over many possible positions for the block boundaries so that if two nodes, u, v, are distance d apart (counting the number of items spanned by $(u, v]$ or $(v, u]$, according as $v > u$ or $v < u$) then, on average, the two items are contained in a block of size less than d^4.

More precisely, imagine an ordered set of $2n-1$ items, with the n items of the tree occupying n contiguous positions in the set. We define block boundaries on the set of $2n-1$ items, and consider the n positions that can be occupied by the items in the tree. For each such position, the block boundaries partition the tree of n items; the corresponding potential is associated with the tree. The overall potential for the tree is simply the average of these associated potentials.

Now it is straightforward to bound the cost of a search. Suppose the current access is distance $d \geq 16$ from the previous access. Suppose $2^{2^{k+1}} \leq d < 2^{2^{k+2}}$. As we have seen, the first access to a j-block, which we call a search of type j, costs $f_1 2^{j+1} + f_5 \lceil \log \log(2n-1) \rceil$, where $f_1 = 8,000$ and $f_5 = [6,000,000 + 3,000,000\alpha(2n-1)]$. For the n choices of block boundary at hand, let n_j denote the number of boundaries for which the current access is a search of type j. Then $n_{k+h} = d \cdot n / 2^{2^{k+h+1}}$. The cost of the search is therefore

$$\leq \frac{1}{n} \sum_{h \geq 1} f_1 2^{k+h+1} n_{k+h} + f_1 2^{k+1} + f_5 \lceil \log \log(2n-1) \rceil$$

$$\leq f_5 \lceil \log \log(2n-1) \rceil + f_1 \sum_{h \geq 1} 2^{k+h+1} \cdot d / 2^{2^{k+h+1}} + f_1 \cdot \log d$$

$$\leq f_5 \lceil \log \log(2n-1) \rceil + f_1 2^{k+1} \sum_{h \geq 1} 2^h \cdot 2^{2^{k+2}} / 2^{2^{k+h+1}} + f_1 \cdot \log d$$

$$\leq f_5 \lceil \log \log(2n-1) \rceil + f_1 \log d \cdot 4 = 4 f_1 \log d + f_5 \lceil \log \log(2n-1) \rceil.$$

For $d < 16$, the $\log d$ term can be absorbed into the $\log \log$ term.

So we have shown the following.

THEOREM 1. *Consider a sequence of searches performed on an n-node splay tree, $n \geq 16$. Let item v be the current item being accessed and u the previous item accessed. Suppose that the distance from u to v is d. Then the amortized cost of the access, in*

rotations, is bounded by $32,000 \log d + \lceil \log\log(2n-1) \rceil [6,000,000 + 3,000,000\alpha(2n-1)]$.

Finally, we need to consider the initialization costs. For each i-block, we need to provide its root with its global potential, unless it is the root of an i-superblock. Per i-block root, this costs at most $gp(2^{i+1}+1)$; there are at most $n/2^{2^{i+1}}+2$ such nodes, allowing for up to two undersize nodes. In addition, we need to provide the path and reserve path potentials, whose cost is bounded by $144c \cdot n$ as shown in Lemma 12 (the proof requires slight modification to account for up to two undersize blocks per level, but the result is unchanged), and the zig-zag potentials, with cost bounded by cn.

Summing over all levels i, we obtain that the initialization costs are bounded by

$$144c \cdot n + gp \sum_{i=1}^{vis} (2 + n/2^{2^{i+1}}) \cdot (2^{i+1}+1) + cn$$

$$\leq 4930 + 3gp \cdot n \sum_{i \geq 1} (2^{i+1}+1)/2^{2^{i+1}} + gp(2^{vis+3} + 2vis)$$

$$\leq 4930 + 3gp \cdot n + 9gp \cdot n$$

$$\leq 6000n.$$

We conclude the following.

THEOREM 2. *Consider a sequence of searches performed on an n-node splay tree, $n \geq 16$. Let item v be the current item being accessed and u the previous item accessed. Suppose that the distance from u to v is d. (For the first search, $d = n$.) Then the amortized cost of the access, in rotations, is bounded by* $32,000 \log d + \lceil \log\log(2n-1) \rceil [6,000,000 + 3,000,000\alpha(2n-1)]$, *where the cost of initializing the potential is at most* $6000n$.

4. Paying for the level jumps. In this section we show how to remove the additive term of $O([\log\log n]\alpha(n))$ for each access. This charge is associated with the increases in visibility on the part of the accessed item. Recall that an item on the inner skeleton of an i-superblock has visibility i and an item on an external path of the vis-block has visibility vis. For each increase in the visibility of the accessed item there are three distinct charges. We detail these charges for the rotation that causes the accessed item to attain visibility $i+1$, for $i \geq 1$:

(i) the charge vp (see section 2);

(ii) the charge, $a_3\alpha(2^{2^{i+2}}) + b_3$, for making the topmost node of the just traversed right access path lazy (see section 3.1). Recall that an i-superblock contains c' items; now we are setting $c' = \min\{n, 2^{2^{i+2}}\}$;

(iii) the charge of c for each h-block, $h \leq i+1$, which receives a new root; this is a charge of at most $2c$ for each unit increase in visibility (see Lemma 13).

These costs arise at most once at each increase in the visibility of the accessed item. They are charged to the new visibility of the accessed item. The total charge to visibility level $i+1$ is bounded by $cost_{i+1} = vp + a_3\alpha(2^{2^{i+2}}) + b_3 + 2c$.

The cost of the rotation in which the accessed item attains visibility 1 is charged to the access itself; this is a charge of at most $vp + 2c$ (for no lazy trees are created within a 1-block as accesses to the same 0-block are accesses to the same item, which therefore is to be found at the root of the splay tree).

The analysis focuses on an access to the right of the splay tree root; an access to the left of the root is treated entirely analogously.

A few definitions will be helpful. Let B be the i-superblock being accessed currently. Suppose the current access is to the same i-block as the preceding access. The right i-header, i_h_r, is defined to be the topmost node on the right access path for B (see section 3.1 for the definition of the right access path). The *right i-leader* is defined to be the lowest ancestor of the right i-header which is on the right extreme path of the splay tree. We say the right i-header is *available* if it is on the left path (the *left i-path*) descending from the right i-leader. The number of proper right ancestors of i_h_r, called its *right depth*, is denoted i_d_r.

i_h_r has potential $c_5(cost_{i+1} \cdot \min\{i\_d_r, d_5 i \cdot cost_{i+1}\})$, where c_5 and d_5 are constants to be specified.

If the right i-header is not traversed there are no charges to visibility level $i+1$.

Consider an access which traverses the right i-header. The intuition is that the right i-header reduces its right depth and this either reduces its potential, or if it is too deep, i-spares (to be defined) will be available; in either case, these pay for the charge $cost_{i+1}$. Specifically, if the right i-header is available, but is neither the topmost nor the second highest node on the left i-path, then on traversal its right depth decreases. If it is at right depth at most $d_5 i \cdot cost_{i+1}$, then the decrease in its potential provides the required $cost_{i+1}$ charge. If it is at greater depth, then $s'/2$ of the spares associated with the nodes on the left i-path at right depths $(d_5(i-1) \cdot cost_{i+1}, d_5 i \cdot cost_{i+1}]$ cover the $cost_{i+1}$ charge (note that these nodes are traversed). While the accesses are to the same i-block, whether or not the right i-header is traversed, its right depth does not increase.

Next, we handle the case that the right i-header is at right depth 0 or 1. But here charge (ii) is reduced: it becomes $a_2 + b_2$ (see section 3.3.3). This is charged to the access operation itself, as is charge (i). This is a total charge of at most $a_2 + b_2 + 2vp$ (there may be two charges vp: one for a segment ending at the deepest node at right depth 1 and one for a segment ending at the deepest node at right depth 0). In this case we do not provide potential c to a node displaced from the root of its h-block, for $h \leq i$. Instead, we eventually provide the unlabeled portions of the extreme paths of such blocks with new path potentials. Recall that the old root of the h-block was unlabeled because only a top portion of the chain adjacent to the root was traversed (see section 2). As we will see, for each such block there are two possibilities.

To understand the possibilities, consider the right access path up to the right i-header. Consider an h-block B, $h \leq i$, whose left extreme path is partially on the right access path. If a chain of block B has a node on the right access path, then the bottommost node of this chain is on the right access path (for its left child, if any, on the right access path is the root of another h-block). Thus at any point, for each $i \leq vis$, for each $h \leq i$, there is at most one chain belonging to an h-block, for which nodes remain on the right access path and for which paths of unlabeled nodes have been created. It remains to explain how to pay for these unlabeled paths. This happens in one of two ways.

(i) The chain becomes a path (i.e., all its nodes are contiguous in the splay tree). The above process may have created up to two paths of unlabeled nodes without associated path potentials (note we intend paths and not just chains); one path was created while the root of the block had right depth 1, and the other was created while the root had right depth 0. These two paths are provided with path potentials by using the reserve path potential associated with the chain.

(ii) An access is made to a new i-block. Besides the h-block which was being accessed previously, there are at most $2(i-h+1)$ h-blocks, $h \leq i$, which have extreme paths with unlabeled nodes without associated path or chain potentials. The additional h-blocks are those whose roots are the j-leaders, $h \leq j \leq i$, if these j-leaders are the highest or second highest nodes on their j-paths. Again, each such block has at most two unlabeled paths. These paths are provided with path potentials and the cost is charged to the new access. This is a charge of at most $\sum_{h=1}^{i} 8c(i-h+1)\lfloor \log c_h \rfloor \leq 8c \sum_{h=1}^{i} (i-h+1)2^{h+1} \leq 64c \cdot 2^i$.

Now, suppose that the right i-header is not available. Then it need not be the case that its right height will be reduced when next traversed. However, the right i-header then becomes available. Note that once available, the right i-header can cease to be available only if it is traversed, at least during accesses to the same i-block. To cover the cost of the traversal of an unavailable right i-header, we provide it with an additional $cost_{i+1}$ potential. Outside of the initialization cost, this is obtained by doubling the charge for the traversals of the right i-header when available; i.e., in this case a charge of $2cost_{i+1}$ needs to be covered. Thus it suffices to have

$$c_5 \geq 2, \tag{42}$$

$$d_5 s'/2 \geq 2. \tag{43}$$

When an access to a new i-block is made, following the access, the potentials for the right and left h-headers are reinitialized, for $h \leq i$, at a cost of

$$2 \sum_{h=1}^{i} [c_5 d_5 h cost_{h+1}^2 + cost_{h+1}] \leq 2i(i+1)c_5 d_5 cost_{i+1}^2,$$

which is bounded by

$$2(i+1)^2 c_5 d_5 (a_3 \alpha(2^{2^{i+2}}) + b_3 + vp + 2c)^2.$$

In addition, in an access to a new i-block, the charge for the portion of the access in which the accessed item has not yet reached the inner $(i+1)$-skeleton totals

$$\sum_{1 \leq h \leq i} [(4e'+9)gp \cdot (2^{h+1}+1) + vp + 2c \cdot h]$$

(see (10)). The costs of the remainder of the access are covered by the analysis of this section; they total

$$a_2 + b_2 + 2vp + 64c \cdot 2^i.$$

Finally, we note that each node on the access path spends at most 3 sets of $s'/2$ spares (one set is used for the pseudotraversals of section 3.1, one set for traversing an extreme path of a lazy tree, and one set is used for the analysis of this section). This is a total of $s/2$ spares per traversed node; these spares are at hand for s spares are provided per couple.

On taking equality in (42), (43), we obtain $c_5 = 2$, $d_5 = 12$. On summing the charges for this section, we obtain a charge of $48[(i+1)\alpha(2^{2^{i+2}})]^2(6 \cdot 10^6)^2 + 10,000 \cdot 2^{i+1} + 4,300i + 34i^2 + 3 \cdot 10^6$ for an access to a new i-block which is an access to

the same i-superblock. In addition to the initialization costs present previously, we also have to initialize the right and left h-headers, for $h \geq 1$; this costs less than $10^{16}(\lceil \log\log n \rceil \alpha(n))^2$. Using the argument of section 3.7, we obtain the following.

THEOREM 3. *Consider a sequence of searches performed on an n-node splay tree, $n \geq 16$. Let item v be the current item being accessed and u the previous item accessed. Suppose that the distance from u to v is d. (For the first search, $d = n$.) Then the amortized cost of the access, in rotations, is bounded by $42,000 \log d + 10^{16}(\lceil \log\log d \rceil \alpha(d^2))^2$, where the cost of initializing the potential is at most $6000n + 10^{16}(\lceil \log\log n \rceil \alpha(n))^2$.*

5. Insertions and deletions. In this section, the result is extended to incorporate insertions and deletions. The main idea is to allow blocks to have varying sizes and to combine or partition them as they become too small or too large. The previous bounds on block sizes will become lower bounds: An i-superblock will contain at least $b'_{i+1} = 2^{2^{i+1}}$ i-blocks and at most $b_{i+1} = 8b'_{i+1}$ i-blocks. Now, a 1-block will contain between 16 and $8 \cdot 16$ items. There will be no undersize blocks, except possibly at the topmost level, where there is just one vis-block, which contains all the $(vis-1)$-blocks. In fact, with one exception dealt with later, an i-superblock will contain at most $4b'_{i+1}$ i-blocks.

We use the following rule: when an i-superblock reaches its usual upper boundary size ($4b'_{i+1}$ i-blocks), it is partitioned into two i-superblocks, one containing the leftmost $2b'_{i+1}$ i-blocks and the other the rightmost $2b'_{i+1}$ i-blocks. When an i-superblock reaches its lower bound size, it is combined with a neighboring i-superblock in the same $(i+1)$-superblock; this new i-superblock is partitioned in turn if it has size at least $4b'_{i+1}$ i-blocks. If the vis-block is partitioned, then we need to create a new higher level block, a $(vis+1)$-block, to contain the two vis-blocks at hand. Likewise, if there are only 2 $(vis-1)$-blocks and they are combined, then the vis-block is removed and the top level block is at level $vis-1$.

Comment. We use the terms combine and partition rather than the more natural join and split to avoid confusion with the split of a lazy tree.

We determine the cost of the combine and partition operations on an i-superblock. When performing a partition we have the following costs. Each global node on the two extreme paths between the new superblocks has its rank raised to $\lceil \log 2K \rceil$, where $K = 2^{\lceil \log b_{i+1} \rceil}$. This costs up to $4 \cdot 2^{2^{i+1}} \cdot (2^{i+1}+3) \cdot gp$. In addition, a path potential has to be provided for up to four new chains: these are the chains starting and ending at the node that has newly become the root of an i-superblock, chains which are also parts of the extreme paths of i-superblocks. An i-superblock contains at most $4^{i+1} \cdot 2^{2^{i+2}}$ items. So a chain can contain at most one fewer items, and hence the cost of the new chains is at most $24c[2(i+1) + 2^{i+2}] + 4c$ (see section 2). In addition, on the new chains, the i-global nodes may need to be provided with an extra zig-zag potential of c units each, if they are not adjacent to two chain nodes. In fact this applies only to the global nodes newly on an extreme path of an i-superblock, which is at most $4b'_{i+1}$ nodes. So this is a further cost of $4c \cdot 2^{2^{i+1}}$. Finally, the new root of an i-superblock receives its $(i+1)$-global potential, which costs at most $(2^{i+1}+3)gp$. This in turn may raise the global potential of some of its ancestors for a further cost of $(2^{i+1}+3)gp$. The total cost of the partition is

$$part = 2^{2^{i+1}}[4gp(2^{i+1}+3) + 4c] + 2^{i+1} \cdot (48c + 2gp) + i \cdot 48c + (52c + 6gp).$$

The combine has zero cost.

In addition, when a combine or partition is performed, we treat it as ending the current sequence of accesses to the superblock being combined or partitioned. This entails additional costs of $48[(i+2)\alpha(2^{2^{i+3}})]^2(6\cdot 10^6)^2 + 10,000\cdot 2^{i+2} + 4,300(i+1) + 34(i+1)^2 + 3\cdot 10^6$ (see section 4).

In order to cover these costs, each time an i-block is combined or partitioned, it provides the following potential to its parent (the i-superblock in which it is contained): $p_{i+1} = 4gp(2^{i+1}+3) + 4c + [2^{i+1}\cdot(48c+2gp) + i\cdot 48c + (52c+6gp) + 48[(i+2)\alpha(2^{2^{i+3}})]^2(6\cdot 10^6)^2 + 10,000\cdot 2^{i+2} + 4,300(i+1) + 34(i+1)^2 + 3\cdot 10^6 + p_{i+2}]/2^{2^{i+1}}$. It can be checked that $p_{i+1} = 152\cdot 2^{i+1} + 10^{16}$ suffices. The accumulation of this potential by an i-superblock will ensure that when and if it is removed by a combine or partition it will have enough potential at hand to pay for its removal and for the potential that has to be provided to its parent block; for between the creation and removal of an i-superblock there will have been at least b'_{i+1} combines and/or partitions of its i-blocks. At level 0, when an item is inserted or deleted, it needs to provide potential p_1 to its 1-block, but this is $O(1)$.

Next, we need to modify the analysis. For the moment, we ignore the question of accesses that straddle a block boundary. A search is treated as before. An insertion is analyzed as follows. The item is inserted as in a binary search tree. After adding the item, we perform any necessary block partitions. The item is splayed now. When the item is added, it is not provided with any potential, nor are the appropriate chain potentials increased. The reason is that the item does not need any associated potential in order to be splayed; the potential is needed at nodes which are traversed. The access is analyzed as before: as the access proceeds, the item will be provided with potential as described in the earlier analysis. So the cost of an insertion at distance d from the previous access is p_1 plus the cost of a search at distance d.

For the deletion, at least for our analysis, we need to modify the operation described by Sleator and Tarjan [ST85]. We offer reasons as to why such a modification may be needed in order to obtain the finger search result we seek. Sleator and Tarjan proceed as follows. First the item e is splayed, placing it at the root. Then the resulting two subtrees of e are joined (for example, by splaying the rightmost item in the left tree, and then making the right tree its child). Our method differs. First we swap the item, e, to be deleted with its predecessor, if e has a left child. Then the current predecessor of e (previously, the second predecessor, if e had a left child at the start of the operation) is splayed. Following the splay, e has no children, and it is removed. The reason for our choice of deletion method is that the change in potential owing to the removal of item e might be nonconstant if it is the root of a block.

There is an unpleasing assymmetry to the operation described above. An alternate implementation is to perform two splays, in turn on the predecessor and successor of the item, e, to be deleted; then e is deleted—it is the right child of the left child of the root of the splay tree and has no children at this point. The accesses are analyzed as before and the deletion can only reduce the potentials associated with the various blocks. The unattractive feature here is that two searches and two splays are needed.

Now, we analyze the first implementation of the delete operation. The access is analyzed as before. The only novel feature is the elimination of an item, which may add nodes to the extreme path of the blocks to which the item belongs and thereby increase the potential of the splay tree; but this effect can arise only if the item in question is an extreme item in its block. In fact, at the time item e is deleted it is childless, so no nodes are added to the extreme paths of any blocks; indeed, the removal of e can only shorten these paths, thereby reducing the potential. So the cost

of a deletion is at most c_1 plus the cost of an access at distance $d+2$.

It remains to deal with nearby accesses that straddle a block boundary. First we explain how to form sequences of accesses. Consider the i-blocks. The accesses are partitioned into maximal subsequences S_i of accesses to pairs of neighboring i-blocks. Whenever an i-block is partitioned or combined a new sequence is started.

It may be that a particular sequence S_i accesses only one i-block. But suppose that S_i accesses two neighboring i-blocks, B_1' and B_2'. If they are contained in distinct i-superblocks, B_1 and B_2, respectively, then at the start of the sequence S_i, B_1 and B_2 are combined into i-superblock B; at the end of the sequence B is split into B_1 and B_2 with the new boundary being between B_1' and B_2', as before. B_1' and B_2' still exist at this point in time for no partition or combine of these blocks has been caused by their becoming too large or small during sequence S_i, although they may have been combined and partitioned because of accesses straddling neighboring $(i-1)$-blocks. As before the cost of the combine is zero. The cost of the split is modest, because the most recent accesses to the combined B_1 and B_2 are to adjacent i-blocks B_1' and B_2'. We need only provide the following potentials: chain potential for up to three new chains, which costs at most $18c[2(i+1)+2^{i+2}]+3c$, and the $(i+1)$-global potential for the new root of an i-superblock, which costs at most $(2^{i+1}+3)\cdot gp$. There are no i-global nodes whose potential needs increasing.

The subsequence S_i is further partitioned, so that all the accesses in each new subsequence are to the same block. Suppose the i-blocks B_1' and B_2' are not combined for some pair of accesses of S_i which straddle the boundary between i-superblocks B_1 and B_2; this implies these two accesses are to nonneighboring i-subblocks. So an access to a nonneighboring $(i-1)$-block, i.e., an access at distance at least 2^{2^i}, may start a new i-sequence of accesses to an i-block. Let d denote the distance spanned by this access; then $d+2 \geq 2^{2^i}$ (the $d+2$ takes account of the possible additional shift by 2 due to a delete operation). Following the analysis of section 4, the cost of this access is bounded by

$$\sum_{1\leq h\leq i}[(4e'+9)gp\cdot(2^{h+1}+4)+vp+2c\cdot h]$$
$$+\sum_{1\leq h\leq i}2(i+1)^2c_5d_5(a_3\alpha(2^{2^{i+2}})+b_3+vp+2c)^2+p_1+a_2+b_2+2vp+64c2^i.$$

The term $(2^{h+1}+4)$ reflects the increased maximum rank on the boundary of an h-block. The cost is bounded by $new\_cost_i = 20,000\cdot 2^i+10^{16}[(i+1)\alpha(2^{2^{i+2}})]^2+10^{16}$. In turn this is bounded by $20,000\log(d+2)+10^{16}[(\log\log(d+2)+1)\alpha(2^{4\log(d+2)})]^2+10^{16}$.

The last step in the analysis is to account for a series of accesses which each shift a small distance, but cummulatively shift a large distance and cause a high index block boundary to be crossed. Clearly a simple amortization can handle this; we specify this precisely next. Suppose the current access is at distance d from the previous access, where $2^{2^i}\leq d+2<2^{2^{i+1}}$. Then potential $2new\text{-}cost_i\cdot 5^{2-h}$ is provided to the $(i+h)$-block to which the previously accessed item belongs, for $h\geq 0$. Consider a maximal sequence S_j of accesses to a j-block, B_j, and the access immediately following it. The distance spanned by these accesses is at least $2^{2^j}-2$, for otherwise the accesses would all be in adjacent $(j-1)$-blocks and hence in the same j-block (because of our combining rule). We determine the contribution made by these accesses to B_j and show that it is at least $new\text{-}cost_j$. The contribution by an access of length d, where $2^{2^{j-h}}\leq d+2<2^{2^{j-h+1}}$ is $2new\text{-}cost_{j-h}\cdot 5^{2-h}\geq 2new\text{-}cost_j\cdot 5^{2-2h}$. The total

contribution is minimized when each d takes on the maximal value consistent with a given contribution; this yields a total contribution of the form $\sum_l 2new\text{-}cost_j \cdot 5^{2-2h_l}$, where $\sum_l [2^{2^{j-h_l+1}} - 3] \geq 2^{2^j} - 2$; by inspection, the contribution is minimized with $h_l = 2$, when it totals at least $new\text{-}cost_j$.

The additional charge for each operation is $\frac{25}{2} new\text{-}cost_i$, where $2^{2^i} \leq d+2 < 2^{2^{i+1}}$.

We have shown the following.

THEOREM 4. *The extension of the dynamic finger conjecture to incorporate the insert and delete operations is true when the delete is modified as described above. The amortized cost of an access at distance d from the previous access is bounded by* $250,000 \log(d+2) + 10^{18}[(\log\log(d+2)+1)\alpha(2^{4\log(d+2)})]^2 + 10^{18}$.

6. Improved constants. Our conjecture is that the true bound has much smaller constants than those shown here. We briefly sketch one approach for obtaining such a bound.

The main difficulty in improving the bounds lies in the potentials needed for the lazy trees. If it were possible to have $\log n$ levels of visibility, where each superblock contained just a constant number of blocks, we would avoid the need for global potentials and hence for lazy trees. There would still be a need for potentials associated with extreme paths (small and large debits and chain potentials), but changes in these would all be paid for by increases in visibility. This would now be of size $\Theta(\log n)$, rather than $\Theta(\log\log n)$, so there would be a continued need to pay for level jumps (see section 4). Unfortunately, following the analysis used there would lead to an $O(\log^2 d)$ cost for an access at distance d from the previous access. It appears possible to improve this by not providing the chain potentials in full and instead taking a *lazy* approach. In particular, we would provide only $O(1)$ potential for each individual jump in visibility on the access path, however large. This eliminates the need for potentials associated with right i-headers; offsetting this, we now have to handle "missing" chain potentials. We have not worked out all the details, however, for it appears that we still need an inverse Ackerman term, arising for similar reasons to the inverse Ackerman term used in analyzing paths P_3 (see section 3.3.4). At present we do not see how to amortize this term, and it leads at the very least to an additive inverse Ackerman term on the amortized cost of each access. In fact, we suspect that this bound can be reduced to $O(1)$. Perhaps this would follow from improving Sundar's bound for the dequeue conjecture [Su89], or some variant of this conjecture.

Acknowledgments. We thank an anonymous referee for making a number of detailed suggestions that helped improve the paper's presentation. Any remaining errors or lack of clarity are solely due to the author.

REFERENCES

[CMSS00] R. COLE, B. MISHRA, J. SCHMIDT, AND A. SIEGEL, *On the dynamic finger conjecture for splay trees. Part* I: *Splay sorting* $\log n$*-block sequences*, SIAM J. Comput., 30 (2000), pp. 1–43.

[Luc88a] J.M. LUCAS, *Arbitrary Splitting in Splay Trees*, Technical Report DCS-TR-234, Computer Science Department, Rutgers University, Piscataway, NJ, 1988.

[Luc88b] J.M. LUCAS, *Canonical Forms for Competitive Binary Search Tree Algorithms*, Technical Report DCS-TR-250, Computer Science Department, Rutgers University, Piscataway, NJ, 1988.

[ST85] D.D. SLEATOR AND R.E. TARJAN, *Self-adjusting binary search trees*, J. ACM, 3 (1985), pp. 652–686.

[STT86] D.D. SLEATOR, R.E. TARJAN, AND W.P. THURSTON, *Rotation distance, triangulations, and hyperbolic geometry*, J. Amer. Math. Soc., 1 (1988), pp. 647–681.

[Su89] R. Sundar, *Twists, turns, cascades, deque conjecture, and scanning theorem*, in Proceedings of the 13th Symposium on Foundations of Computer Science, Research Triangle Park, NC, 1989, pp. 555–559.

[T85] R.E. Tarjan, *Sequential access in splay trees takes linear time*, Combinatorica, 5 (1985), pp. 367–378.

[W86] R. Wilber, *Lower bounds for accessing binary search trees with rotations*, SIAM J. Comput., 18 (1989), pp. 56–67.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 86–109

ON RAM PRIORITY QUEUES*

MIKKEL THORUP†

Abstract. Priority queues are some of the most fundamental data structures. For example, they are used directly for task scheduling in operating systems. Moreover, they are essential to greedy algorithms. We study the complexity of integer priority queue operations on a RAM with arbitrary word size, modeling the possibilities in standard imperative programming languages such as C. We present exponential improvements over previous bounds, and we show tight relations to sorting.

Our first result is a RAM priority queue supporting `find-min` in constant time and `insert` and `delete-min` in time $O(\log\log n)$, where n is the current number of keys in the queue. This is an exponential improvement over the $O(\sqrt{\log n})$ bound of Fredman and Willard [*Proceedings of the 22nd ACM Symposium on the Theory of Computing*, Baltimore, MD, pp. 1–7]. Plugging this priority queue into Dijkstra's algorithm gives an $O(m \log\log m)$ algorithm for the single source shortest path problem on a graph with m edges, as compared with the previous $O(m\sqrt{\log m})$ bound based on Fredman and Willard's priority queue. The above bounds assume $O(n2^{\varepsilon w})$ space, where w is the word length and $\varepsilon > 0$. They can, however, be achieved in linear space using randomized hashing.

Our second result is a general equivalence between sorting and priority queues. A priority queue is *monotone* if the minimum is nondecreasing over time, as in many greedy algorithms. We show that on a RAM, the amortized operation cost of a monotone priority queue is equivalent to the per-key cost of sorting. For example, the equivalence implies that the single source shortest paths problem on a graph with m edges is no harder than that of sorting m keys. With the current RAM sorting, this gives an $O(m \log\log m)$ time bound, as above, but the relation holds regardless of the future developments in RAM sorting.

From the equivalence result, for any fixed $\varepsilon > 0$, we derive a randomized monotone $O(\sqrt{\log n}^{1+\varepsilon})$ priority queue with expected constant time `decrease-key`. Plugging this into Dijkstra's algorithm gives an $O(n\sqrt{\log n}^{1+\varepsilon} + m)$ algorithm for the single source shortest path problem on a graph with n nodes and m edges, complementing the above $O(m \log\log m)$ algorithm if $m \gg n$. This improves the $O(n \log n/\log\log n + m)$ bound by Fredman and Willard [*Proceedings of the 31st IEEE Symposium on the Foundations of Computer Science*, St. Louis, MO, 1990, pp. 719–725], based on their $O(\log n/\log\log n)$ priority queue with constant `decrease-key`.

Key words. priority queues, sorting, greedy algorithms, shortest paths, RAM model

AMS subject classifications. 68P05, 68P10, 68Q05, 68Q25, 68R10

PII. S0097539795288246

1. Introduction.

Priority queues are some of the most fundamental data structures. For example, they are used directly for task scheduling in operating systems. Moreover, they are essential to greedy algorithms. A *priority queue* is a dynamic representation of an ordered set, or multiset, X of keys. A *basic priority queue* supports the operations `find-min`(X) returning $\min X$, `insert`(x, X) setting $X := X \cup \{x\}$, and `delete-min`(X) setting $X := X \setminus \{\min X\}$. Often we will talk about *extracting the minimum key*, which is the combination of finding and deleting the minimum key from X. Using the previous operations, extracting the minimum key can be implemented as $x :=$ `find-min`(X); `delete-min`(X); return x. Sometimes we will go beyond basic priority queues and consider the operations `delete`(x, X) setting

*Received by the editors June 26, 1995; accepted for publication (in revised form) September 13, 1999; published electronically April 25, 2000. A short preliminary version of this paper was presented at the 7th ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996. Some of this work was completed while the author was at the University of Copenhagen.

http://www.siam.org/journals/sicomp/30-1/28824.html

†AT&T Research Labs, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932 (mthorup@research.att.com).

$X := X \setminus \{x\}$, and `decrease-key`(x, d, X) setting $X := (X \setminus \{x\}) \cup \{x - d\}$. Finally, we will study *monotone* priority queues, which are priority queues with the restriction that the minimum should be nondecreasing. Here the minimum of an empty priority queue is considered to be 0. The monotonicity restriction on priority queues has been considered previously by Ahuja et al. [1] and is not a problem for applications in greedy algorithms such as Dijkstra's single source shortest paths algorithm [13]. Our priority queues will allow the user to associate extra information with inserted keys. Then `find-min` returns not only the minimal key but also the information associated with it.

We study the complexity of priority queue operations on a RAM modeling the possibilities in imperative programming languages such as C. The memory is divided into addressable words of w bits. Addresses are themselves contained in words. Hence, assuming that we have at least n input words to address, $w \geq \log n$. Moreover, we have a constant number of registers, each with capacity for one word. The basic machine instructions are conditional jumps and direct and indirect addressing for loading and storing words in registers. Moreover, we have some computational instructions for manipulating words in registers. The computational instructions used in this paper are shifts, comparisons, bit-wise Boolean operations, conversion of integers to floating points, addition, subtraction, multiplication, and division. Except for multiplication and division, all the above operations are in AC^0 meaning that they can be implemented by a constant depth circuit of size polynomial in w. We refer to them as *standard AC^0 operations* to emphasize the fact that they are available in most imperative programming languages. The time complexity is the number of instructions performed and the space complexity is the maximal memory address used. All keys are assumed to be nonnegative integers, each stored in one word, and hence we can operate on them in constant time. We shall return to a discussion of the model and the "dirty tricks" it allows in section 1.3.

On the above RAM, we present exponential improvements over previous bounds for priority queues. Moreover, we establish an equivalence between monotone priority queues and sorting.

1.1. A fast basic priority queue. Our first main result is the following theorem.

THEOREM 1.1. *There is a priority queue supporting* `find-min` *in constant time and* `insert` *and* `delete-min` *in $O(\log\log n)$ time, where n is the current number of keys in the queue.*

This matches the best known per-key cost of integer RAM sorting due to Andersson et al. [3]. Here, by the per-key cost of sorting we mean the total sorting time divided by the number of sorted keys.

1.1.1. A historical perspective. In the comparison based model, we can support a general search structure in $\Theta(\log n)$ time per operation, matching the per-key cost of sorting. Here by a *search structure*, we mean a dynamic ordered set with `insert` and `delete`, and an operation `succ`(x, X) returning y, where y is the smallest element in X which is bigger than or equal to x. This generalizes a priority queue since `find-min`$(X) =$ `succ`(0) and `delete-min`$(X) =$ `delete`(`succ`(0)). Then again, priority queues generalize sorting in that we can sort a set X by first inserting all elements from X in an empty priority queue and then extract them in increasing order.

When Fredman and Willard in 1990 broke the $\Omega(n \log n)$ barrier for RAM integer sorting [17], they did it in terms of a general search structure working in $O(\sqrt{\log n})$

time per operation, implying $O(n\sqrt{\log n})$ sorting as a special case.

Then in 1995, Andersson et al. presented an $O(n \log\log n)$ integer RAM sorting algorithm [3]. As they point out in [3], their sorting algorithm does not work for dynamic sets. Nevertheless, as stated in Theorem 1.1, here we will generalize their $O(\log\log n)$ bound to priority queues for dynamic sets.

It turns out that our $O(\log\log n)$ bound provably does not generalize to searching. Miltersen has pointed out that for $n = 2^{\log^{3/2} w}$, his results in [28] imply an $\Omega(\log^{1/3-o(1)} n)$ lower bound for search structures. This lower bound holds even if we allow randomization and amortization, and it has later been improved to $\Omega(\sqrt{\log n/\log\log n})$ by Beame and Fich [8]. In [28], Miltersen asks if the lower bounds for searching can be beaten if we restrict ourselves to a basic priority queue. Our new $O(\log\log n)$ upper bound for all word sizes answers Miltersen's question in the affirmative. In particular, this implies a separation between priority queues and searching on the RAM.

1.1.2. Space bounds. The priority queue of Theorem 1.1 is simple and implementable with standard AC^0 operations, but it needs $O(n2^{w^\varepsilon})$ space for an arbitrarily small constant $\varepsilon > 0$. If we are satisfied with a monotone priority queue, we can get an *amortized* time bound of $O(\log\log n)$ using $O(n + 2^{w^\varepsilon})$ space. We can also get the whole way down to linear space even without amortization if we are willing to use randomization giving $O(\log\log n)$ *expected* time per operation. Thereby we also involve multiplication which is not in AC^0. The above mentioned sorting and searching results from [3, 17] have similar space bounds. However, in [17] they also achieve a deterministic $O(\log n/\log\log n)$ worst-case bound in linear space.

1.1.3. Applications in greedy algorithms. Plugging the priority queue of Theorem 1.1 into Dijkstra's algorithm [13], we get the following.

COROLLARY 1.2. *There is an $O(m \log\log m)$ algorithm for the single source shortest path problem on a graph with m edges.*

Dijkstra's algorithm above should only be taken as a representative for greedy algorithms. We could equally well apply our priority queue to Hu and Tucker's algorithm from [23] and get an $O(n \log\log n)$ algorithm for the alphabetic tree problem on n codes. Based on Fredman and Willard's priority queue from [18], the previous best bounds for the two problems were $O(m\sqrt{\log m})$ and $O(n\sqrt{\log n})$, respectively. The advantage of focusing on Dijkstra's algorithm is that it facilitates later discussions of constant time `decrease-key` operations.

1.1.4. Techniques. In order to prove Theorem 1.1, we first prove a result for short keys.

THEOREM 1.3. *There is a priority queue with capacity for n $(w/(\log n \log\log n))$-bit keys, supporting* `insert`, `find-min`, *and* `delete-min` *on integers in constant time per operation and in space $O(n)$. The priority queue can be implemented with standard AC^0 operations. Moreover, it allows one word of information to be associated with each key inserted. Then* `find-min` *returns the smallest key and the information associated with it.*

The point of the associated word is that it could be an address to some additional information on the key, or it could be an identification of the key in case we had several keys with the same value. Note that associating one word of information is trivial for priority queues dealing with one-word keys, for then, by only doubling the key length, we can just append the information directly to the key as a least significant

part. Doubling the key length is not a problem because our RAM can simulate $2w$-bit instructions with a constant number of w-bit instructions.

Having proved Theorem 1.3, we reduce the general case of one-word keys to the case of $(w/(\log n \log\log n))$-bit keys using the recursion underlying van Emde Boas' data structure [35, 36].

1.2. Monotone priority queues ≡ sorting. Our other main result is a computational equivalence between sorting and monotone priority queues.

THEOREM 1.4. *For a RAM with arbitrary word size w, if we can sort n w-bit keys in time $n \cdot s(n)$, where s is nondecreasing, then (and only then) there is a monotone priority queue with capacity for n w-bit keys, supporting* `find-min` *in constant time and* `insert` *and* `delete-min` *in $s(n) + O(1)$ amortized time. The amortization presumes that the priority queue starts empty.*

The reduction is deterministic and uses linear space. As stated, it needs either multiplication or some nonstandard AC^0 operations. A variant using only standard AC^0 operations supports `find-min` *in constant time and* `insert` *and* `delete-min` *in $O(\sum_{i=0}^{\log^* n} s(\log^{(i)} n))$ amortized time. Here $\log n = \log n \log\log n$, $f^{(0)}n = n$, $f^{(i)}n = f(f^{(i-1)})$, and $f^*n = \min\{i | f^{(i)}n \le 1\}$.*

The "only then" part follows because we can sort a set of keys by first inserting them into a monotone priority queue and then extract them in increasing order.

To prove Theorem 1.4 we will show how to maintain a monotone priority queue spending constant time on `find-min` and constant amortized time on `insert` and `delete` plus some sorting time. Each key passing through the queue participates in at most one sorting, and when we sort a set of keys, they are all currently contained in the queue.

As an immediate consequence of Theorem 1.4, we get the following strengthening of Corollary 1.2.

COROLLARY 1.5. *On a RAM, the single source shortest path problem on a graph with m edges is no harder than that of sorting m keys, no matter the future developments in sorting.*

Note that Theorem 1.4 potentially could be used to obtain lower bounds for sorting. As mentioned above, [28] implies amortized nonconstant lower bounds on search operations. Similar nonconstant lower bounds on monotone priority queue operations would imply a nonlinear lower bound for RAM sorting.

1.2.1. Applications to constant cost `decrease-key`. We will now show how Theorem 1.4 can be used to construct priority queues with a constant cost `decrease-key` operation. First, Andersson et al. [3] have shown that we can sort n keys in linear expected time if $(\log n)^{2+\varepsilon} \le w$ for some fixed $\varepsilon > 0$. Thus by Theorem 1.4, we have the following.

COROLLARY 1.6. *If $(\log n)^{2+\varepsilon} \le w$ for some fixed $\varepsilon > 0$, there is a randomized linear space monotone priority queue for n keys supporting* `find-min` *in constant time and* `insert` *and* `delete-min` *in expected constant amortized time.*

Based on Fredman and Willard's AF-heaps [18], we will get the following lemma.

LEMMA 1.7. *Provided a monotone/unrestricted basic priority queues for up to $f(n) = \Omega(\log^2 n)$, $f(n) = O(n)$, keys supporting* `find-min`, `insert`, *and* `delete-min` *in constant amortized time, there is a monotone/unrestricted priority queue with capacity n, supporting the operations* `find-min`, `insert`, *and* `decrease-key` *in constant amortized time, and* `delete` *in time $O(\log n / \log f(n))$. The reduction is deterministic and uses linear space, and it can be implemented with standard AC^0 operations.*

In [18], Fredman and Willard show that given $O(n)$ space and preprocessing time, they can support unrestricted priority queues with constant amortized operation cost for up to $(\log n)^2$ keys. By Lemma 1.7, this gives them an unrestricted priority queue with capacity n, supporting `find-min`, `insert`, and `decrease-key` in constant amortized time, and supporting `delete` in $O(\log n/\log(\log n)^2) = O(\log n/\log\log n)$ amortized time.

Corollary 1.6 together with Theorem 1.4 implies a monotone priority queue with constant amortized operation cost for up to $\exp(\sqrt{w}^{1-\varepsilon})$ keys for any fixed $\varepsilon > 0$. Thus we get `delete` in time $O(\log n/\log(\exp(\sqrt{w}^{1-\varepsilon}))) = O(\log n/\sqrt{w}^{1-\varepsilon})$. Using $w \geq \log n$, we conclude the following.

COROLLARY 1.8. *There is a randomized monotone priority queue with capacity n supporting* `find-min`, `insert`, *and* `decrease-key` *in expected amortized constant time, and* `delete` *in expected amortized time $O(\log n/\sqrt{w}^{1-\varepsilon}) = O(\sqrt{\log n}^{1+\varepsilon})$ for any fixed $\varepsilon > 0$.*

Plugging Corollary 1.8 into Dijkstra's algorithm [13], we get the following corollary.

COROLLARY 1.9. *For any $\varepsilon > 0$, randomized and in linear space, we can solve the single source shortest path problem on a graph with n nodes and m edges in expected time $O(n\sqrt{\log n}^{1+\varepsilon} + m)$.*

The previous best bound linear in m was $O(n \log n/\log\log n + m)$, based on the priority queues from [18]. For completeness, Corollary 1.9 should also be compared with the word size dependent bound of $O(n\sqrt{w} + m)$ due to Ahuja et al. [1]. In fact, the bound from [1] is based on priority queues restricted beyond monotonicity, tailored particularly for Dijkstra's algorithm. Since $w \geq \log n$, this bound is only better for $w = (\log n)^{1+o(1)}$. Recall that for small m, we have the complementing $O(n + m \log\log m)$ bound from Theorem 1.1.

1.3. Machine model. The RAM model distinguishes itself from the so-called comparison based model in that besides comparing the integer keys, we can also add, subtract, and multiply them. Of course, it is meaningless to work on integers and say that one can compare but not add them. The point in the comparison based model is that it works for any domain supporting comparisons. In this context it should be noted that ordering nonnegative integers actually covers many more domains than just nonnegative integers. For example, in the standard representation of signed integers, we get the right ordering if we flip the sign bit and interpret the result as an unsigned integer. Floating points are even easier, for the IEEE 754 floating-point standard is designed so that the ordering of floating-point numbers can be deduced by perceiving their representations as multiple word integers. Also, if we are working with fractions where both numerator and denominator are w-bit integers, we get the right ordering if for each fraction, we make the division in floating-point numbers with $2w$ bits of precision. Now we get the correct ordering of the original integer fractions by perceiving the corresponding floating-point numbers as integers. Finally, it will be shown later that we can implement an efficient priority queue for ℓ-word keys using the priority queue for one-word keys from Theorem 1.1.

The essential benefit of adding and subtracting integers is in coding multiple comparisons. The idea of multiple comparisons was first introduced by Paul and Simon [29] in 1980. In our paper, the multiple comparisons are hidden in a black box provided by Albers and Hagerup [2]. It should be noted that this use of uniprocessors as vector processors is a standard trick in practice, not in connection with sorting,

but in connection with graphics, where a single word operation is used to manipulate the information on several pixels, each represented by one byte of the word. With modern MMX graphics processors, these kinds of tricks are even supported directly in hardware, for example, with a special byte-wise multiplication that views words as vectors of bytes and then multiplies the bytes coordinate-wise. This byte-wise multiplication is faster than a multiplication over the full word.

The RAM model further distinguishes itself both from the comparison based model and from the pointer machine in that we allow ourselves to use keys and segments of keys as addresses. We will use this for bucketing in the same way as done in van Emde Boas's data structure [35, 36]. The use of buckets has a very long tradition within sorting. In radix sort, for example, the integers are bucketed according to one digit at the time, starting from the least significant digit. According to Knuth [25], radix sort was mentioned as early as in 1929 by Comrie. Another classic use of buckets in sorting is bucket sort proposed in 1956 by Isaac and Singleton [21]. Bucket sort is interesting in this context because it provides an elementary expected linear time sorting algorithm for uniformly distributed w-bit unsigned integer keys, thoroughly surpassing the $\Omega(n \log n)$ comparison based lower bound for average case sorting. Bucket sort may even be implemented efficiently with standard AC^0 operations: let X be a set of n unsigned w-bit integer keys. Compute $\ell = \lceil \log_2 n \rceil$ and $\Delta = w - \ell$. Bucket each key x according to $\lfloor x \times 2^\ell / 2^w \rfloor = x \gg \Delta$, where $\gg$ denotes right shift. More precisely, we allocate an array B of $2^\ell = O(n)$ lists and put x in list $B[x \gg \Delta]$. Second, we use insertion sort to sort each list $B[i]$ independently. Finally, we concatenate the sorted lists $B[i]$ into one sorted list containing all the elements of X. As analyzed in [25, pp. 99–102], the above algorithm takes linear time for uniformly distributed keys.

Fredman and Willard [17] further use the RAM for advanced tabulation of complicated functions over small domains. Their tabulation is too complicated to be of practical relevance, but tabulation of functions is in itself commonly used to tune code. As a simple example, Bentley [10, pp. 83–84] suggests that an efficient method for computing the number of set bits in 32-bit integers is to have a preprocessing where we first tabulate the number of set bits in all the 256 different 8-bit integers. Now, given a 32-bit integer x, we view it as the concatenation of four 8-bit integers, and for each of these, we look up the number of set bits in our table. Finally we just add up these four numbers to get the number of set bits in x.

Summing up, in the above discussion, it has been argued that all the "dirty tricks" facilitated by the RAM are well established in the practice of writing fast code. Hence, if we disallow these tricks, we are not discussing the time complexity of running imperative programs on real world computers. For further discussion of the model, the reader is referred to [20].

1.4. Subsequent work and historical remarks. Since an extended abstract of this paper was presented at SODA'96 [32], the results and constructions have been involved in several other papers. In [32], Theorem 1.3 appeared in two weaker forms. In one form the keys were of size $w/\log n$ instead of $w/(\log n \log\log n)$, but then the time bounds were $O(\log\log n)$ instead of constant. The other form had constant but amortized time bounds. The key size in the amortized version was $w/(\log n \log\log n)$ as in the current paper. The amortized version used multiplication as it was based on Fredman and Willard's atomic heaps [18]. Later, Raman [30] has found a simple multiplication-free alternative to the atomic heaps inside the priority queue of Theorem 1.3 (cf. Lemma 3.4 below). Consequently, he got an amortized

TABLE 1.1

State of the art in priority queues. The capital letters following a bound show the weaknesses of the result: "M" denotes monotone, "A" denotes amortization, "R" denotes randomization, and "×" denotes use of mutiplication. Finally, "D" means that we use a dictionary over a large set of integers. Using hashing, "D" can be replaced by RM, but alternatively, we may use nonlinear space. If there is no letter following a bound, it means that it is unrestricted, deterministic, and worst-case, and it uses linear space and only standard AC^0 operations.

Priority queue	$O(\log\log n)$ D [this paper] • $O(\log\log n)$ MAR [33] • $O((\log\log n)^2)$ [7] • $O(\log w)$ D [35, 36]
Constant `decrease-key`	$O(\log n)$ A [19] • $O(\log n/\log\log n)$ A× [18] • $O(\sqrt{\log n\log\log n})$ MA [30] • $O(\sqrt[3]{w\log w})$ MA [12] • $O(\min\{\sqrt[4]{w}^{1+\varepsilon}, \sqrt[3]{\log n}^{1+\varepsilon}\})$ MAR× [31]

version of Theorem 1.3 using only standard AC^0 operations. Raman applied this within an $O(\sqrt{\log n\log\log n})$ amortized monotone priority queue with constant time `decrease-key` which uses only linear space and only standard AC^0 operations. In this paper, Theorem 1.3 reaches its final form with constant worst-case time per operation using only standard AC^0 operations. This strong form has not been announced previously.

In the original abstract [32] of this paper, Corollary 1.8 quoted the $O(\sqrt{\log n}^{1+\varepsilon})$ bound but not the $O(\log n/\sqrt{w}^{1-\varepsilon})$ bound. In [32], w was replaced by $\log n \le w$ already in the derivation leading to Corollary 1.8 as the author found it more esthetic just to quote a bound based on n exclusively.

Later, Cherkassky, Goldberg, and Silverstein [12] used the $O(\sqrt{\log n}^{1+\varepsilon})$ bound from Corollary 1.8 for a randomized $O(\sqrt[3]{w}^{1+\varepsilon})$ monotone priority queue with expected constant cost decrease key, improving over a previous $O(\sqrt{w})$ bound of Ahuja et al. [1].

However, Raman [31] has observed that for Cherkassky, Goldberg, and Silverstein's construction [12] it is beneficial not to replace w by $\log n \le w$ but to use the $O(\log n/\sqrt{w}^{1-\varepsilon})$ of Corollary 1.8 directly. In an elegant balancing, Raman arrived at an $O(\min\{\sqrt[4]{w}^{1+\varepsilon}, \sqrt[3]{\log n}^{1+\varepsilon}\})$ bound for monotone priority queues with constant cost decrease key, thus improving both over Cherkassky, Goldberg, and Silverstein's bound of $O(\sqrt[3]{w}^{1+\varepsilon})$ and over the $O(\sqrt{\log n}^{1+\varepsilon})$ bound of Corollary 1.8. For a nice survey of these developments, the reader is referred to Raman's paper [31].

The author [33] has used the reduction of Theorem 1.4 to derive a randomized monotone priority queue with an expected amortized operation cost of $O(\log\log n)$ which is implementable in linear space using only standard AC^0 operations. Further, the author [34] has provided an amortized deterministic unrestricted priority queue with operation time $O((\log\log n)^2)$ using linear space and standard AC^0 operations. The construction was later deamortized together with Andersson [7]. An overview of the current theoretically strongest results on priority queues is presented in Table 1.1.

1.5. Contents. After some preliminaries in section 2, the priority queue of Theorem 1.3 is presented in section 3. Next we derive the time bounds of Theorem 1.1 in section 4 and discuss the corresponding space bounds in section 5. Section 6 contains a brief discussion on priority queues for strings of words. The general reduction of Theorem 1.4 is presented in section 7, and finally, we discuss the reduction of Lemma 1.7 in section 8.

2. Preliminaries. Unless otherwise stated, all logarithms are assumed to be base 2. Our integers are viewed as bit-strings, stored with the most significant bit to the left. Hence, we get the right ordering if we view them as lexicographically ordered bit-strings. By bit i of an integer x, denoted $x[i]$, we mean the $(i+1)$th bit from the left. Thus, if x is a b-bit integer, the number represented by x is $\sum_{i=0}^{b-1} 2^i x[b-1-i]$.

We have defined standard AC^0 operations as AC^0 operations available in a standard programming language such as C. These include shifts, bit-wise Boolean operations, addition, subtraction, and conversion from integer to floating-point representation. We use $\ll$ and $\gg$ to denote left and right shifts. Typically, we use shifts to divide and multiply by powers of two, e.g., $\lfloor x/2^i \rfloor = x \gg i$. Also, we will use shifts to extract parts of a word x: for $i \leq j < w$, let $x[i..j]$ denote $x[i] \cdots x[j]$. Using shifts, we move $x[i..j]$ to the right of x, making all other bits 0 by $(x \ll i) \gg (i+w-1-j)$. Finally, we use shifts to flip, set, and unset bits in words. For example, we flip bit i in x by $x \oplus (1 \ll (w-1-i))$.

The reason for mentioning conversion to floating-point is that if we convert an integer x to floating-point and extract the exponent $\texttt{expo}(x)$, then $\texttt{expo}(x) = \lfloor \log x \rfloor$. Here we presume normalized representation of floating-point numbers, as recommended in the IEEE 754 floating-point standard. In particular, the first set bit in x is bit $w-1-\texttt{expo}(x)$. We shall use this computation in section 7. Although this method of computing the first set bit may seem contrived, it is not just a theoretical trick to stay within AC^0. A quick test on an SGI computer indicated it to be faster than a hard-coded binary search combined with tabulation. It should be noted that Fredman and Willard have shown that `expo` can be computed in constant time using multiplication without the trick of conversion to floating-point [17]. This solution is, however, worse in theory as multiplication is not in AC^0, and worse in practice as it is both more complicated and uses slower operations.

The rest of this section is devoted to some simple standard-type observations on priority queues that will prove helpful in later sections.

2.1. Variable versus fixed size. As our end result, we want *variable-sized* priority queues whose space consumption is proportional to the current number of keys in them. In our constructions, however, we will just focus on priority queues with some given *capacity* n, meaning priority queues with room for n keys. We can then use the following standard-type simulation to get from given capacities to variable size.

LEMMA 2.1. *We can simulate a basic variable-sized priority queue Q using two basic priority queues Q' and Q'' with given capacity linear in the current number of keys in Q. Each operation on Q will translate into a constant number of operations on Q' and Q''.*

Proof. All keys of Q will either be in Q' or in Q'', but not in both. The smallest key will always be in Q', so $\texttt{find-min}(Q) = \texttt{find-min}(Q')$. Also `insert` and `delete-min` operations are addressed to Q'. While Q has at most four keys, we are in an initial state with all keys in Q' and with $Q'' = \emptyset$. In the initial state, Q' has capacity of five keys. Whenever Q gets more than four keys, we enter a general state described below. If Q later gets down to four keys again, we return to our initial state.

In the general state, after each `insert` and `delete-min`, if $|Q''| > 3$, we extract the two smallest keys from Q'' and insert them in Q'. If $|Q''| \leq 3$, we move all keys from Q'' to Q', set $Q'' := Q'$, reallocate Q' with capacity $3/2$ times the current number of keys, rounding up to the nearest integer, extract the smallest key μ from Q'', and put μ in Q'. Note that we automatically enter the latter case when we insert a fifth

key, bringing us from the initial to the general state.

We need to argue that both Q' and Q'' always have enough capacity for the keys inserted and that their capacities remain linear in the current number of keys in Q. When $Q^* = Q'$ is first allocated, it gets capacity $\lceil 3n/2 \rceil$, where n is the current number of keys. After $\lfloor n/2 \rfloor$ operations, all keys from Q'' are moved to $Q^* = Q'$. In this period, the total number of keys in Q remain between $n/2$ and Q^*'s capacity of $\lceil 3n/2 \rceil$. Next Q^* is assigned to Q''. Since no more keys can be inserted in $Q^* = Q''$, its capacity can never be exceeded. Let n' be the number of keys in Q^* when Q^* was assigned to Q''. After $\lfloor n'/2 \rfloor$ operations, all keys are moved from $Q'' = Q^*$ to Q' and Q^* is deallocated. Hence during the life time of Q^*, the number of current keys in Q remain between $\lceil n'/2 \rceil \geq n/4$ and $\lfloor 3n'/2 \rfloor \leq 9n/4$, both of which are linear in Q^*'s capacity of $\lceil 3n/2 \rceil$. □

2.2. Associating information with the keys. In applications of priority queues, we will often want to associate a word of information with the keys of the priority queue. That is, the priority queue contains key-information pairs, but the pairs are only required ordered with respect to the keys. For example, the information could be a pointer to some additional information about the key. If the keys are themselves one word, we can just append the information directly to the key, so that it becomes a two-word key, where the most significant word is the original key. However, here we will also deal with very short keys, and then we need different techniques.

PROPOSITION 2.2. *Using linear space and a basic priority queue for distinct $2b$-bit keys, we can maintain a priority queue for b-bit keys allowing multiplicity of key values and associating one word of information with each key.*

Proof. First observe that it suffices to associate an index $\in \{0, \ldots, n-1\}$ with each key, for then we can just allocate a table I of n words mapping indices to the desired words of information. If $n \leq 2^b$, we simply append the indices to the keys as an extra least significant part. The uniqueness of the indices implies uniqueness of the augmented keys. If $n > 2^b$, we use a priority queue for distinct b-bit keys and a table with 2^b entries, mapping each possible key value x to a list $L[x]$ of indices of keys with that value. The minimum key-index pair (x, i) is then the minimum key value x together with the first index in $L[x]$. □

2.3. Amortized deletions. Most of the priority queues in this paper are basic priority queues from which only the minimum keys can be deleted. Occasionally, we will need to augment a basic priority queue with a general delete, which is the concern of this subsection. In our applications, for delete we are satisfied with amortized time. In principle the implementation via basic priority queues is straightforward. We just take notes on which keys are to be deleted but postpone the actual removal till the keys come out as minimum keys. Then the amortized cost of general deletions is that of `find-min` and `delete-min`. One obstacle in supporting general deletions is, however, that we need to settle on a good interface between the priority queue and its user.

The main issue that arises in implementing a general `delete`(x, Q) is if the priority queue Q contains several keys with the same value x. Then the user may need to be able to specify which of them is to be deleted. This issue becomes even more prominent in connection with the `decrease-key` operation in Lemma 1.7, where the priority queue actually needs a pointer to the key in order to decrement it at constant cost.

The standard solution in practice is that when `insert`(x, Q) is called, the priority queue Q returns a unique identifier $\imath_x$. If the user wants to delete x, he calls

`delete`$(\imath_x, Q)$, and then the priority queue knows exactly which key to delete.

For a given capacity n, our identifiers will be indices $\{0, \ldots, 2n-1\}$. The priority queue Q has a table D which keeps track of which keys are to be deleted when they come out at minimum keys. Thus `delete`(i_x, Q) is implemented by setting $D[i_x] := 1$. Then `find-min` and `delete-min` are implemented so that they skip keys that are to be deleted according to D.

If ever we run out of indices, we "clean" the priority queue Q as follows. First we extract all keys from the queue, finding all keys to be deleted including their indices. These indices are freed for reuse, and all the nondeleted keys are returned to Q. Since the capacity of Q is n, we identify $\geq n$ keys to be deleted, and the removal of these keys pays for the cleaning.

Now, if we want a variable-sized priority queue, we actually need some extra collaboration with the user. Increasing sizes could quite easily be supported by the priority queue alone, but for shrinking sizes, we may have problems with large indices. The simplest solution is to leave the change of size to the user. That is, whenever the number of keys in the queue doubles or halves, the user should extract all keys from the queue and reinsert them into a new priority queue with capacity twice the current number of keys. Thereby all keys get reassigned their indices. Asymptotically, this does not affect the amortized operation cost. Assuming the interface described above, we conclude the following lemma.

LEMMA 2.3. *A basic priority queue can be augmented to support a general delete with the same amortized time bound as* `find-min` *and* `delete-min`. *This includes variable-size and associated one-word information.*

In applications of Lemma 2.3, we will typically not be explicit about the exact interface but just implicitly assume that the user is maintaining the identifiers as described above.

3. A priority queue for small integers. The goal of this section is to prove the following theorem.

THEOREM 3.1. *There is a priority queue with capacity for n $(w/(\log n \log\log n))$-bit keys, supporting* `insert`, `find-min`, *and* `delete-min` *on integers in constant time per operation and in space $O(n)$.*

Combined with Proposition 2.2, this gives the statement of Theorem 1.3.

We need the following result due to Albers and Hagerup.

LEMMA 3.2 (see [2]). *We can merge two sorted lists, each of at most k (w/k)-bit keys stored in a single word, into a single sorted list stored in two words in time $O(\log k)$.*

As an immediate consequence, we get the following lemma.

LEMMA 3.3. *For $n \geq k$, given two sorted lists of n (w/k)-bit integers, spread over $\lceil n/k \rceil$ words, we can merge them into $\lceil 2n/k \rceil$ words in time $O(n/k \cdot \log k)$.*

Lemma 3.3 is implicit in [2] but in a parallel setting. We note the following simple sequential derivation.

Proof. Pop the first word of each of the two lists to be merged. Apply Lemma 3.2 to these two words, obtaining w_0 and w_1. Now w_0 is the first word in the merged list, while w_1 is pushed on the list from which its biggest key came from. Thus we repeat until one list is empty. Hence each application of Lemma 3.2 extracts one word for the final merged list, while the other goes back for the next iteration. □

Further, we shall use the following result by Raman.

LEMMA 3.4 (see [30]). *There is a priority queue with up to $O(k)$ (w/k)-bit integers, supporting* `insert`, `find-min`, *and* `delete-min` *in constant time. Further,*

it supports `delete` *in constant amortized time.*

To prove Theorem 3.1, we will use Lemma 3.2, Lemma 3.3, and Lemma 3.4 with $k = \log n \log\log n$. First, in order to present the intuition, we give a simple amortized version of the result. Afterwards, we deamortize.

3.1. Amortized bounds. Our data structure has the following components: a "reception" R which is a priority queue from Lemma 3.4 with capacity for k keys. Further we have "buffers" $B_0, \ldots, B_{\log(n/k)}$ containing 0, 1, or 2 sorted lists of $\leq 2^i k$ keys where the keys are packed in words with k keys in each word. By Lemma 3.3, two full lists in B_i can be merged in $O(2^i \log k)$ time, which is $O(2^i \log k/(2^i k)) = O(1/\log n)$ time per key. The minimum of R and of each B_i is collected in a priority queue M from Lemma 3.4. Thus, the minimum key in the whole priority queue is found in M. Using Proposition 2.2, we associate with each key information about which R or B_i it came from.

It is only temporarily that a buffer may contain more than one list. Whenever we start supporting an operation, `insert`, `find-min`, or `delete-min`, we may assume that no buffer contains more than one list.

Deleting the minimum is done by deleting the minimum key from M and deleting it from the corresponding R or B_i. To support constant time deletions of the minimum key of a packed sorted list, we keep an index of the first nondeleted key in the first word. Deleting the minimum is then done by incrementing this index and removing the first word when all keys in it are deleted.

When an element is inserted, it is inserted into R in constant time. If thereby R gets full, the keys are extracted from R in sorted order and packed in a sorted list in one word. This list is put in B_0. If there was already a list in B_0, these two lists are merged. If the resulting list has $\leq 2^0 k = k$ keys, it stays in B_0. Otherwise, it is moved to B_1, which may now have two lists that have to be merged. If the resulting list has $> 2^1 k = 2k$ keys, it is moved to B_2 and so forth. As the lists are merged and moved, the minima of R and the B_i in M are updated accordingly.

Finding the minimum and possibly deleting it takes constant time. For the insertions, first note that inserting the key in R takes constant time. Also, moving keys from R to B_0 takes constant time per key. We will now consider the per-key cost accumulated over the merges. For the analysis, we will be a bit lazy removing deleted keys from the list. Lists in buffer B_i will have exactly $2^i k$ keys, some of which may have to be deleted. It is only when we merge two lists in B_i and discover that $\geq 2^i k$ are to be deleted that we remove exactly $2^i k$ of the keys to be deleted. Thus, the result of the merge is of length $2^i k$ or $2^{i+1} k$. In the former case, the result stays in B_i and it is the $2^i k$ removed keys that pay for the merge. In the latter case, the result is moved to B_{i+1}. Since the per-key cost of the merge is $1/\log n$ and since there are only $\log n$ buffers, the total cost of merges moving keys to larger buffers is $O(1)$ per key.

Since the buffers have exponentially increasing sizes, the space usage is dominated by that of the maximal buffer B_{max}. Further, when we created the buffer B_{max}, we know we did it with a newly merged list from B_{max-1} with $> 2^{max-1} k$ nondeleted keys. Hence the space bound for the buffers is $O(n/k)$, and for R and M, the space is constant.

3.2. Worst-case insertions. We are now going to translate the above priority queue into one working in worst-case constant time. First we ignore deletions of the minimum and focus on deamortizing the merging.

Instead of one reception R, we will in fact have three of them: R', R'', and R'''. The first two, R' and R'', consist of priority queues from Lemma 3.4 with at most k keys. The third, R''', consists of a sorted list of k keys stored in one word. All keys in R''' will be smaller than those in R'', that is, $\max R''' \leq \min R''$. Each of the buffers B_i contains 0, 1, or 2 sorted lists of $2^i k$ keys. If there are two lists, we may have parts of them merged. As in the amortized version, we have a priority queue M from Lemma 3.4 taking care of the minimum keys of R', R'', R''', and each B_i. Using Proposition 2.2, we associate information with the keys in M telling whether they are from R', R'', R''', or some B_i. Until section 3.4, we will pretend that M supports general deletions in constant time.

We proceed in rounds of k insertions. At the beginning of a round, R' is empty and so is the sorted list R'''. When a key is inserted, it is inserted in R'. Moreover, if the priority queue R'' is nonempty, the smallest key of R'' is moved to the end of the sorted list R'''. After k insertions, we have moved all keys from R'' to R'''. We then set $R'' := R'$ and $R' := \emptyset$. Further, we move the packed sorted list R''' of k keys to B_0 and set $R''' = \emptyset$.

If a buffer B_i has two packed sorted lists, we will start merging them. Each list is stored in at most 2^i words of k keys, so by Lemma 3.3, merging them takes $O(2^i \log k) = O(2^i k/\log n)$ time. The merging is viewed as divided into $2^i k/\log n$ steps, each taking constant time.

We are going to merge all buffers in parallel as follows. A round of k insertions is divided into $\log n$ mini-rounds, one for each buffer. We will have a counter i telling which buffer B_i we are working in. When a round starts $i = (\log n) - 1$, and we are going to count i down by one modulo $\log n$ at the end of every mini-round.

In a given mini-round, if B_i contains two lists, for each of the $k/\log n$ inserts in the mini-round, we perform a merge step on the two lists. If the merging is thereby completed, we move the resulting list to B_{i+1}. Finally, no matter whether we merged or not, we complete the round by setting $i := i - 1 \pmod{\log n}$.

In the last mini-round of a round, $i = 0$. It is not until this mini-round is completed, and we have moved a possible merged list from B_0 to B_1, that we complete the round by moving the packed list from R''' to B_0.

3.3. No overloading. We are now going to show that we never get more than two lists in any buffer. Here we are still ignoring deletions. First observe the following.

OBSERVATION 3.5. *Merging two lists in B_i takes 2^i rounds.*

Proof. Within every round we spend one mini-round of $k/\log n$ merge steps in B_i. However, by definition, it takes $2^i k/\log n$ merge steps to merge the lists. □

Concerning B_0, it receives a packed sorted list of k keys from R''' at the very end of every round. Suppose B_0 receives a second list at the very end of one round. By the above observation, the two lists will be merged completely during the subsequent round. Then B_0 is emptied before we, at the very end of the subsequent round, receive a third list from R'''.

For the other buffers, we have to show that when B_i sends a merged list to B_{i+1}, B_{i+1} does not already have two lists. To see this, we first argue the following.

LEMMA 3.6. *Buffer B_i is emptied at most every 2^{i+1}th round.*

Proof. The proof is by induction on i. For $i = 0$, it takes two rounds for B_0 to receive two lists, so B_0 is emptied every other round.

For the inductive step, we note that when the first list is moved to B_i, then B_{i-1} is emptied. Thus, by induction, it takes at least 2^i rounds before a second list is

moved from B_{i-1} to B_i, and by Observation 3.5, it further takes 2^i rounds before the two lists are merged so that the result can be moved to B_{i+1}. □

For $i \geq 0$, by Observation 3.5, B_{i+1} contains two lists for at most 2^{i+1} rounds, but this is the minimum time it takes before it receives a third list. Hence B_{i+1} can never have more than two lists.

Introducing `delete-min`. We are now going to introduce deletions in a way that does not tamper with the scheduling of our parallel merging. A key to the scheduling is that it takes exactly 2^i rounds to merge two lists in buffer B_i. Simultaneously 2^i is an upper bound on the number of rounds two lists can remain in B_i, and a lower bound on the number of rounds it takes before B_i can receive a third list from B_{i-1}.

When the minimum key μ is to be deleted from M, we extract μ together with information telling us whether it comes from R', R'', R''', or some B_i. If μ comes from R' or R'', we delete μ with `delete-min`. If μ comes from R''', it is the first key in the list, and it is just deleted. Clearly, deleted keys can only make it faster to move keys from R'' to R''', so this move is still completed within one round.

Suppose now that μ is from some B_i. If we have two partially merged lists in B_i, we preserve the original lists till the merging is completed. Keys are deleted from the original lists, that is, for each list, we have a min-pointer to the first nondeleted key, and one of these two keys is the smallest key in B_i. If μ is deleted from B_i, first we delete it by moving the min-pointer of the list of μ to the μ's successor in the list. Second, we find out which of the two lists in B_i now has the smallest nondeleted key. This new smallest key of B_i is inserted in M. In connection with such a deletion, we further make one merge step in B_i. As a consequence, if we have d deletions from B_i during the merging of B_i, the merging will terminate at least d steps early. Hence to terminate the merging within the 2^i rounds, we have d time steps that we can use to remove d keys from the merged list. These removals may be interspersed with new deletions via M. New deletions are still done in the original lists, but we make a simultaneous deletion in the merged list. Thus we arrive at a merged list with all deleted keys removed within 2^i rounds. In case we finish early, however, for the sake of scheduling we will not view the merging as completed before exactly 2^i rounds have passed. Consequently, this gives us the following.

OBSERVATION 3.7. *Merging two lists in B_i interspersed with deletions takes 2^i rounds.*

LEMMA 3.8. *Despite deletions, buffer B_i is emptied at most every 2^{i+1}th round.*

Proof. As the proof of Observation 3.5, the proof is by induction on i. For $i = 0$, we note that B_i still needs to receive at least two lists before it can be emptied, but now, of course, it may need more lists.

For the inductive step, suppose B_i has just been emptied. When the first new list is moved to B_i, B_{i-1} is emptied. Thus, by induction, it takes at least 2^i rounds before a second list is moved from B_{i-1} to B_i, and by Observation 3.7, it further takes 2^i rounds before the two lists are merged so the result can be moved to B_{i+1}. Note that the merged list is not moved to B_{i+1} if it contains $\leq 2^i k$ keys. Then it will take at least another 2^i rounds before B_i is emptied. □

Now, by Observation 3.7, B_{i+1} contains two lists for at most 2^{i+1} rounds, but this is the minimum time it takes before it receives a third list.

3.4. Deletions from *M*. Recall from section 3.2 that when we emptied a buffer, we assumed that we could just delete the minimum of the buffer from M in constant time. However, M does not solve deletions in constant time. We circumvent this problem as follows. While a buffer contains two lists, it may actually have the minima

of both lists in M. The minima are then associated with the lists rather than with the buffers, so when a list is moved, its minimum remains unchanged in M. The point is that when a buffer receives its second list, it will have two lists for at least one round of k inserts, and this will be plenty of time for us to remove the bigger of the minima of the two lists from M. More precisely, during every round, we will copy M to M', extracting the keys one by one from M, and insert them in M' only if they are not the bigger of the minima of two lists in a buffer. We make one such move from M to M' in connection with each insert. Since we have $O(\log n)$ keys in M, M is emptied within $O(\log n)$ insertions, which is less time than a round. When M is emptied, we swap the roles of M and M'. This completes the proof of Theorem 3.1.

3.5. Sequential storage. It should be mentioned that merging previously has been used in connection with priority queues to adapt them for sequential storage [15]. Our algorithm seems simpler than the one in [15], so it could be interesting to see how it performs on sequential storage if we ignore the packing of keys in words.

4. Priority queues for arbitrary integers. In this section we derive the time bounds of Theorem 1.1, postponing the space bounds to the next section. To get from arbitrary integers to short integers we will use a recursive range reduction which can be seen as a simple specialized variant of van Emde Boas' data structure [35, 36], inspired by the developments in [6, 22, 24, 27]. Let $T(n, b)$ be the time for `insert` and `delete-min` in a priority queue with up to n distinct b-bit integers supporting `find-min` in constant time. We assume that b but not n is known in advance. By Theorem 3.1, $T(n, w/(\log n \log\log n)) = O(1)$. We will show that

$$T(n, b) = O(1) + T(n, b/2). \tag{4.1}$$

Here b is assumed to be a power of 2. We will now implement a b-bit key priority queue Q. For any b-bit integer x, let $\texttt{low}(x)$ denote the contents of last $b/2$ bits, and let $\texttt{high}(x)$ denote the contents of the first $b/2$ bits. Thus, $x = \texttt{high}(x)2^{b/2} + \texttt{low}(x)$.

Let H be a $(b/2)$-bit key priority queue. H will be used for the `high`-values of the inserted keys. We will assume that we can freely allocate constant access time tables with entries from the values in H. We shall return to the implementation of these, including space issues, in the next section. Particularly, such tables allow us to answer whether a value is in H in constant time. Moreover, for each $h \in H$, they allow us to store a $(b/2)$-bit key $l[h]$ and a $(b/2)$-bit key priority queue $L[h]$ such that

- $\{l[h]\} \cup L[h] = \{\texttt{low}(x) | x \in Q, \texttt{high}(x) = h\}$.
- $l[h] < \min L[h]$.

Besides, we will have an integer $h_{min} = \min H$. Initially $H = \emptyset$ and $h_{min} = \infty$. If Q is nonempty,

$$\texttt{find-min}(Q) = 2^{b/2} h_{min} + l[h_{min}] = (h_{min} \ll b/2) + l[h_{min}].$$

We are now ready to present concrete implementations of `insert` and `delete-min` settling recurrence (4.1).

ALGORITHM A. $\texttt{insert}(x, Q)$
A.1. $(h, l) := (\texttt{high}(x), \texttt{low}(x))$.
A.2. If $h \notin H$,
A.2.1. $h_{min} := \min(h, h_{min})$.
A.2.2. $\texttt{insert}(h, H)$.
A.2.3. $l[h] := l$.
A.2.4. Allocate an empty priority queue $L[h]$.

A.3. If $h \in H$,
A.3.1. $(l[h], l) := (\min(l[h], l), \max(l[h], l))$.
A.3.2. `insert`$(l, L[h])$.
ALGORITHM B. `delete-min`(Q)
B.1. If $|L[h_{min}]| > 0$, then
B.1.1. $l[h_{min}] :=$ `find-min`$(L(h_{min}))$.
B.1.2. `delete-min`$(L(h_{min}))$.
B.2. Else
B.2.1. Deallocate $L[h_{min}]$.
B.2.2. $h_{min} :=$ `find-min`(H).
B.2.3. `delete-min`(H).

Each of the above algorithms makes exactly one recursive call, so this settles the time bound described in recurrence (4.1).

We are now essentially in the position to settle Theorem 1.1. By Proposition 2.2 we can support a basic w-bit key priority queue with multiplicity of key values and associated information if we have a basic priority queue for distinct $2w$-bit keys. Starting with these $2w$-bit keys, we apply recurrence (4.1) $\log(2 \log n \log\log n) = O(\log\log n)$ times. In time $O(\log\log n)$ per operation, this reduces our problem to dealing with $(w/(\log n \log\log n))$-bit keys. By Theorem 3.1 together with Lemma 2.1, these can be dealt with in constant time, so the total time is $O(\log\log n)$ per operation. We are still missing, however, how we can test membership in H and how we can use the members of H as entries for tables. These issues are the subject of the next section.

5. Space issues. In order to clarify the space usage of our algorithm, we need to discuss dynamic dictionaries. A *dynamic dictionary* maintains a subset X of a typically much bigger universe $\mathcal{U}$. Elements can be inserted and deleted from X, and further, we can ask membership queries, that is, given $x \in \mathcal{U}$, does x belong to X? In addition, we want it to associate with each element $x \in X$ a unique index $\imath_X$ of size $O(|X|)$. These indices allow us to use X, or really $\{\imath_x | x \in X\}$, as entries to arrays/tables. Thus a dynamic dictionary is exactly what we needed in the last section for testing membership of H and for allocating the tables l and L with entries from H. For our $O(\log\log n)$ time bound for priority queues to hold, we can only use dictionaries where all operations are supported in constant time.

LEMMA 5.1 (see [11, 16, 26]). *We can support a dynamic dictionary for a set* $X \subseteq \{0,1\}^w$

(1) *deterministically with constant operation time in space* $O(|X|2^{\varepsilon w})$ *using standard* AC^0 *operations (here* ε *is any positive constant),*
(2) *randomized with expected constant operation time in space* $O(|X|)$.

Proof. The randomized solution is simply Carter and Wegman's multiplicative hashing [11]. For the deterministic solution, based on Fredkin's tries [16], Mehlhorn [26, section III, Theorem 1] has shown that for universe size N and for any k, we can get operation time $O(\log_k N)$ using $O(|X|k)$ space. Here $N = 2^w$, and we set $k = 2^{\varepsilon w}$, getting an operation time of $O(\log_{2^{\varepsilon w}} 2^w) = O(1/\varepsilon) = O(1)$. The algorithm divides by k and division is not an AC^0 operation. However, we can choose k as a power of 2, and then the division is just a right shift. □

The space dependence on w in the deterministic AC^0 solution is unsatisfying, but if we insist on AC^0 operations, Andersson et al. [4] have proved that there cannot be a solution to the dynamic dictionary problem combining constant operation time with space polynomial in $|X|$.

We will now take a closer look at the dynamic dictionaries needed for the priority queue from the last section. The recursive structure only depends on the keys in the queue. Hence the per-key space consumption can be found by analyzing `insert`. In connection with a call to `insert`, we may introduce a new entry h to l and L. Further, we allocate an empty priority queue $L[h]$. Since the depth of the recursion is $O(\log\log n)$, we conclude that both the total number of dictionaries used and the total number of entries used in all dictionaries is $O(n \log\log n)$.

Following ideas of van Emde Boas [35], we can reduce the number of dictionaries and entries to $O(n)$ as follows. The keys are collected as they are inserted into a list of micro sets, each of size $\log\log n$. For each micro set, it is only the minimum key that is inserted in our recursive priority queue Q. Hence the overall minimum key μ is always found in Q. When μ is deleted from Q, it is deleted from the corresponding micro set S. The new minimum x of $S \setminus \{\mu\}$ is found by a simple scan in $O(|S|) = O(\log\log n)$ time, and then x is inserted in Q. If ever two neighboring sets together get less than $\log\log n$ elements, we merge them. This can only happen in connection with a deletion, and only for one pair of neighbors. The merge is done in $O(\log\log n)$ time. Thus, without violating our $O(\log\log n)$ time bound, we can reduce the number of keys in Q to $O(n/\log\log n)$. Hence the total number of dictionaries and dictionary entries in Q become $O((n/\log\log n)\log\log(n/\log\log n)) = O(n)$. The space bounds in Lemma 5.1 are both linear in $|X|$, so we get the following theorem.

THEOREM 5.2. *There is a priority queue supporting* `find-min` *in constant time and* `insert` *and* `delete-min` *in $O(\log\log n)$ time, where n is the current number of keys in the queue. It uses $O(n2^{\varepsilon w})$ space and may be implemented with standard AC^0 operations. Also, there is a randomized implementation using non-AC^0 operations giving $O(\log\log n)$ expected time and $O(n)$ space.*

In the introduction, we also mentioned that we could get $O(n + 2^{\varepsilon w})$ space deterministically for a monotone priority queue with an $O(\log\log n)$ amortized operation time. This matches the space bound of the deterministic sorting $O(n \log\log n)$ algorithm of Andersson et al. [3], so to get the monotone amortized priority queue, we simply plug the sorting algorithm from [3] into Theorem 1.4.

6. A priority queue for strings. In this section, we implement a priority queue for ℓ-word keys using the priority queues for one-word keys from Theorem 1.1. Essentially, we just use Fredkin's trie [16] as presented by Mehlhorn [26, section III]. Each key is viewed as a string of ℓ characters. The keys are stored in a trie, which is a rooted tree of depth ℓ. Each edge is labeled by a one-word character, and each leaf v represents the string of characters labeling the path from the root to v. Thus n strings are represented by a trie with n leaves.

In order to facilitate basic priority queue operations, we associate with each internal node with more than one child, a dictionary and a priority queue over the characters of its outgoing edges. Thereby we get a total of $n - 1$ entries in the dictionaries and the priority queues, so by Lemma 5.1 and Theorem 5.2, the total space bound for all the dictionaries and priority queues becomes $O(n2^{\varepsilon w})$ for a deterministic solution implemented with standard AC^0 operations, and $O(n)$ for a randomized solution using non-AC^0 operations. Additionally, we need $O(n\ell)$ space to store the keys.

Now, in order to find the minimum string in the queue, we descend from the root of the trie, using a `find-min` operation at each internal node to figure out which child to descend to. Thereby we end at the leaf representing the minimum string. Since `find-min` takes constant time, the time bound for finding the minimum string is $O(\ell)$.

Having performed `find-min` and identified the path from the root to the leaf w

representing the minimum string, we can delete the minimum string by deleting the path from the leaf and up to its first ancestor v with > 1 child. Let a be the character that labeled the edge from v towards w. Then a has to be deleted from the dictionary and the priority queue at v. Deleting the path nodes takes time $O(\ell)$, deleting a from the dictionary at v takes constant time, and deleting a from the priority queue at v is done using `delete-min` in time $O(\log\log n)$. Finally, if v has now only one child, the dictionary and priority queue at v are deallocated in constant time. Consequently, the cost of deleting the minimum string is $O(\ell + \log\log n)$.

To insert a new string s, we first use $O(\ell)$ finds in dictionaries to descend to the node v at which the path representing s should branch out from the current trie. Then the remaining part of the path to the leaf representing s is added to the trie. Finally, the character a on the new path below v has to be inserted in the dictionary and priority queue at v in time $O(\log\log n)$. Note that if v does not already have > 1 child, we first allocate a dictionary and priority queue at v in constant time. Thus, the total cost of an insertion is $O(\ell + \log\log n)$.

7. From monotone priority queues to sorting. We are now going to present a general reduction of Theorem 1.4 from monotone priority queues. More precisely, fix $k = \Theta(\log n \log\log n)$ as a power of 2. Each key of the priority queue will participate in one sorting and go through one priority queue for (w/k)-bit integers and one priority queue with capacity for at most $k+1$ integers at the time. The former priority queue has constant operation cost by Theorem 3.1, and the latter priority queue has constant operation cost due to the following result of Fredman and Willard [18].

LEMMA 7.1 (see [18]). *Given $O(n)$ space and preprocessing time, we can maintain an unrestricted priority queue for up to $O(\log^2 n)$ keys at constant amortized cost per operation.*

Thus, the operation cost of our priority queue becomes the per-key cost of sorting plus a constant. The construction of Lemma 7.1 relies heavily on multiplication, which is not an AC^0 operation [9]. However, Andersson, Miltersen, and Thorup [5] have shown that the construction can be implemented using AC^0 operations only if we allow ourselves some simple nonstandard AC^0 operations.

7.1. Preliminaries. In our reduction, keys are viewed as divided into k (w/k)-bit characters. For $i = 0, \ldots, k-1$, $x\langle i\rangle$ denotes character i in x. Further, for $i \le j < k$, $x\langle i..j\rangle$ denotes $x\langle i\rangle \cdots x\langle j\rangle$. For different keys x and y, $\texttt{split}(x,y)$ denotes the first i such that $x\langle i\rangle \neq y\langle i\rangle$. If $x = y$, we define $\texttt{split}(x,y) = k$. Since our integer keys are stored with the most significant bit first, $x < y \iff x\langle\texttt{split}(x,y)\rangle < y\langle\texttt{split}(x,y)\rangle$.

LEMMA 7.2. *For any two words x, y, $\texttt{split}(x,y)$ is computable in constant time with standard AC^0 operations.*

Proof. If $x = y$, $\texttt{split}(x,y) = k$, and we are done, so assume $x \neq y$. Then $\texttt{split}(x,y) = \lfloor (w - 1 - \texttt{expo}(x \oplus y))/(w/k) \rfloor$. Now k and w are powers of 2, and hence so is w/k, that is $w/k = 2^e$ for some e. Now, we compute $\texttt{split}(x,y)$ as $(w - 1 - \texttt{expo}(x \oplus y)) \gg e$. ∎

7.2. The general idea. Fix μ to denote the minimum key in Q. Since μ is the minimum key, for any other keys $x, y \in Q$

$$\texttt{split}(x,\mu) < \texttt{split}(y,\mu) \Rightarrow x > y. \tag{7.1}$$

Thus $\texttt{split}(\cdot,\mu)$ provides us with a preordering of the keys in Q. We will use this for an initial bucketing of the keys into buckets $B_0, \ldots, B_{k-1}$. That is, when a key $x \neq \mu$ is first inserted, it is put in a bucket $B_{\texttt{split}(x,\mu)}$.

Now, when μ is increased, this may violate the placement of some keys x in a bucket B_i. That is, when μ increases, we may get $\texttt{split}(x, \mu) > i$. In this case, all such x will be sorted and put in a list L_i. It will turn out that we never will need more than one list L_i for $i = 0, \ldots, k-1$.

Thus each key of Q is either in a buckets B_i or in some sorted lists L_i but not in both.

The minimum key in all the lists L_i is maintained by a priority queue from Lemma 7.1. Maintaining the minimum key in each bucket B_i will essentially be done using the priority queue from Theorem 3.1.

It should be noted that bucketing according to $\texttt{split}(\cdot, \mu)$ goes back to Denardo and Fox [14]. However, in [14], when a key x is misplaced because of an increase in the minimum key μ and $\texttt{split}(x, \mu)$, x is reinserted in a new bucket $B_{\texttt{split}(x,\mu)}$. This gives up to k reinsertions per key. Here, instead, we sort all misplaced keys, adding them to a set of sorted lists. The key insight is that only k sorted lists are needed at any time.

7.3. Components. We are now ready to formally state the components of the reduction.

DESCRIPTION C.

C.1. The smallest key μ in the queue. Thus `find-min` is trivially implemented in constant time.

C.2. Some sorted lists $L_0, \ldots, L_{k-1}$ of keys where L_i satisfies for all $x \in L_i$: $\texttt{split}(\mu, x) > i$. Note that $i < j$ implies that any key in L_j could also have been in L_i. Also note that all keys in L_{k-1} have the same value as μ.
Using a priority queue from Lemma 7.1, we keep track of which of the lists $L_0, \ldots, L_{k-1}$ has smallest first key. This key is then the smallest of all the keys in the lists, so $\mu_L = \min \bigcup_i L_i = \min_i \{\min L_i\}$ is computable in constant amortized time.

C.3. Some buckets $B_0, \ldots, B_{k-1}$. Every key $x \in B_i$ has $\texttt{split}(x, \mu) = i$. Thus every key x in $\bigcup_i B_i$ fits in a unique bucket B_i; namely $B_{\texttt{split}(x,\mu)}$.
The keys in B_i are stored in a monotone priority queue from Theorem 3.1 with (w/k)-bit key and w-bits of information. Each keys $x \in B_i$ is entered as the pair $(x\langle i\rangle, x)$. That is, in B_i, the original keys are only ordered with respect to character i.

C.4. The maximum index i_{max} of a nonempty B_i. This is done using a single word α. In α, we set bit $w - 1 - i$ if and only if B_i is nonempty. Then $i_{max} = \texttt{expo}(\alpha)$.

Above it is understood that each key in Q is only represented once, either as the minimum key μ, or in one L_i, or in one B_i.

7.4. Implementing `find-min` and `insert`. As mentioned above, `find-min` is trivially implemented in constant time by returning the value of μ. Concerning insertions, since the priority queue is monotone, an insertion cannot change the minimum key μ, and hence an insertion cannot misplace any key currently in the queue. The implementation is therefore straightforward.

ALGORITHM D. $\texttt{insert}(x, Q)$

D.1. If $x = \mu$, add x to L_{k-1}.

D.2. If $x \neq \mu$ then

D.2.1. $i := \texttt{split}(x, \mu)$.

D.2.2. $\texttt{insert}((x\langle i\rangle, x), B_i)$.

D.2.3. Set bit $w - 1 - i$ in α (cf. C.4).

7.5. Implementing `delete-min`. In order to implement `delete-min`, we need to find the current second smallest key μ' in Q. Note that in case of multiplicities, the second smallest key may have the same value as μ.

LEMMA 7.3. *Let μ_B be the smallest key in $\bigcup_i B_i$. Then μ_B is in $B_{i_{max}}$ and if $(a, x) = \texttt{find-min}(B_{i_{max}})$, $\mu_B\langle 0, \ldots, i_{max}\rangle = \mu\langle 0, \ldots, i_{max} - 1\rangle a$.*

Proof. By (7.1), for all $i < i_{max}$, $\max B_{i_{max}} < \min B_i$. Further, since the keys in $B_{i_{max}}$ agree on characters $0, \ldots, i_{max} - 1$, the smallest key in $B_{i_{max}}$ must minimize character i_{max}. □

Let μ_B, and a be as defined in the above lemma. Further, let μ_L be the minimum key over all lists L_i. Then the current second smallest key μ' is the smaller of μ_B and μ_L, or either if $\mu_B = \mu_L$. From C.2 we know that $\mu_L = \min_i\{\min L_i\}$ is available in constant time. On the other hand, μ_B is not automatically available. However, by Lemma 7.3, $\mu_B\langle 0, \ldots, i\rangle = \mu\langle 0, \ldots, i_{max}\rangle a$, where μ, i_{max}, and a are all available in constant time. That is, we know $\mu_B\langle 0, \ldots, i_{max}\rangle$.

We now divide in cases depending on whether or not $\mu_L\langle 0, \ldots, i_{max}\rangle < \mu_B\langle 0, \ldots, i_{max}\rangle$.

Suppose $\mu_L\langle 0, \ldots, i_{max}\rangle < \mu_B\langle 0, \ldots, i_{max}\rangle$. Then $\mu' = \min\{\mu_B, \mu_L\} = \mu_L$. Hence we delete μ_L from the list L_i it came from and set $\mu := \mu_L$.

We need to argue that replacing μ with μ_L maintains for all $x \in L_i : \texttt{split}(\mu, x) > i$ and for all $x \in B_i : \texttt{split}(\mu, x) = i$ for $i = 0, \ldots, k-1$. In the argument, μ_{old} denotes the old value of μ.

To see that for all $x \in L_i : \texttt{split}(\mu, x) > i$ is maintained, note that for any x, y, z, $x \le y \le z$ implies $\texttt{split}(x, z) \le \texttt{split}(y, z)$. Since μ_L was the second smallest key in the queue, it follows that replacing μ with μ_L can only increase $\texttt{split}(\mu, x)$ for the remaining keys x. Hence the replacement maintains for all $x \in L_i : \texttt{split}(\mu, x) > i$.

We will now argue that for all $x \in B_i : \texttt{split}(\mu, x) = i$ is maintained for $i = 0, \ldots, k-1$. Since μ_L is strictly smaller than all keys in the B_i, by (7.1), $\texttt{split}(\mu_L, \mu) \ge i_{max}$. Hence $\mu_L\langle 0..i_{max} - 1\rangle = \mu_{old}\langle 0..i_{max} - 1\rangle$, so $\texttt{split}(\mu, x) = i$ is preserved for $i < i_{max}$. It follows for $i < i_{max}$ that for all $x \in B_i : \texttt{split}(\mu, x) = i$ is maintained. Consider any $x \in B_{i_{max}}$. Then $x\langle 0..i_{max} - 1\rangle = \mu_{old}\langle 0..i_{max} - 1\rangle = \mu_L\langle 0..i_{max} - 1\rangle$. Moreover, $x\langle 0..i_{max}\rangle \ge \mu_B\langle 0..i_{max}\rangle > \mu_L\langle 0..i_{max}\rangle$, so we conclude that $\texttt{split}(\mu_L, x) = i$, as desired. Since $B_i = \emptyset$ for all $i > i_{max}$, we conclude that for all $x \in B_i : \texttt{split}(\mu, x) = i$ is maintained, for $i = 0, \ldots, k-1$, as desired.

Suppose $\mu_B\langle 0, \ldots, i_{max}\rangle \le \mu_L\langle 0, \ldots, i_{max}\rangle$. Then $\mu'\langle 0, \ldots, i_{max}\rangle = \mu_B\langle 0, \ldots, i_{max}\rangle$. Further, we have the following key lemma.

LEMMA 7.4. *If $\mu_B\langle 0, \ldots, i_{max}\rangle \le \mu_L\langle 0, \ldots, i_{max}\rangle$, $L_{i_{max}} = \emptyset$.*

Proof. Suppose for a contradiction that there is a key $x \in L_{i_{max}}$. By definition (cf. C.2), $\texttt{split}(x, \mu) > i_{max}$. However, $\mu \le \mu_L \le x$, so by (7.1), $\texttt{split}(\mu_L, \mu) > i_{max}$. Hence $\mu_L\langle 0, \ldots, i_{max}\rangle = \mu\langle 0, \ldots, i_{max}\rangle$. Since μ is the smallest key in Q and $\texttt{split}(\mu_B, \mu) = i_{max}$, this contradicts $\mu_B\langle 0, \ldots, i_{max}\rangle \le \mu_L\langle 0, \ldots, i_{max}\rangle$. □

Since $L_{i_{max}} = \emptyset$ is empty, we can now extract the set X of all keys x from $B_{i_{max}}$ with $x\langle i_{max}\rangle = a$, sort S, and make S the new sorted list $L_{i_{max}}$. Since $\mu'\langle 0, \ldots, i_{max}\rangle = \mu_B\langle 0, \ldots, i_{max}\rangle = \mu\langle 0, \ldots, i_{max} - 1\rangle a$, all keys x in $L_{i_{max}}$ satisfy $\texttt{split}(x, \mu') > i_{max}$. Now, μ_B is simply the first key in $L_{i_{max}}$, so `delete-min` is implemented, setting μ to the smaller of μ_B and μ_L, deleting μ from the list L_i it is in.

We need to argue that we have maintained for all $x \in L_i : \texttt{split}(\mu, x) > i$ and for all $x \in B_i : \texttt{split}(\mu, x) = i$ for $i = 0, \ldots, k-1$. Let μ_{old} and be the old value of μ. Further, we have μ' denoting the new value of μ. We have already argued that for

all $x \in L_{i_{max}} : \texttt{split}(\mu', x) > i_{max}$, and the rest of the argument follows the same lines as in the previous case of $\mu_L\langle 0, \ldots, i_{max}\rangle < \mu_B\langle 0, \ldots, i_{max}\rangle$. First, since μ' was the second smallest key, $\texttt{split}(\mu, x)$ can only increase for the remaining keys, so for all $x \in L_i : \texttt{split}(\mu, x) > i$ is preserved for $i \neq i_{max}$.

It remains to argue that for all $x \in B_i : \texttt{split}(\mu, x) = i$ is maintained for $i = 0, \ldots, k-1$. Since $\mu'\langle 0..i_{max}\rangle = \mu_B\langle 0..i_{max}\rangle$, and $\texttt{split}(\mu_B, \mu_{old}) = i_{max}$, $\mu'\langle 0..i_{max} - 1\rangle = \mu_{old}\langle 0..i_{max} - 1\rangle$. Hence for all $x \in B_i : \texttt{split}(\mu, x) = i$ is maintained for $i < i_{max}$. However, all remaining keys x in $B_{i_{max}}$ have $x\langle i_{max}\rangle > a$, so $\texttt{split}(\mu', x) = i_{max}$, as desired. Hence, we conclude that for all $x \in B_i : \texttt{split}(\mu, x) = i$ is maintained for $i = 0, \ldots, k-1$. This completes our proof that `delete-min` is implemented correctly.

The above of description `delete-min` is turned into pseudocode as follows.

ALGORITHM E. `delete-min`(Q)

E.1. $\mu_L := \min \bigcup_h L_h$. Here $\min \emptyset = \infty$ and $\texttt{split}(\mu, \infty) = 0$ for $\mu < \infty$.

E.2. $i_{max} := \texttt{expo}(\alpha)$ (cf. C.4).

E.3. $(a, x) := \texttt{find-min}(B_{i_{max}})$.

E.4. If $\mu\langle 0, \ldots, i_{max} - 1\rangle a \leq \mu_L\langle 0, \ldots, i_{max}\rangle$ then

E.4.1. loop

E.4.1.1. $L_{i_{max}} := L_{i_{max}} \cup \{x\}$.

E.4.1.2. $\texttt{delete-min}(B_{i_{max}})$.

E.4.1.3. if $B_{i_{max}} = \emptyset$, unset bit $w - 1 - i_{max}$ in α (cf. C.4) and exit the loop.

E.4.1.4. $(b, x) := \texttt{find-min}(B_{i_{max}})$.

E.4.1.5. if $b > a$, exit the loop.

E.4.2. Sort $L_{i_{max}}$.

E.4.3. $\mu_L := \min\{\mu_L, \min L_{i_{max}}\}$.

E.5. $\mu := \mu_L$.

E.6. Delete μ_L from its list.

Above we maintain the priority queue spending constant amortized time per `insert` or `delete-min` operation plus sorting. Each key participates in exactly one sorting; namely when it is transferred from $B_{i_{max}}$ to $L_{i_{max}}$. Thus sorting is only done for disjoint subsets of the keys, and all keys sorted are current keys in the queue. This completes the proof of Theorem 1.4 except for the remarks on a variant using only standard AC^0 operations. For $s(n)$ the per-key cost of sorting n keys and $\operatorname{logg} n = \log n \log\log n$, it was claimed that we could support `insert` and `delete-min` in $O(\sum_{i=0}^{\operatorname{logg}^* n} s(\operatorname{logg}^{(i)} n))$ amortized time.

The only part of our construction that is not implementable with standard AC^0 operations is a priority queue Q_L from Lemma 7.1, which was used to maintain the smallest of the minima of the lists L_i (cf. C.2). There are only $k = \operatorname{logg} n$ such lists L_i, so in order to get the claimed bound, it suffices to maintain their minima recursively. Unfortunately, the Q_L is not immediately monotone even though the overall priority queue is monotone. The problem is in `delete-min` if we pass the condition of step E.4 and create a new list $L_{i_{max}}$. If $\mu_B < \mu_L$, the new list $L_{i_{max}}$ will have minimum μ_B which is strictly smaller than the previous minimum μ_L of Q_L.

To preserve monotonicity, note that we do not need to know μ_L if $\mu\langle 0, \ldots, i_{max} - 1\rangle a \leq \mu_L\langle 0, \ldots, i_{max}\rangle$ (cf. E.4). Set $\mu_B^- := \mu\langle 0, \ldots, i_{max} - 1\rangle a \ll (w - i_{max} - 1)$. Then $\mu < \mu_B^- \leq \mu_B$. Before deleting μ from Q_L, we insert μ_B^-. Suppose μ_B^- becomes the minimum after μ is deleted. Then we pass the condition of step E.4, and the new minimum μ_B of $L_{i_{max}}$ is not smaller than μ_B^-, so monotonicity is respected. If μ_B^- does not become the new minimum after deleting μ, we do

not pass the condition of step E.4. Then μ_B^- is just skipped whenever it comes out. Here we refer to the methods described in section 2.3. This completes the proof of Theorem 1.4 including the case of standard AC^0 operations. □

8. Getting constant cost `decrease-key`. In this section, we want to prove the statement of Lemma 1.7:

> *Provided a monotone/unrestricted basic priority queues for up to $f(n) = \Omega(\log^2 n)$, $f(n) = O(n)$, keys supporting* `find-min`*,* `insert`*, and* `delete-min` *in constant amortized time, there is a monotone/unrestricted priority queue with capacity n, supporting the operations* `find-min`*,* `insert`*, and* `decrease-key` *in constant amortized time, and* `delete` *in time $O(\log n/\log f(n))$. The reduction is deterministic and uses linear space, and it can be implemented with standard AC^0 operations.*

The result of Lemma 1.7 is essentially achieved by Fredman and Willard's AF-heaps [18, section 2.2], except that the AF-heaps are not concerned with monotonicity. As in [18], we shall refer to the small constant cost priority queues from the assumption of the lemma as *atomic priority queues*. By Lemma 2.3, we can assume that atomic priority queues support general deletions of arbitrary keys, also at amortized constant cost. Below we briefly repeat the construction of AF-heaps, referring the reader to [18, section 2.2] for further explanations. The emphasis in our presentation is to ensure that if the overall priority queue is monotone, then so are all involved atomic heaps.

The keys in the overall priority queue Q are distributed over B-trees with $B = f(n)/\log n$. Here, a B-tree is a rooted tree where all leaves are on the same depth and where the degree of each internal node is between $B/2$ and B. Since the total number of keys is n, the depth of any involved B-tree is $O(\log n/\log B)$. Since $f(n) = \Omega(\log^2 n)$, $f(n) = \Omega(\sqrt{B})$, and hence the maximal depth is $O(\log n/\log B) = O(\log n/\log f(n))$, which is the claimed cost of deletions.

The keys are placed in heap order meaning that the keys get smaller as we get closer to the root. Thus, the minimum key is among the roots of the trees. There will be at most B trees of each possible height. Since the maximal height is $O(\log n/\log B)$, the maximal number of trees is $O(B\log n/\log B) = O(B\log n) = O(f(n))$. An atomic priority queue, called the "root queue," can therefore maintain the minimum key among the roots, which in turn is the minimum key in the priority queue. Further, at each internal node, we have an atomic priority queue, called the "children queue," maintaining the smaller key among its $\le B = O(f(n))$ children.

In order to implement `insert`, `decrease-key`, and `delete`, we will temporarily relax the requirements to how many keys there are in the different priority queues. Also, we will temporarily allow some node in a B-tree to be empty in the sense of having no key associated with it.

`insert`(x, Q). Create a single node tree containing x, adding x to the root queue. Since the root queue always contains the smallest key in Q, we are not violating monotonicity of the root queue if the overall priority queue is monotone.

`delete`(x, Q). If x is a leaf, we just delete x. Otherwise, we replace x by its smallest child y, and then, recursively, we delete y from where it was. Since $y \ge x$, we are sure to avoid violating monotonicity if we insert y into the children queue of the parent of x before we delete x. Above, if x was a root, the children queue of its "parent" is understood to be the root queue.

`decrease-key`(x, d, Q). First, if x is not a root, cut x from its parent, deleting x from the children queue of its parent. Now the subtree of x is an independent tree

with root x, and then x is added to the root queue. If x is not a root, x is already in the root queue. Now we replace x by $x-d$ in the root priority queue. Since the root queue always contains the smallest key in Q, monotonicity of the root queue is not violated if Q is monotone.

The above operations may result in too many trees, or in some internal node having $< B$ children. As long as any of these problems appear, we repeat.

(i) Suppose at some stage we get B or more trees of the same height h. We then collect a set F of B such trees and remove all but the smallest root x in F from the root queue. Next we create a new tree T. Its root is a new node v with the same value as x, and the subtrees under v are the trees in F. Then T is a B-tree of height $h+1$. The root v already has its value in the root queue, and the roots of F are collected in children queue under v. Finally, we perform a `delete`(v, Q), getting rid of the extra copy of x.

(ii) If at some stage, some internal key x gets $< B/2$ children, we first cut all the children from x. The subtrees of the children are now independent, and the children are moved from priority queue at x to the root queue. Next, if x is not a root, we cut x from its parent, making x a single node tree, and move x from the priority queue at its parent to the root queue.

Above, we always apply (i) before applying (ii). Thereby, we will never get more than $3B/2$ trees of any given height. To see the efficiency of AF-heap, consider the potential function $\Pi = \tau + 2\gamma$ where τ is the number of trees in Q and γ is the sum over all internal nodes v of $B - d(v)$, where $d(v)$ is the degree of v.

Now, `insert`(x, Q) creates a single node tree x in constant time. This increments τ by one while γ is unchanged since x is a leaf. Thus the amortized cost of `insert`(x, Q) is $O(1)$, as desired.

Calling `delete`(x, Q) takes time proportional to the height, which is $O(\log n / \log f(n))$. The only structural change is that some leaf is deleted, increasing γ by 1. Hence the amortized cost of `delete`(x, Q) is $O(\log n / \log f(n))$, as desired.

When calling `decrease-key`(x, d, Q), structurally, we cut x from its parent. This turns the subtree rooted in x into a new tree whose root x is decreased by d and added to the root queue. All this takes constant time. Since we decrease the degree of x's original parent, we increase γ by one. Further, the creation of a new tree increases τ by 1. Summing up, the total amortized cost of `decrease-key`(x, d, Q) is $O(1)$, as desired.

Finally, we need to argue that (i) and (ii) have no associated cost. Now (i) takes $O(\log n / \log f(n))$ time as it deletes v. However, structurally, it decreases the number τ of trees by $B - 1 = \Omega(f(n)/\log n)$, thus paying for the time spent. Note that the new internal node v does not change γ since it gets B children. Hence γ is only increasing by one as part of the deletion.

Clearly (ii) takes $O(B)$ time to execute. It may create as many as $B/2$ new trees, thus increasing τ by $B/2$. On the other hand, the change in γ around x is a decrease by $\geq B/2 - 1$, so the change in Π is a decrease of $B/2 - 2 = \Omega(B)$, paying for the time spent on (ii).

Referring the reader to [18, section 2.2] for further details, this completes the proof of Lemma 1.7.

REFERENCES

[1] R.K. Ahuja, K. Mehlhorn, J.B. Orlin, and R.E. Tarjan, *Faster algorithms for the shortest path problem*, J. ACM, 37 (1990), pp. 213–223.

[2] S. Albers and T. Hagerup, *Improved parallel integer sorting without concurrent writing*, Inform. and Comput., 136 (1997), pp. 25–51.
[3] A. Andersson, T. Hagerup, S. Nilsson, and R. Raman, *Sorting in linear time?*, J. Comput. System Sci., 57 (1998), pp. 74–93.
[4] A. Andersson, P.B. Miltersen, S. Riis, and M. Thorup, *Static dictionaries on AC^0 RAMs: Query time $\Theta(\sqrt{\log n/\log\log n})$ is necessary and sufficient*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 441–450.
[5] A. Andersson, P.B. Miltersen, and M. Thorup, *Fusion trees can be implemented with AC^0 operations only*, Theoret. Comput. Sci., 215 (1999), pp. 337–344.
[6] A. Andersson and S. Nilsson, *A new efficient radix sort*, in Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 714–721.
[7] A. Andersson and M. Thorup, *Tight(er) worst-case bounds on dynamic searching and priority queues*, in Proceedings of the 32nd ACM Symposium on the Theory of Computing, Portland, OR, 2000, to appear.
[8] P. Beame and F. Fich, *Optimal bounds for the predecessor problem*, in Proceedings of the 31st ACM Symposium on the Theory of Computing, Atlanta, GA, 1999, pp. 295–304.
[9] P. Beame and J. Håstad, *Optimal bounds on the decision problems on the CRCW PRAM*, J. ACM, 36 (1989), pp. 643–670.
[10] J. Bentley, *Programming Pearls*, Addison-Wesley, Reading, MA, 1986.
[11] J.L. Carter and M.N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.
[12] B.V. Cherkassky, A.V. Goldberg, and C. Silverstein, *Buckets, heaps, lists, and monotone priority queues*, in Proceedings of the 8th ACM-SIAM Symposium on Discrete Algoritms, New Orleans, LA, 1997, pp. 83–92.
[13] E.W. Dijkstra, *A note on two problems in connection with graphs*, Numer. Math., 1 (1959), pp. 269–271.
[14] D.V. Denardo and B.L. Fox, *Shortest-route methods* 1: *Reaching, pruning, and buckets*, Oper. Res., 27 (1979), pp. 161–186.
[15] M.J. Fischer and M.S. Paterson, *Fishspear: A priority queue algorithm*, J. ACM, 41 (1994), pp. 3–30.
[16] E. Fredkin, *Trie memory*, Comm. ACM, 3 (1960), pp. 490–499.
[17] M.L. Fredman and D.E. Willard, *Surpassing the information theoretic bound with fusion trees*, J. Comput. System Sci., 47 (1993), pp. 424–436. See also the Proceedings of the 22nd ACM Symposium on the Theory of Computing, Baltimore, MD, 1990, pp. 1–7.
[18] M.L. Fredman and D.E. Willard, *Trans-dichotomous algorithms for minimum spanning trees and shortest paths*, J. Comput. System Sci., 48 (1994), pp. 533–551. See also the Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 719–725.
[19] M.L. Fredman and R.E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. ACM, 34 (1987) pp. 596–615.
[20] T. Hagerup, *Sorting and searching on the Word RAM*, in Proceedings of the 15th Symposium on the Theoretical Aspects of Computer Science, Paris, Lecture Notes in Comput. Sci. 1373, Springer, Berlin, 1998, pp. 366–398.
[21] E.J. Isaac and R.C. Singleton, *Sorting by address calculation*, J. ACM, 3 (1956), pp. 169–174.
[22] D.B. Johnson, *A priority queue in which initialization and queue operations take $O(\log\log D)$ time*, Math. Systems Theory, 15 (1982), pp. 295–309.
[23] T.C. Hu and A.C. Tucker, *Optimal computer search trees and variable-length alphabetic codes*, SIAM J. Appl. Math., 21 (1971), pp. 514–532.
[24] D. Kirkpatrick and S. Reisch, *Upper bounds for sorting integers on random access machines*, Theoret. Comput. Sci., 28 (1984), pp. 263–276.
[25] D.E. Knuth, *The Art of Computer Programming* 3: *Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
[26] K. Mehlhorn, *Data Structures and Algorithms* 1: *Sorting and Searching*, Springer, Berlin, 1984.
[27] K. Mehlhorn and S. Näher, *Bounded ordered dictionaries in $O(\log\log N)$ time and $O(n)$ space*, Inform. Process. Lett., 35 (1990), pp. 183–189.
[28] P.B. Miltersen, *Lower bounds for union-split-find related problems on random access machines*, in Proceedings of the 26th ACM Symposium on the Theory of Computing, Montreal, Canada, 1994, pp. 625–634.
[29] W.J. Paul and J. Simon, *Decision trees and random access machines*, in Proceedings of the

Symposium über Logik and Algoritmik, Zürich, Switzerland, 1980, pp. 331–340.
[30] R. Raman, *Priority queues: small, monotone, and trans-dichotomous*, in Proceedings of the European Symposium on Algorithms, Barcelona, Spain, Lecture Notes in Comput. Sci. 1136, Springer, Berlin, 1996, pp. 121–137.
[31] R. Raman, *Recent results on the single-source shortest paths problem*, SICACT News, 28 (1997), pp. 81–87.
[32] M. Thorup, *On RAM priority queues*, in Proceedings of the 7th ACM-SIAM Symposium on Discrete Algoritms, Atlanta, GA, 1996, pp. 59–67.
[33] M. Thorup, *Randomized sorting in $O(n \log\log n)$ time and linear space using addition, shifts, and bit-wise Boolean operations*, in Proceedings of the 8th ACM-SIAM Symposium on Discrete Algoritms, New Orleans, LA, 1997, pp. 352–359.
[34] M. Thorup, *Faster deterministic sorting and priority queues in linear space*, in Proceedings of the 9th ACM-SIAM Symposium on Discrete Algoritms, Baltimore, MD, 1998, pp. 550–555.
[35] P. van Emde Boas, *Preserving order in a forest in less than logarithmic time and linear space*, Inform. Process. Lett., 6 (1977), pp. 80–82.
[36] P. van Emde Boas, R. Kaas, and E. Zijlstra, *Design and implementation of an efficient priority queue*, Math. Systems Theory, 10 (1977), pp. 99–127.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 110–144

ROBOT NAVIGATION WITH DISTANCE QUERIES*

DANA ANGLUIN†, JEFFERY WESTBROOK‡, AND WENHONG ZHU§

Abstract. We consider the problem of online robot navigation in an unfamiliar two-dimensional environment, using comparatively limited sensing information. In particular, the robot has local sensors to detect the proximity of obstacles and permit boundary-following, and it is able to determine its current distance and relative bearing to its final destination (via distance queries). By contrast, most previous algorithms for online navigation have assumed that the robot knows its exact current position. Because determining exact location is prone to error that accumulates over time, the usefulness of such algorithms may be limited. In contrast, distance queries give less information, but the accuracy of each query is independent of the number of queries, which means distance queries can be more robust. We formally define our model and give new, efficient navigation algorithms and lower bounds for this setting.

Key words. robot navigation, range query, distance query, robot exploration, analysis of algorithms

AMS subject classification. 68T05

PII. S0097539797330057

1. Introduction.

We consider the problem of navigating a robot from its initial position to a target destination in a two-dimensional (2D) environment partially obstructed by initially unknown obstacles of arbitrary shapes. We assume that the robot is equipped with two kinds of abstract sensors: (1) local sensors, which allow it to detect the proximity of an obstacle and travel along the boundary of an obstacle, and (2) range sensors, which allow it to determine the distance of its current position from the target destination and its relative bearing to the target.

Previous theoretical work on navigation has generally assumed that the robot has access to the (x, y) coordinates of both the target destination and the robot's current position. In practice, an estimate of the robot's current position is usually obtained by odometry, i.e., by integrating a speed estimate over time. This means that error in the position estimate steadily accumulates as the robot travels. The difficulty of determining exact position has recently led robotics researchers to look for navigation algorithms that do not need exact global position but use weaker information that can be determined without cumulative error. In this connection, Taylor and Kriegman [8] introduced the idea of distance queries, based on the capabilities of their robot, RJ.

The general question of how much and what kinds of sensory information a robot needs for various tasks constitutes an exciting area of research; the main contribu-

*Received by the editors November 14, 1997; accepted for publication (in revised form) April 27, 1999; published electronically April 25, 2000. A preliminary version of this paper appeared in *Proceedings of the 28th ACM Symposium on the Theory of Computing*, Philadelphia, 1996.

http://www.siam.org/journals/sicomp/30-1/33005.html

†Department of Computer Science, Yale University, P.O. Box 208285, New Haven, CT 06520-8285 (dana.angluin@yale.edu). This research was partially supported by National Science Foundation grants CCR-9213881 and CCR-9610295.

‡AT&T Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 09732-0971 (jeffw@research.att.com). This research was done while this author was at the Department of Computer Science, Yale University.

§Amiable Technologies, Inc., International Plaza Two, Suite 625, Philadelphia, PA 19113 (wzhu@amiableworld.com). This research was done while this author was at the Department of Computer Science, Yale University, and was partially supported by ARPA/ONR contract N00014-93-1-1235.

tion of this paper is to explore the implications for online navigation of replacing the assumption of exact position by the weaker assumption of distance queries. Another contribution of this paper is the definition of our model, the distance query model, which we now describe sufficiently to state our results. (More details appear in section 3).

The boundary of an obstacle is represented as a simple closed curve in R^2 of finite length, with two additional restrictions given below. Each obstacle has an *obstructed region*, which for *normal obstacles* is the interior and for a *room wall* is the exterior. This allows us to model navigation inside a closed room. An obstacle is the union of its boundary and its obstructed region.

A *scene* consists of a nonzero finite number of nonintersecting obstacles, a start point S, and a target point T. *Free space*, the set of points that a robot may reach, consists of all points not in the obstructed region of any obstacle. The boundaries of obstacles are in free space. *Open space* consists of all points not contained in obstacles. The boundaries of obstacles are not in open space. S is assumed to be in free space, but T may be in free space (*reachable*) or not (*unreachable.*) We distinguish between two situations: the *indoor setting,* in which the scene may contain a room wall, and the *outdoor setting*, in which there is no room wall and the robot is effectively located on an unbounded plane.

We place two restrictions on obstacles. The first is that any line segment intersects a given obstacle boundary in a finite number of points and segments. The second is that a clockwise (or counterclockwise) traversal of an obstacle boundary consists of a finite number of maximal segments, each of which is of monotonically increasing, monotonically decreasing, or constant distance to the target. These segments intersect only at their endpoints. The points at which distinct segments intersect are *break points*. The number of these segments is the *complexity* of the obstacle. The parameter n is the maximum complexity of any obstacle in the scene; n is not known to the robot but is used in quantifying the performance of our navigation algorithms.

The robot is modeled as a point. The sensing and action capabilities of the robot are precisely defined in section 3. Informally speaking, there are two actions available to the robot: (1) It may travel toward T through open space until it reaches T or hits an obstacle boundary. When the robot hits an obstacle boundary, it learns its distance to T. (2) It may follow an obstacle boundary in clockwise or counterclockwise direction until it reaches T or a break point. At a break point, the robot learns its distance to the target and whether the directions toward and away from T are blocked. If the direction to T is unblocked, the robot may choose action (1) or (2). Otherwise, it may choose action (2) to continue traversing the boundary of the obstacle in either direction.

We consider four general kinds of problems related to navigation; in each, the robot is started at S, executes some sequence of legal actions, and must eventually halt.

- Target reachability. The robot must correctly declare T reachable or not.
- Circumnavigation. S is on the boundary of an obstacle, and the robot is required to visit every point on that boundary.
- General navigation. The robot must halt at T if it is reachable; otherwise it must halt and declare T to be unreachable. Clearly a solution of the general navigation problem also solves the target reachability problem.
- Reachable-target navigation. T is in free space. The robot must halt at T.

We measure the performance of a navigation algorithm by comparing the distance

traveled by the robot to the sum of the perimeters of the obstacles in the scene. In particular, the *excess distance* traveled by a robot in a scene is the length of its path minus the straight-line distance from S to T. The *excess distance ratio* of a navigation algorithm *with respect to a scene* is the ratio of the excess distance traveled by the robot to the sum of the perimeters of the obstacles in the scene. This ratio measures how much the navigation algorithm has to retrace the boundaries of obstacles in the course of getting from S to T.

The *excess distance ratio* of a navigation algorithm is $f(n)$ if, for every n, the supremum over all scenes of complexity n of the excess distance ratios is $f(n)$. Lumelsky and Stepanov give a construction that implies the following result for the distance query model.

THEOREM 1.1 (Lumelsky and Stepanov [4]). *For any deterministic algorithm for reachable-target navigation in the outdoor setting, the excess distance ratio is bounded below by* 1 *for all* $n \geq 4$.

We present the following new results for the distance query model.

1. If T may be unreachable, then no deterministic algorithm can solve either the target reachability, the circumnavigation, or the general navigation problems.

2. There is a deterministic algorithm for reachable-target navigation in the indoor setting with an excess distance ratio of $O(\log n / \log \log n)$.

3. There is a matching lower bound of $\Omega(\log n / \log \log n)$ for a restricted class of deterministic algorithms for reachable-target navigation in the indoor setting.

4. There is a deterministic algorithm for reachable-target navigation in the outdoor setting with an excess distance ratio bounded by 3. (Compare the lower bound of 1 for this problem.) A variant of this algorithm solves the circumnavigation problem in the outdoor setting when the target is reachable.

Our navigation and circumnavigation algorithms for reachable-target navigation can be used in the general case, but they will fail to halt if the target is unreachable. It is interesting to note that our algorithms will still work if the distance queries return not the exact distance to the target, but values that preserve the linear ordering on distances. This is potentially useful in practice, as discussed below.

2. Related results. Previous theoretical research on online robot navigation has progressed in at least two research communities: the online competitive analysis community [1, 2, 6] and the theoretical robotics community [3, 4, 5, 8]. The competitive analysis community has generally considered various restricted types of polygonal obstacles and has measured the performance of a navigation algorithm by the ratio of the distance traveled by a robot using that algorithm to the length of the shortest obstacle-avoiding path from S to T. The theoretical robotics community has considered obstacles of essentially arbitrary shape and measured the performance of a navigation algorithm by the excess distance ratio, defined above. Since our work continues this latter line of research we now briefly sketch the background.

Lumelsky and Stepanov [3, 4] began this line of research, giving navigation algorithms for a robot with local sensors and access to its exact current position. Their algorithm, BUG1, navigates directly toward T in free space until an obstacle boundary is detected. It then follows the obstacle boundary until it circumnavigates the obstacle, keeping track of the minimum distance to T among points on the boundary. When the robot completes the circumnavigation, it follows the boundary back to one of the points at minimum distance to T and resumes navigating directly toward T in free space.

As described, BUG1 achieves an excess distance ratio bounded by 2 for general

navigation in the outdoor setting in Lumelsky and Stepanov's setting. Assuming the additional capability of keeping track of path lengths, a refinement of BUG1 achieves an excess distance ratio bounded by 3/2, by taking the shorter direction on the boundary back to the departure point. Lumelsky and Stepanov also give a lower bound of 1 on the excess distance ratio for deterministic navigation algorithms in their setting, as mentioned above.[1]

Taylor and Kriegman [8], addressing the practical difficulty of determining a robot's exact position, consider a model in which the robot's global sensors can determine only distance and relative bearing to the target.[2] On their experimental mobile robot, RJ, this information is provided by a vision system which tracks the target as the robot moves. The vision system gives quite accurate distance data using known properties of the target, and the error in the answers is not cumulative. The vision system can give a linear ordering of points by distance simply by comparing the size of the target image at each point. The local sensors are implemented primarily with sonar, which gives a localized map of the free space around the robot.

Taylor and Kriegman give an adaptation of Lumelsky and Stepanov's BUG1 algorithm for this setting. Their algorithm, however, requires a subroutine to solve the circumnavigation problem, and they leave open the question of whether such a subroutine can actually be implemented with local sensors and distance queries. Assuming that the subroutine both exists and traverses the boundary of an obstacle exactly once, Taylor and Kriegman show that their algorithm achieves an excess distance ratio of 2 for general navigation in open space. They use the navigation algorithm as a subroutine in an exploration procedure to find all the recognizable targets in the environment.[3]

The existence of a circumnavigation subroutine is crucial to Taylor and Kriegman's version of BUG1. One goal of our paper is to answer some of the open problems implied by their work: Under what conditions do local sensors and distance queries suffice to solve the circumnavigation problem? When they don't suffice, and hence Taylor and Kriegman's algorithm is unusable, is there some other solution to general navigation with distance queries?

We remark that if the robot has an accurate compass in addition to distance data, then exact position can be recovered. Magnetic compasses are subject to distortions inside buildings due to metals in the construction, and inertial compasses are quite expensive. Therefore, the use of compasses is generally considered impractical for indoor navigation. Nevertheless, Lumelsky and Tiwari [5] examined the use of pure directional information, assuming a robot with an on-board compass and relative bearing to the target in addition to local sensors but no distance data. They give a navigation algorithm for this setting with an $\Omega(n)$ excess distance ratio.

3. The distance query model. We propose a model of navigation in the plane in which an obstacle is defined as the interior or exterior of a simple closed path with some mild restrictions. We hope that the model is intuitive, but we also provide a careful theoretical development of its properties because we discovered subtle pitfalls

[1] Their claim that it holds also for randomized algorithms is not proved.

[2] Differential GPS gives (x, y) position without accumulated error. However, Taylor and Kriegman state that indoor differential GPS accuracy is still insufficient for typical robot navigation problems.

[3] Their model also permits some obstacles to obscure the view of the target, so that in certain regions of space no distance data is available. Their navigation algorithm still works in this setting, under additional assumptions about the regions of obscuration. Our algorithms for navigation within a room can also be modified to work in this setting. See section 8 below for further discussion.

and problems in our earlier attempts to formalize the model, particularly in proving Theorems 4.1 and 7.4.

The environment is an infinite 2D plane, R^2. We follow Newman's treatment of the topology of the plane [7]. Recall that S and T denote the start and target positions, respectively. Without loss of generality, we may assume the target T is at the origin. For any two distinct points P and Q in R^2, PQ denotes the line segment connecting P and Q, and $|PQ|$ denotes the distance between P and Q.

DEFINITION 3.1. *A* path *in R^2 is a continuous mapping p of some closed interval $[0, x] \in R^1$ to R^2. A* closed *path is a path such that $p(0) = p(x)$. A* simple closed *path is a closed path that intersects itself only at $p(0) = p(x)$. A simple closed path is also called a Jordan curve.*

It is convenient to use polar coordinates to specify paths, with the origin at the target T and a particular ray from T specified for the measurement of angles. Path p is specified by the pair of functions $(r(t), \theta(t))$, $t \in [0, x]$, where $r(t)$ is the distance of point $p(t)$ from the origin and $\theta(t)$ is its counterclockwise angular displacement in radians from the specified ray. The angular displacement of any point but the origin is unique modulo 2π, but since we wish to assume that $\theta(t)$ is a continuous function of t, we permit $\theta(t)$ to be outside the interval $[0, 2\pi)$. We use the notation $(r_1, \theta_1) \equiv (r_2, \theta_2)$ to indicate that (r_1, θ_1) and (r_2, θ_2) denote the same point in the plane.

3.1. Obstacles and scenes.

DEFINITION 3.2. *An* obstacle boundary *is a simple closed path of finite length $\beta : [0, 1] \to R^2$. (We specify some additional conditions on an obstacle boundary below.) By Jordan's theorem, β divides the plane into two domains, one bounded and the other unbounded (containing a point arbitrarily far from the origin). The bounded region is called the* interior *and the unbounded region is called the* exterior. *Note that Definition* 3.1 *implies a direction on an obstacle boundary, given by increasing t. We will assume that the boundary curve $\beta = (r(t), \theta(t))$ is parameterized so that increasing t corresponds to a counterclockwise traversal of the boundary path.*

DEFINITION 3.3. *An* obstacle *is a region $Ob \subset R^2$ consisting of an obstacle boundary together with an* obstructed region, obs(Ob), *which is either the interior, in the case of a* normal obstacle, *or the exterior, in the case of a* room wall. *P_{Ob} denotes the length of the boundary of Ob.*

Our definition of obstacle precludes holes without loss of generality. In an actual application the boundary may come from a physical object that contains arbitrarily many holes, but since the robot cannot physically reach any part of the region inside the obstacle boundary, it cannot distinguish this from an obstacle without holes.

DEFINITION 3.4. *A* (proper) boundary segment *of an obstacle with boundary $\beta : [0, 1] \to R^2$ is defined to be $\beta([a, b])$ for any nonempty interval $[a, b] \subset [0, 1]$, or $\beta([a, 1]) \cup \beta([0, b])$, where $0 < b < a < 1$.*

DEFINITION 3.5. *Let $\mathcal{O}$ denote a set of nonintersecting obstacles. The* obstructed space *with respect to $\mathcal{O}$ is the union of the obstructed regions of all the obstacles in $\mathcal{O}$. The* free space *with respect to $\mathcal{O}$ is the complement of the obstructed space with respect to $\mathcal{O}$. Free space therefore includes the obstacle boundaries. The* open space *with respect to $\mathcal{O}$ is the complement of $\mathcal{O}$. Open space therefore excludes the obstacle boundaries.*

DEFINITION 3.6. *A* scene *consists of a finite set of nonintersecting obstacles $\mathcal{O}$, a start position S in free space, and a target position T that may be located anywhere, including in the obstructed region of an obstacle. Note that a scene may contain at*

most one room wall.

Restrictions on obstacles. As we have indicated, the boundary of an obstacle is a simple closed curve of finite length with some additional restrictions. Previous papers have generally assumed obstacle boundaries to be piecewise smooth. We instead require the following two conditions, which are primarily intended to keep the boundaries sufficiently well behaved to allow us to speak reasonably of an "algorithm."

1. The intersection between an obstacle boundary and a line segment can be expressed as the union of a finite number of closed intervals. (A point is a closed interval of length zero.)

2. In any scene, each obstacle boundary can be divided into a finite number of maximal segments such that each of them is of monotonically increasing, monotonically decreasing, or constant distance from T. The segments intersect only at their endpoints.

DEFINITION 3.7. *A* local minimum *or* maximum *of distance to T on a boundary segment is a point closest to or furthest from T, respectively, within some small neighborhood of the point.*

DEFINITION 3.8. *Each point where distinct segments of increasing, decreasing, or constant distance intersect is called a* break point. *A break point is a local minimum or maximum, depending on the precise types of intersecting segments.*

DEFINITION 3.9. *The* complexity *of the obstacle is the number of break points on its boundary.*

LEMMA 3.10. *Suppose P and Q are distinct points in an arbitrary scene. The line segment PQ can be partitioned into a finite sequence of maximal intervals, each of which is contained entirely in one of*

1. *the obstructed region of exactly one obstacle,*
2. *the boundary of exactly one obstacle, or*
3. *open space.*

Proof. Let a segment PQ be given. By the conditions on obstacle boundaries the intersection of PQ and each obstacle boundary is the union of a finite number of closed intervals. Since there are finitely many obstacles, and they are pairwise disjoint, the intersection of PQ and all the obstacle boundaries is the union of a finite number of closed intervals. If we consider the endpoints of these intervals in order along the segment from P to Q, the open intervals between two consecutive endpoints must be contained in the boundary of a single obstacle or contained in open space. The lemma follows. □

Note that in the relative topology of the segment PQ, intervals of types (1) and (3) are open, and intervals of type (2) are closed. Hence intervals of types (1) and (3) can be adjacent to only intervals of type (2) and vice versa.

DEFINITION 3.11. *In the sequence of intervals just described, a* boundary crossing *is a triple of intervals of types* (1), (2), *and* (3) *in sequence, or* (3), (2), *and* (1) *in sequence. That is, there is a transition from the obstructed region of an obstacle to open space, or vice versa, crossing the boundary of that obstacle. Note that because of the alternation required, a segment with both ends in obstructed space (or both ends in open space) must have an even number of boundary crossings.*

DEFINITION 3.12. *For points $P \neq Q$, the* PQ-probe *is the subsegment PQ' of PQ, obtained from the partition described above by starting at P and taking the first interval if it is of nonzero length or by taking the union of the first two intervals if the first one is of length zero (that is, just a single point).*

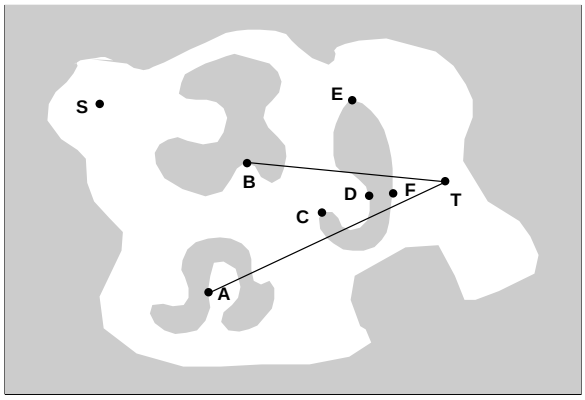


FIG. 3.1. *A scene with a room wall and three normal obstacles. Point A is blocked-away and unblocked-toward. B is both blocked-toward and blocked-away. C, D, E, and F are all break points: C and E are local maxima, D and F are local minima, and F is the closest point of the obstacle to T.*

We note that the PQ-probe is contained either in free space or (except possibly for the point P) in the obstructed region of one obstacle. This is clear if the first interval in sequence from P is of nonzero length; otherwise, the point P must be on the boundary of an obstacle and the following interval is contained either in the obstructed region of that obstacle or in open space. If the PQ-probe is contained in free space, we say it is *free*; otherwise, we say it is *obstructed*. The PQ-probe is thus an idealization of a sensor probe which determines if a robot at point P has an unobstructed path starting in the direction of Q.

DEFINITION 3.13. *A point $P \neq T$ is* unblocked-towards *T if the PT-probe is free. Point P is* blocked-towards *T if the PT-probe is obstructed.*

DEFINITION 3.14. *Let $P \neq T$ and let $Q \neq P$ be any point such that $P \in QT$. P is* unblocked-away *from T if the PQ-probe is free. Point P is* blocked-away *from T if the PQ-probe is obstructed.*

Informally, the robot may move toward the target from a point that is unblocked-towards, and it may move away from the target from a point that is unblocked-away. Although none of our algorithms moves directly away from T, the open space navigation algorithm in section 7 uses the information of whether that direction is blocked or unblocked. Figure 3.1 gives examples of the definitions.

3.2. The robot.

DEFINITION 3.15. *A* robot *is a point-like automaton with the following acting and sensing capabilities:*

1. *move-to-T*(): Move directly along a ray toward T through free space until the robot reaches T or arrives at a point that is blocked-towards. In the latter case, the robot is said to collide with the obstacle. This function returns a ground track entry, defined below.

2. *follow-boundary(DIR)*: Follow the boundary of an obstacle in a clockwise or counterclockwise direction, according to DIR, until the robot reaches T or a break point. DIR is "RIGHT" or "CCW" to indicate a counterclockwise direction and "LEFT" or "CW" to indicate a clockwise traversal. This function returns a ground track entry.

A point at which the robot collides with an obstacle is called a *hitpoint*. By definition, a hitpoint must be blocked-towards. The robot's *ground track* is the sequence of ground track entries generated during its travels. A *ground track entry* contains the following:

1. the distance to the target from the current hitpoint or break point;
2. a field indicating whether the entry was generated by a hitpoint, a local maximum break point; a local minimum break point, or the start point S;
3. a bit indicating whether the point is blocked-towards or unblocked-towards;
4. a bit indicating whether the point is blocked-away or unblocked-away.

Our model is a particular choice of discretization of the control of the robot for navigation tasks. We assume a lower-level reactive control system using the sensors and actuators of the robot that enable it to travel toward the target in free space, to detect the proximity of an obstacle, and to follow the boundary of an obstacle in either direction to a local minimum or maximum of distance to the target. After colliding with an obstacle, the robot can turn right to follow the boundary in a counterclockwise direction or turn left to follow clockwise. Since there may be segments of the boundary of constant distance to the target, we assume that the robot will traverse these to their endpoints (that is, to break points.)

At a hitpoint or break point, the robot receives information about its current situation (summarized as a ground track entry) and decides upon its next action. We assume that the global sensors give distance to the target and also the relative bearing of the target. Combining this with the local sensors, we assume the robot can detect whether the directions straight toward and away from the target are immediately obstructed by obstacles. In the case of Taylor and Kriegman's robot, RJ, the local sensors are implemented by sonar, which in effect gives a localized map of the free space around the robot. Thus, the information we assume to be available in a ground track entry is empirically reasonable.

At the start of an execution, the robot receives an initial ground track entry describing the start point S. Each subsequent ground track entry is associated with a particular point on the boundary of some obstacle in the scene. Multiple (indistinguishable) ground track entries may be associated with the same boundary point. (This happens, for example, if the robot circumnavigates an obstacle twice.)

Specification of the robot's path. Let $rp(k)$ denote the path traveled by the robot up to the kth ground track entry. The path is specified as $rp(k) = (\rho_k(t), \vartheta_k(t))$, $t \in [0,1]$, where $\rho_k(t)$ and $\vartheta_k(t)$ are functions defining the polar coordinates of the points on the robot path. The parameter t is a nondecreasing function of real time, so that as t increases the robot's path is traversed in the forward direction. The actual time taken by the robot and the speed at which the robot travels are irrelevant, since we do not assume that the robot has sensors to measure these quantities.

Path $rp(0)$ consists only of the start point, S. For $k > 0$, path $rp(k-1)$ is a prefix of path $rp(k)$; path $rp(k-1)$ is extended to $rp(k)$ as follows. Let P_{k-1} be the point that generated ground track entry $k-1$. Either $P_{k-1} \equiv S$ or $P_{k-1} \equiv \beta(s)$, where β is the boundary of some obstacle and $s \in [0,1]$. We determine the subpath path, p', followed by the robot from P_{k-1} to the next hitpoint or break point. If there is no such next hitpoint or break point, then the robot does not halt and $rp(k)$ is undefined. Otherwise, path p' is appended to $rp(k-1)$, and the result is reparameterized, giving $rp(k)$. The subpath p depends on the action of the robot after receiving ground track entry $k-1$.

Case 1. The robot executes *follow-boundary(CCW).* Let s' be the minimum value

such that $s \leq s' \leq 1$ and $\beta(s')$ is a break point. Then $p' = \beta([s, s'])$. If there is no such s', then let s' be minimum such that $0 \leq s' \leq s$ and $\beta(s')$ is a break point and let $p' = \beta([s, 1]) \cup \beta([0, s'])$. If there is still no such s', then $rp(k)$ is undefined.

Case 2. The robot executes *follow-boundary(CW)*. Let s' be maximum such that $s \geq s' \geq 0$ and $\beta(s')$ is a break point. Then $p' = \beta([s', s])$. If there is no such s', then let s' be maximum such that $s \leq s' \leq 1$ and $\beta(s')$ is a break point and let $p' = \beta([s', 1]) \cup \beta([0, s])$. If there is still no such s', then $rp(k)$ is undefined. The path p' must be reparameterized so that $p'(0) = \beta(s)$ and $p'(1) = \beta(s')$.

Case 3. The robot executes *move-to-T*(). If the robot is blocked-towards, then $p' = \emptyset$ and $rp(k) = rp(k-1)$. Otherwise, p' is the segment of the inbound ray from P_{k-1} to the next hitpoint reached (or to T if the robot reaches T first).

3.3. Properties of scenes. In this section, some basic properties of scenes are established.

PROPERTY 3.16. *The distance between any two obstacles is positive.*

Proof. This follows because the boundaries of obstacles are nonintersecting closed and bounded sets. □

PROPERTY 3.17. *Other than T, every point is either unblocked-towards or blocked-towards, and every point is either unblocked-away or blocked-away.*

Proof. If $P \neq T$, then the PT-probe is either free or obstructed (see the remark following Definition 3.12). Hence P is either unblocked-towards or blocked-towards but not both. A similar argument applies to the blocked/unblocked-away case. □

PROPERTY 3.18. *Let Ob be an obstacle with boundary β. Either* (a) *all points on β are equidistant from T or* (b) *β contains at least two break points: one at a point at maximum distance from T and one at a point at minimum distance from T.*

Proof. Suppose (a) is false, so that β contains two points P, Q such that $|PT| < |QT|$. Then there are distinct points P' and Q' on β at minimum and maximum distance from T, respectively. Either P' is an endpoint of a maximal segment of monotonically decreasing distance, in which case it is a break point by definition, or it is in the middle of a segment of constant distance, in which case either endpoint of the segment is a break point. A similar argument establishes the existence of a break point at maximum distance from T. □

PROPERTY 3.19. *Let Ob be an obstacle not containing T with boundary β. Any point P on β at minimum distance from T is unblocked-towards.*

Proof. If P is on the boundary of an obstacle Ob, then the PT-probe is contained either in the obstructed region of Ob or in open space. The former is impossible, since otherwise PT must intersect Ob at a point besides P, implying P is not at minimum distance to T. Thus, the PT-probe is free, implying that P is unblocked-towards. □

PROPERTY 3.20. *Let Ob be a normal obstacle with boundary β. Any point P on β at maximum distance from T is unblocked-away.*

Proof. An argument analogous to that for Property 3.19 suffices. □

PROPERTY 3.21. *If the boundary of an obstacle Ob not containing T has a point P that is blocked-towards, then it has a break point.*

Proof. Since P is blocked-towards, the PT-probe, PT', must be contained in the obstructed region of Ob except for P. Hence $T'T$ must intersect the boundary of Ob at a point Q closer to T than P. Hence the boundary of Ob contains two points at different distances from T, and we apply Property 3.18. □

PROPERTY 3.22. *Let Ob be a normal obstacle not containing T. Let P be a break point on the boundary of Ob that is unblocked-towards and a local minimum. Then*

the PT-probe is contained in open space except for the point P.

Proof. Because P is unblocked-towards, the PT-probe is contained in free space and must be either exterior to Ob except for the point P or contained in the boundary of Ob, which means that there are points of the boundary of Ob arbitrarily close to P that are closer to T than P. However, this would contradict the assumption that P is a local minimum of distance to T. □

PROPERTY 3.23. *Let Ob be a normal obstacle not containing T, and let P be a break point on the boundary of Ob that is unblocked-away and a local maximum of distance to T. Let $Q \neq P$ be any point such that P is in the segment QT. Then the PQ-probe is contained in open space except for the point P.*

Proof. The proof is analogous to the proof of Property 3.22. □

4. The limits of distance queries. In this section we show that the circumnavigation, target reachability, and general navigation problems cannot be solved by deterministic algorithms. This shows that distance information is strictly weaker than exact position information, because all these problems can be solved deterministically if exact position information is available [3, 4].

These negative results are derived from a theorem, proved below, which states that there is no deterministic algorithm which will successfully circumnavigate every obstacle that contains the target in its interior. There is one obstacle that defeats every algorithm: a circular obstacle boundary centered on the target. Once a robot hits this obstacle and starts to follow the boundary in any direction, it never encounters a break point, or indeed any point distinguishable from the hitpoint, and so never halts. A similar statement is true for a circular room wall centered on T.

The case of a circle centered at T, however, is uniquely pathological in having no break points; all other obstacle boundaries have at least two break points. One might hypothesize therefore that navigation is possible as long as there are no such pathological obstacles. We will show, however, that even if the obstacles are required to have an arbitrarily large number of break points, any deterministic algorithm will fail on some obstacle.

The intuition behind this result is as follows. Let Ob be some normal obstacle of the desired complexity with T in its interior. Let $\beta = (r(t), \theta(t))$ be the boundary of Ob, $t \in [0, 1]$, where the angle is taken with respect to T, and $\theta(0) = 0$. Let A be a deterministic circumnavigation algorithm, used to control a robot that is started at position $(r(0), \theta(0)) = (x, 0)$. If A fails to circumnavigate Ob, then the desired result is immediate. Suppose A circumnavigates Ob exactly once. Now consider the curve $\beta^* = (r(t), \theta(t)/2)$, which starts at angle 0 and stops at angle π. Since scaling the angle does not affect distance, the curve β^* contains the same sequence of local minima and maxima, at exactly the same distances from T and with exactly the same blocked/unblocked status, as the entire boundary of Ob. Therefore, if the robot is started on β^* at $(r(0), \theta(0)/2) = (x, 0)$, it will see exactly the same sequence of ground track entries and will halt upon reaching point $t = 1$. We can close β^* in any way (in particular, with another copy of β^*) to get an obstacle Ob^* which A will thus fail to circumnavigate.

In general, the robot may go around the obstacle more than once, in one or both directions. This can be dealt with by scaling the obstacle boundary more strenuously and joining several scaled copies of it in a circle around T. Figure 4.1 shows an example spiral-like obstacle and Figure 4.2 shows how six scaled copies of this obstacle are combined. Thus, an algorithm that made five counterclockwise circumnavigations of the obstacle in Figure 4.1 and then halted would not complete a circumnavigation

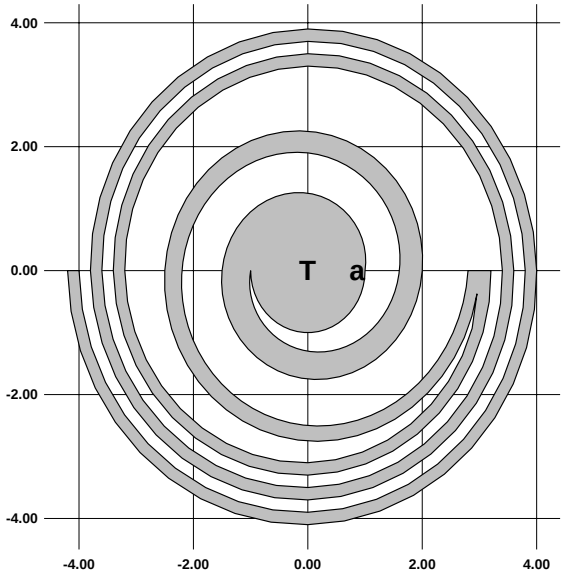


FIG. 4.1. *A spiral-like obstacle Ob. The target T is at* $(0,0)$. *Letting* β *denote the boundary curve,* $a = \beta(0) = \beta(1) = (1,0)$.

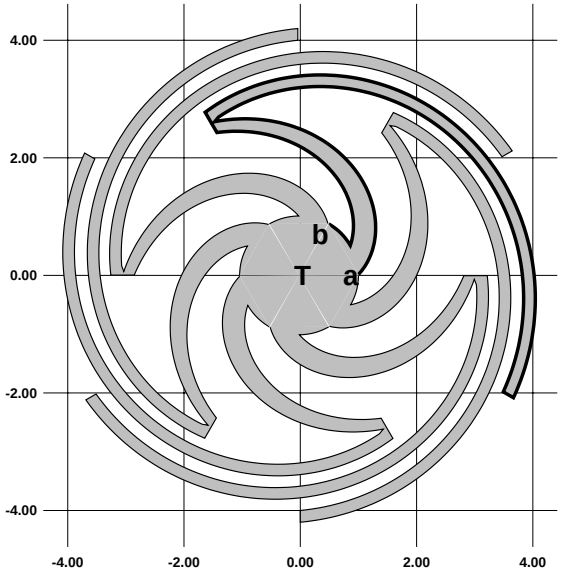


FIG. 4.2. *Obstacle* $Z(Ob, 6)$ *constructed from the obstacle Ob of Figure* 4.1. *(The function Z is defined precisely in the appendix.) The first scaled copy of the original boundary is shown in bold. Letting* β^* *denote this scaled curve, point* $a = \beta^*(0) = (1,0)$ *and* $b = \beta^*(1) = (1, \pi/3)$.

of the obstacle in Figure 4.2. In the appendix to this paper we formalize this transformation and prove the indistinguishability of the robot's ground track on the two obstacles, which leads to a proof of the following result.

THEOREM 4.1. *In the distance query model, if there may exist an obstacle whose interior contains the target (either a normal obstacle with the target in its obstructed region, or a room wall with the target in its complement), then there is no deterministic algorithm for circumnavigation. Furthermore, there are arbitrarily complex obstacles which any given deterministic algorithm fails to circumnavigate.*

COROLLARY 4.2. *In the distance query model, there is no deterministic algorithm for target reachability.*

Proof. We show that any algorithm A that solves target reachability must also solve circumnavigation. Consider an obstacle Ob containing the target T, and suppose we start A at some point on the boundary of T. Suppose A halts and declares T unreachable but does not circumnavigate Ob. Then there is some nonzero segment of the boundary of Ob, no point of which is traversed by A. Consider the obstacle Ob^* in which that segment is replaced by an arbitrarily narrow passage that leads to T. Since A never reaches any point on the segment, its behavior on Ob^* will be the same as on Ob, and in particular it will incorrectly declare the target unreachable. □

5. Target-reachable navigation inside a room. For the remainder of this paper we assume that the target is reachable. If the target is contained in the obstructed region of an obstacle, the algorithms we present will reach that obstacle but will not halt.

In any instance of the navigation problem, the robot alternates between moving toward T in free space and following an obstacle boundary searching for T or a *possible departure point*, that is, an unblocked-towards break point from which it can head back into free space. A point at which the robot heads back into free space is an *actual departure point*. In general, an obstacle may have many possible departure points, and making a good choice of actual departure points is the heart of a navigation algorithm in this setting.

One strategy is to search in a fixed direction for the first possible departure point and leave from there. A simple construction (adapted from Lumelsky and Stepanov) forces this strategy to cycle indefinitely, repeatedly departing from an obstacle and hitting it again. A variant requiring the actual departure point to be a local minimum also cycles.

For example, in Figure 5.1, if the robot starts in the center interior of the U-shaped obstacle, the nearest break points in either direction are the tips of the U. But departing from those points leads the robot right back to the center of the obstacle.

Cycling is prevented if the robot always leaves from a *monotone departure point*, that is, a possible departure point whose distance from T does not exceed the distance from T of the preceding hitpoint. To formalize this, we define the class MONOTONE of navigation algorithms that operate as follows. Initially the robot executes *move-to-T*(). If any action ever returns a ground track entry at distance zero from T, the robot halts, having reached T. Whenever *move-to-T*() returns a hitpoint, the robot uses some sequence of *follow-boundary(DIR)* operations to search the obstacle boundary, either halting at T or choosing some monotone departure point from which to execute *move-to-T*() and leave the obstacle.

LEMMA 5.1. *Regardless of how the actual departure points are chosen, any algorithm from the class MONOTONE reaches T after a finite number of actions.*

Proof. If the robot moves straight to T without colliding with any obstacle, then it reaches T. Suppose the robot collides with some obstacle. If T is not on the boundary, then Properties 3.18 and 3.20 show that there is at least one monotone departure point on the boundary, namely, a break point at the global minimum distance from T. If

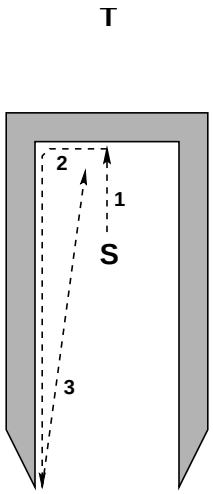


FIG. 5.1. *After hitpoint, turn left or right and leave at the nearest break point. The robot never escapes the cup.*

T is on the boundary, then either there is a monotone departure point or the robot must reach T.

Thus, if the robot never reaches T, it must do so by repeatedly colliding with obstacles that have monotone departure points and departing from them. Let $h_1, d_1, h_2, d_2, \ldots$ be the sequence of distance values of the robot's position at each hitpoint and subsequent actual departure point, ordered by time. By the definition of monotone departure point, $h_i \geq d_i$. On the other hand, by the definition of unblocked-towards, $d_i > h_{i+1}$. Hence the departure point distances form a monotonically decreasing sequence. This implies that the robot can depart from a specific break point at most once. But each obstacle has a finite number of break points, and there are a finite number of obstacles. □

We remark that if the robot starts on the boundary of a room wall consisting of a circle centered at T, then there are no break points, which is why the first action of the algorithm is to move toward T.

Perhaps the simplest algorithm in the class MONOTONE searches the boundary in a fixed direction for the first monotone departure point. The construction in Figure 5.2 shows that this strategy has excess distance ratio $\Omega(n)$.

On the other hand, a very good choice of a departure point is a break point at the global minimum distance to T among all boundary points; this is the choice made by Lumelsky and Stepanov's BUG1 algorithm. If a robot departs from an obstacle at such a point, it will never revisit the obstacle again, since any subsequent departure point is some positive distance closer to the target, and there is no such point on Ob. If the robot could solve the circumnavigation problem, it could guarantee to find and depart from such a point. However, the results of section 4 show that deterministic circumnavigation of a room wall is impossible. Hence the algorithms we develop in this section do not attempt to circumnavigate objects but rather try to limit the number of times an obstacle boundary will be revisited.

5.1. Algorithm 2-REPEATS. We now describe 2-REPEATS, our first navigation algorithm for the indoor setting. Initially the robot executes *move-to-T*().

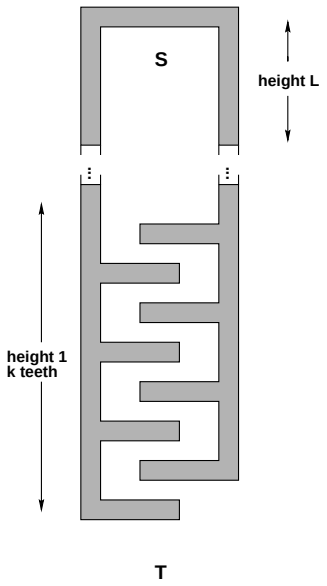


FIG. 5.2. *After each hitpoint, turn left and leave at the nearest monotone break point. (The distance L is $\Omega(k)$.) The robot escapes the comb after traveling completely around the inside of the upper loop $\Omega(k)$ times. The obstacle complexity $n = \Theta(k)$.*

If any action returns a ground track entry indicating that the robot has reached T, it halts. When the robot collides with an obstacle, it turns left and begins traveling clockwise around the boundary, saving its ground track from the hitpoint. If the robot has not already reached T, then as soon as its ground track forms a 2-*repeat departure sequence* the robot leaves the obstacle by executing *move-to-T*().

A 2-repeat departure sequence has the form

$$h\alpha_0 g_0 \alpha_1 g_1 \alpha_2 g_2,$$

where h is the hitpoint, g_i is a ground track entry, α_j is a possibly empty sequence of ground track entries, and $\|\alpha_0\| \leq \|\alpha_1\| = \|\alpha_2\|$, where $\|\alpha\|$ denotes the length of the sequence (number of break points). Each g_i, $0 \leq i \leq 2$, is a ground track entry for an unblocked-towards local minimum which is also the global minimum in the sequence. That is, $|Tg_i| = |Tg_j|$ for all $0 \leq i < j \leq 2$, and $|Tg_i| \leq |TQ|$ for all break points Q in the departure sequence.

LEMMA 5.2. *Let Ob be an obstacle and H any hitpoint at which the robot collides with Ob. As the robot travels clockwise around the boundary, either it will reach T or its ground track will eventually form a 2-departure sequence.*

Proof. By Properties 3.18 and 3.20, the existence of a hitpoint implies the existence of a break point G on the obstacle boundary at minimum distance to T. If the robot travels around the obstacle boundary for sufficiently long, it will either reach T or generate the 2-departure sequence in which $h = H$, $g_i = G$, $h\alpha_0$ is the ground track from H to G, and $\alpha_1 = \alpha_2$ is the ground track given in one complete circumnavigation of *Ob* starting and ending at G. □

Either the robot finds T or it leaves the obstacle from a point no farther from T than the most recent hitpoint. Therefore 2-REPEATS is in the class MONOTONE.

Lemma 5.2 provides some intuition for 2-REPEATS. If the robot sees the same minimum distance twice, then it may well have circumnavigated the obstacle and found the closest point to the target. In this case, the robot will leave and never return to this obstacle. Of course, it may be that a small segment of the boundary, far from the closest point, just happens to contain a departure sequence. The next lemma shows that by looking for the repetition $\alpha_1 g \alpha_2 g$, the robot avoids going over the same ground too many times.

LEMMA 5.3. *Let Ob be an obstacle of complexity n. Any particular break point P on the boundary is traversed $O(\log n)$ times over the total operation of finding a path from S to T.*

Proof. Define an *encounter with obstacle Ob* to be a particular collision with, search around, and departure from Ob by the robot. Since 2-REPEATS is monotone, the robot eventually halts at T (Lemma 5.1), so it encounters Ob a finite number of times. Between each encounter with Ob, the robot may collide with many other obstacles. On the other hand, if Ob is sufficiently complicated, the robot may depart from one point of Ob only to immediately collide with another arm of Ob.

Let $E_1, E_2, \ldots, E_k$ be the subsequence of encounters in which break point P is traversed. If it is traversed more than once in an encounter, then Ob has been circumnavigated, and, as remarked above, the robot will never collide with Ob again. Therefore, in encounter E_i, $i < k$, P is traversed exactly once. In the last encounter, E_k, P is traversed $O(1)$ times: at most three times if E_k circumnavigates Ob, or once otherwise. Hence P is traversed $O(k)$ times. We now bound k.

Let D_i denote the segment of the boundary that is traversed during E_i, and let n_i be the number of break points in D_i. In all encounters but the last there is a one-to-one correspondence between ground track entries in the departure sequence and break points of D_i, so we will not distinguish between them.

CLAIM 5.4. *For any i, j such that $j < i$, D_j contains none of the points h_i or $g_{i\ell}$, $0 \le \ell \le 2$.*

To verify this claim, observe that at each encounter, the hitpoint is strictly closer to T than the preceding departure point, so $|h_i T| < |g_{j2} T|$. Since $|g_{i\ell} T| \le h_i T$, and g_{j2} is the point at minimum distance in D_j, none of the specified points can be in D_j.

Since D_j and D_i both contain P, D_j intersects D_i. Claim 5.4 implies that D_j is entirely contained in one of the three subsegments of D_i lying between h_i, g_{i0}, g_{i2}, and g_{i3}, respectively. Therefore all the break points in D_j must be a subset of the break points in exactly one of α_{i0}, α_{i1}, or α_{i2}. Hence $n_j \le \|\alpha_{i1}\|$. Since $2\|\alpha_{i1}\| \le n_i$, we conclude $2n_j \le n_i$.

Since $n_1 = 2$ and $n_i < n$ for all i, it follows that $k = O(\log n)$. ∎

Each point on the boundary of an obstacle either is a break point or will be traversed at most as many times as the first break point in the clockwise direction. We have the following theorem.

THEOREM 5.5. *Let $\mathcal{O}$ be the set of obstacles with a point at distance at most $|ST|$ from T. Let P_{Ob} be the length of the boundary of $Ob \in \mathcal{O}$ and n_{Ob} be its complexity. The length of the path generated by 2-REPEATS is bounded by*

$$|ST| + \sum_{Ob \in \mathcal{O}} O(P_{Ob} \log n_{Ob}). \tag{5.1}$$

Thus, the excess distance ratio of 2-REPEATS is $O(\log n)$, where n is the maximum complexity of any obstacle in $\mathcal{O}$.

5.2. An improved worst-case bound. We can generalize the above algorithm by making the robot repeat the sequence αg more than twice in the departure sequence. This makes the robot search farther in hopes of finding a better departure point. On the other hand, if the robot finds the closest point right away, then all the remaining repetitions are just a waste of time.

A *q-repeat* departure sequence is a sequence of the form

$$h\alpha_0 g_0 \alpha_1 g_1 \dots \alpha_q g_q$$

such that $\|\alpha_{i-1}\| = \|\alpha_i\|$ for $2 \le i \le q$ and g_i is a global minimum in the sequence. Lemma 5.2 can be easily extended to q-departure sequences, for any q, since the robot can generate such a departure sequence by going entirely around an obstacle q times. The containment property established by Claim 5.4 still holds, and for any constant q, we can extend Lemma 5.3 to show that a boundary point is traversed $O(\log_q n)$ times.

To decrease the asymptotic bound we further require departure sequences to satisfy either the conditions

$$\|\alpha_1\| < 4 \quad \text{and} \quad q = 2$$

or the conditions

$$\|\alpha_1\| \ge 4 \quad \text{and} \quad q = \left\lceil \frac{\log \|\alpha_1\|}{\log\log \|\alpha_1\|} \right\rceil.$$

We call the navigation algorithm using departure sequences of this form MORE-REPEATS.

THEOREM 5.6. *Using the algorithm MORE-REPEATS, the length of the path generated by the robot is bounded by*

$$|ST| + \sum_{Ob \in \mathcal{O}} O(P_{Ob} \log n_{Ob} / \log\log n_{Ob}), \tag{5.2}$$

where $\mathcal{O}$, P_{Ob}, and n_{Ob} are defined as in Theorem 5.5.

Proof. Consider a particular break point P on an obstacle Ob of complexity n. Suppose P is traversed in a sequence of encounters $E_1, \dots, E_k$. Our goal is to bound the length of the sequence, k, since for $j < k$ each boundary point is traversed exactly once and in encounter k the obstacle is circumnavigated $O(\log n / \log\log n)$ times. For convenience we assume $n \ge 4$. Otherwise each break point is visited $O(1)$ times.

We use the definitions of D_i and n_i from the proof of Lemma 5.3. Claim 5.4 from that proof can be extended to any value of q. We can therefore conclude that $n_{i-1} \le \|\alpha_{i1}\| < n_i$ for $i > 1$.

To bound k, we first bound the number of encounters E_i in which $n_i \le n^{1/\log\log n}$. Let k' be maximal such that $n_{k'} \le n^{1/\log\log n}$. Since $n_i \ge 2\|\alpha_{i1}\|$ by either condition, we have $2n_{i-1} \le n_i$ for $1 < i$. Also, $n_1 = 2$. Therefore $n_i \ge 2^i$, and $n_{k'} \le n^{1/\log\log n}$ implies $k' < \log n / \log\log n$.

Now we bound the number of encounters E_i when $i > k' + 1$. For any $i > k' + 1$, the number of repetitions q satisfies

$$q \ge (\log \|\alpha_{i1}\|)/(\log\log \|\alpha_{i1}\|)$$

and

$$\|\alpha_{i1}\| \ge n_{i-1} \ge n_{k'} \ge n^{1/\log\log n},$$

which imply that

$$q \geq (\log n)/(\log\log n)^2.$$

By construction, $n_i \geq q n_{i-1}$, and thus

$$\begin{aligned} n_i &\geq n_{i-1} \frac{\log n}{(\log\log n)^2} \\ &\geq \left(\frac{\log n}{(\log\log n)^2}\right)^{i-k'}. \end{aligned}$$

Since $n_k \leq n$

$$n \geq (\log n/(\log\log n)^2)^{k-k'},$$

which implies

$$k - k' = O(\log n/\log\log n). \qquad \square$$

Thus, the excess distance ratio of MORE-REPEATS is $O(\log n/\log\log n)$, where n is the maximum complexity of any obstacle in $\mathcal{O}$.

6. Lower bounds in the indoor setting. In the previous section we defined the class MONOTONE, containing navigation algorithms that always depart an obstacle from a point no farther from the target than the hitpoint. We now define another class of navigation algorithms, ONE-WAY. A one-way navigation algorithm chooses a fixed direction (clockwise or counterclockwise) for the robot to travel around obstacle boundaries. Upon hitting an obstacle, the robot always travels around the obstacle in that direction and never reverses its direction of traversal. We allow a one-way algorithm to use any possible departure point, and therefore MONOTONE and ONE-WAY are incomparable classes.

The navigation algorithms in this paper all belong to the intersection of MONOTONE and ONE-WAY. We now show that the excess distance used by MORE-REPEATS is optimal among deterministic algorithms in the union of MONOTONE and ONE-WAY.

THEOREM 6.1. *Each deterministic ONE-WAY algorithm for target-reachable navigation in the indoor setting has an excess distance ratio of* $\Omega(\log n/\log\log n)$.

Proof. Let A be a deterministic ONE-WAY navigation algorithm for the indoor setting. We assume that it directs the robot to travel CCW (i.e., turn right) around any obstacle; the CW case is analogous. Let n be a sufficiently large positive integer. We construct a room wall of complexity at most n on which the excess distance traveled by A is $\Omega(\log n/\log\log n)$ times the perimeter of the obstacle. Let $k = \log n/c_1 \log\log n$, where the constant $c_1 > 1$ is chosen to make k an integer and will be determined later.

The overall plan is shown in Figure 6.1. T is placed at the origin, and a room wall is constructed. Most of the wall is a circular arc of radius 1. The shaded region contains a fractal-like boundary segment.

To fill the shaded region, we recursively construct an open path parameterized from $t = 0$ starting at the left side, called the *entry point*, to $t = 1$ at the right side called the *exit point*. The entry and exit points of the path will be at distance 1 from T. To complete the room wall, the open path is closed by an arc of radius

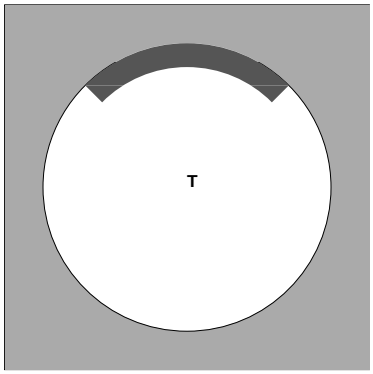


FIG. 6.1. *The structure of the obstacle for the lower bound proofs. The dark shaded region will contain a complex boundary segment spliced into a circular wall centered on T.*

1 centered on T, connecting the exit to the entry. The path is constructed so that vertical segments have very small dimension, and horizontal segments have either very small dimension or dimension slightly greater than 1.

For simplicity, we describe, picture, and analyze our constructions in a Cartesian coordinate system and then consider the effect of transforming them to our actual setting. In Cartesian coordinates, the path will consist of axis-parallel horizontal and vertical segments. Horizontal segments represent arcs of circles centered at T, and vertical segments represent segments of rays from T. We use a rectangle of width w and height h to represent a portion of an annulus between a circle of radius 1 and a circle of radius $1-h$, with outer arc length w and inner arc length $(1-h)w$. Then a boundary of path length L in the rectangle represents a boundary of path length L' in the portion of the annulus, where

$$(1-h)L \leq L' \leq L.$$

The basic structure used is a hook-shaped piece of room wall boundary, as indicated in Figure 6.2. This structure is called a *simple hook*. The L-shaped portion of the boundary is called the *arm* of the hook, and the boundary segment shown in bold is called the *recursive region* of the hook.

If we assume that the boundary corresponding to the recursive region of a hook has been defined, with its entry and exit points, we go on to define the remaining boundary of the hook as follows. Let $\epsilon = 1/k^{2k}$. The entry point of the hook is ϵ to the left of the left edge of the arm, and the exit point of the hook is aligned with the right edge of the arm. The arm itself has width ϵ and extends ϵ to the right of the exit point of the recursive region. It also clears the lowest point of the recursive region by ϵ. This is indicated schematically in Figure 6.2.

We define long and short hooks at level i (H_i^L and H_i^S) and long and short hook sequences at level i (B_i^L and B_i^S) as follows. The recursive region of H_1^L is a horizontal segment of length 1; the recursive region of H_1^S is a horizontal segment of length 0. Once H_i^L and H_i^S are defined, B_i^L consists of one level i long hook (H_i^L) followed by $k-1$ level i short hooks (H_i^S), while B_i^S consists of k consecutive level i short hooks (H_i^S). Finally, once long and short hook sequences at level i are defined, we define level $i+1$ long and short hooks as follows. H_{i+1}^L has B_i^L in its recursive region, and

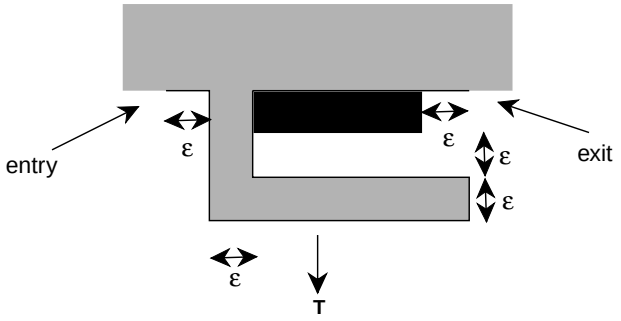


FIG. 6.2. *The basic hook shape. The recursive region is contained in the dark rectangle.*

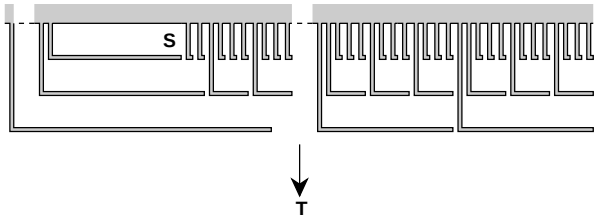


FIG. 6.3. *Constructing B_3^L (with $k = 3$). B_2^L is the recursive region of the long hook H_3^L, and two copies of the short hook H_3^S are added on the right.*

H_{i+1}^S has B_i^S in its recursive region. Figure 6.3 gives an example of the construction of a long hook sequence at level 3 (assuming $k = 3$.)

The intuition behind the construction is as follows. The robot cannot differentiate between traversing a long hook at a given level and traversing a short hook of the same level, since both yield the same ground track. If the robot ever traverses the boundary of a long hook followed by all $k - 1$ short hooks at the same level, then replacing the short hooks by a circular arc makes the robot repeat the boundary of the long hook k times. On the other hand, if the robot always takes off toward T after traversing the long hook on a level (for example, leaving from the bottom of one of the short hooks), then at each level i the long hook will be traversed $k-i+1$ times because of the recursive construction, thus also resulting in a path that is $\Omega(k)$ times as long as the obstacle's perimeter.

Define the *width* of a hook or hook sequence to be the minimum width of an axis-parallel rectangle containing it, and similarly define *height*. Let $W(i, S)$ and $W(i, L)$ denote the width of H_i^S and H_i^L, respectively. Let $ht(i)$ denote the height of both H_i^S and H_i^L. These functions satisfy the following recurrences:

$$
\begin{aligned}
W(1, S) &= 3\epsilon, \\
W(i, S) &= 3\epsilon + k \cdot W(i-1, S), \quad i > 1, \\
W(1, L) &= 1 + 3\epsilon, \\
W(i, L) &= 3\epsilon + W(i-1, L) + (k-1)W(i-1, S), \quad i > 1, \\
ht(1) &= 2\epsilon,
\end{aligned}
$$

$$ht(i) = ht(i-1) + 2\epsilon, \quad i > 1.$$

Using these recurrences, it follows that $W(i,S) = \Theta(\epsilon k^{i-1})$, $W(i,L) = 1 + \Theta(\epsilon k^{i-1})$, and $ht(i) = 2i\epsilon$.

Next we consider the path lengths of the boundary segments. Recall that $W(i,L)$ is the width of hook H_i^L. The path length of the arm H_i^L is $2W(i,L)+2ht(i)-\epsilon$. Since $\epsilon = 1/k^{2k}$ and $ht(i) = \Theta(i\epsilon)$, the contribution of the arm to the path length of H_i^L is $\Theta(W(i,L))$ for $i \le k$. A similar statement holds for H_i^S. Let $P(i,L)$ and $P(i,S)$ denote the total length of H_i^L and H_i^S, respectively. According to our discussion, these path lengths satisfy the following recurrences:

$$\begin{aligned}
&P(0,S) = 0,\\
&P(i,S) = \Theta(W(i,S)) + kP(i-1,S), \quad i > 0,\\
&P(0,L) = 1,\\
&P(i,L) = \Theta(W(i,L)) + (k-1)P(i-1,S) + P(i-1,L), \quad i > 0.
\end{aligned}$$

Together with the formulas for $W(i,S)$ and $W(i,L)$, these recurrences imply $P(i,S) = \Theta(\epsilon i k^{i-1})$ and $P(i,L) = \Theta(i + \epsilon i k^{i-1})$.

Now we consider the transformation from the Cartesian obstacles to the actual obstacles composed of arcs of circles centered at T and segments of rays to T. At the risk of slight confusion, we will use the same names, H_i^S, H_i^L, and so on, for the transformed obstacles. Then $ht(i) = 2i\epsilon$ is equal to 1 minus the minimum distance to T at any point on the boundary of H_i^L or H_i^S. Let $P'(i,L)$ denote the path length of the transformed boundary H_i^L. By our previous remarks,

$$P'(i,L) \ge (1 - 2i\epsilon)P(i,L).$$

Finally, consider the complexity of H_i, denoted $C(i)$. A single H_i consists of $O(1)$ new boundary segments plus the number of segments in the recursive region. Hence $C(i)$ satisfies

$$\begin{aligned}
&C(1) = \Theta(1),\\
&C(i) = \Theta(1) + kC(i-1).
\end{aligned}$$

This implies $C(i) = \Theta(k^{i-1})$.

It can be verified by induction on the level that H_i^L contains exactly one long hook and one long hook sequence at each of the levels $1 \ldots i$ and that H_j^L (respectively, B_j^L) is the leftmost hook (respectively, hook sequence) of level $j \le i$. It can also be verified by induction that if the robot executes *move-to-T*() from some point in the recursive region of H_i it will hit a point that belongs to the boundary of H_i and lies clockwise from the departure point.

The bad obstacle will be constructed from H_i^L, for some $i \le k$, together with the circular arc around T that closes the path. To choose the value of i, consider the actions of a robot using some algorithm A to navigate out of the obstacle H_k^L. The robot is started inside the level 1 long hook, shown as S in Figure 6.4.

The robot's motion can be divided into phases that correspond to the levels of the obstacle. Assume that at the start of phase i, $i < k$, the robot executes *move-to-T*() from a point s that is unblocked-towards and such that the first obstacle the robot will hit is the arm of the single long level i hook, H_i^L. This assumption holds true in phase 1, when the robot is started at point S.

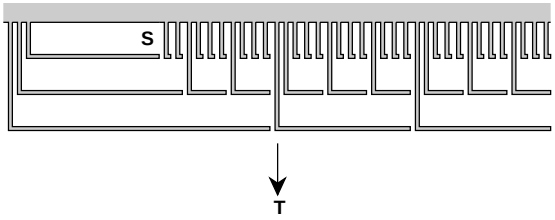


FIG. 6.4. *The result of the construction in Figure* 6.3.

Upon hitting the hook of H_i^L, the robot must turn right (recall the assumption of CCW traversal) and follow the boundary. Eventually, it must make its way through the recursive region and reach the exit point of H_i^L. Otherwise, it can never reach the target, because the arm of H_i^L intersects all rays from the target to points in the recursive region. (We say the hook *blocks* the target from these points.)

After reaching the exit of H_i^L, the robot may continue on through the subsequent copies of H_i^S that make up the hook sequence B_i^L. Either it passes through all k copies and reaches the exit of B_i^L, or it departs from a point that is not blocked from the target by any portion of the boundary making up B_i^L, for example, the bottom of some short hook.

If the robot passes through all k copies, then we choose H_i^L to be the obstacle. Observe that the robot cannot distinguish the ground track given by passing through each successive copy of H_i^S from the ground track formed by going completely around the circular arc from exit to entry and then through the single copy of H_i^L. Hence the robot will circumnavigate the obstacle boundary k times, giving an excess distance ratio $\Theta(k)$.

If the robot departs from a point on B_i^L that is not blocked from the target by any other point on that boundary, then at that time the robot satisfies the assumptions for the start of phase $i+1$. Now suppose that the robot completes k phases, in each phase departing before visiting all k copies. We choose the bad obstacle to be H_k^L. We show that in this case, the total distance traveled by the robot is $\Omega(k^2)$.

CLAIM 6.2. *The distance traveled by the robot in phase i is at least $P'(i, L)$.*

Proof. The robot must at least move from the starting point of phase i to the exit of H_i^L by executing some sequence consisting of *follow-boundary(RIGHT)* operations and *move-to-T*() operations. Since any *move-to-T*() will put the robot at a point clockwise (left) of the departure point, and hence farther from the exit than the departure point, omitting all *move-to-T*() operations only decreases the total distance traveled. But a sequence of *follow-boundary(RIGHT)* operations will still traverse all points on the path between the starting point and the exit.

Using Claim 6.2, the total distance traveled over all phases is at least

$$\sum_{i=1}^{k-1} P'(i, L) \geq (1 - 2k\epsilon) \sum_{i=1}^{k-1} P(i, L) = \Omega((1 - 2k\epsilon)(k^2 + k^{k+1}\epsilon)).$$

Since $\epsilon = 1/k^{2k}$, the ratio of distance traveled to the length of the perimeter of the obstacle, $O(k + \epsilon k^k)$, is $\Omega(k)$.

The maximum value of k is determined by the maximum allowed complexity of the obstacle, n. Since $C(k) = \Theta(k^{k-1})$, any obstacle which satisfies $k^{k-1} \leq n$ is

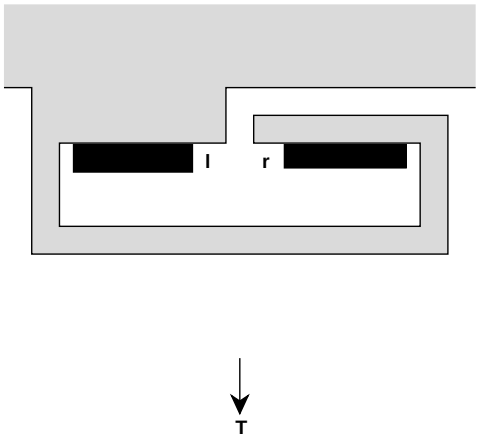


FIG. 6.5. *A simple corral, with left and right exit points and recursive regions shown in bold.*

feasible, and in particular we may choose $c_1 > 1$ and $k = \log n / c_1 \log\log n$. The theorem follows. □

Using a somewhat more complex construction, we prove a similar lower bound on the excess distance used by any deterministic MONOTONE algorithm for reachable-target navigation in a room.

THEOREM 6.3. *Any deterministic MONOTONE algorithm for reachable-target navigation in the indoor setting has an excess distance ratio of* $\Omega(\log n / \log n \log n)$.

Proof. Let A be a deterministic MONOTONE navigation algorithm for the indoor setting. Recall that algorithm A has a property such that upon hitting an obstacle it follows the obstacle boundary, changing direction at any break point. However, it will never leave from a point that is farther from the target than its hitpoint on the obstacle. Let n be a sufficiently large positive integer. We construct a room wall of complexity at most n on which the excess distance traveled by A is $\Omega(\log n / \log\log n)$ times the perimeter of the obstacle. Let $k = \log n / c_1 \log\log n$, where $c_1 > 1$ is a constant chosen to make k an integer, which will be determined later.

As in the proof of Theorem 6.1, we specify our construction using Cartesian coordinates and then transform the boundary so that the top edge lies on the arc of a circle of radius 1 centered at T and is closed using the rest of the circle to form a room wall obstacle. The basic structure we use is a U-shaped piece of room wall boundary, as indicated in Figure 6.5, and referred to henceforth as a *simple corral*. A simple corral has two exits, marked l and r, and two recursive regions, shown in bold.

We first describe the construction of short corrals and short corral sequences. Let $\epsilon = 1/k^{\beta k}$, where $\beta > 0$ is a constant to be determined below. As in the preceding proof, widths and clearances are taken to be size ϵ in the construction of short and long corrals. A level 1 short corral, C_1^S, is a simple corral in which the recursive regions are horizontal segments of length 0. A level i short corral sequence, B_i^S, consists of $2k+1$ copies of C_i^S connected end to end. For $i > 1$, a level i short corral, C_i^S, is constructed from a simple corral by replacing both the left and right recursive regions with B_{i-1}^S. Figure 6.6 gives an example of the construction of a level 3 short corral for $k = 1$. If $W(i, S)$ denotes the width of a level i short corral, then we have

$$W(1, S) = 9\epsilon,$$

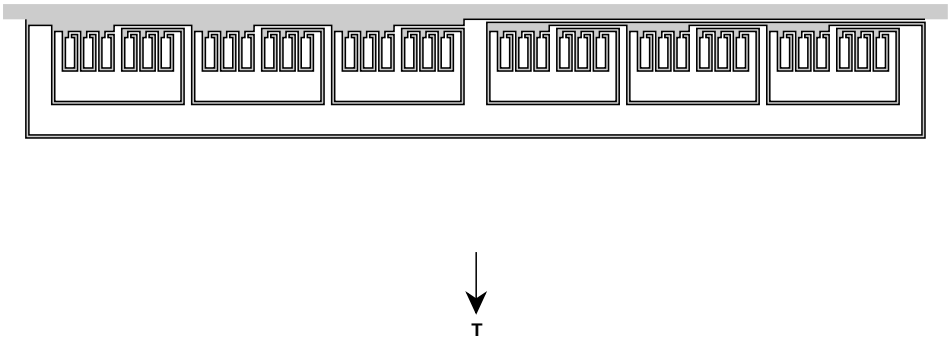


FIG. 6.6. *A level* 3 *short corral.*

and for $i > 1$,

$$W(i, S) = 9\epsilon + 2(2k + 1)W(i - 1, S).$$

The solution to the recurrence is

$$W(i, S) = 9\epsilon((4k + 2)^i - 1)/(4k + 1)) = \Theta(\epsilon(4k + 2)^{i-1}).$$

We also define long corrals and long corral sequences. For level i, there are 2^i possible different long corrals; which one we use at each level in the final construction will depend on the behavior of the algorithm A. A level 1 long corral is a simple corral in which the one of the two recursive regions is a horizontal segment of length 1, and the other is a horizontal segment of length 0. A level i long corral sequence consists of a sequence of k level i short corrals, followed by one level i long corral, followed by k more level i short corrals. For $i > 1$, a level i long corral is constructed from a simple corral by replacing one recursive region by a level i long corral sequence and the other recursive region by a level i short corral sequence. The width of a level i long corral is exactly one more than the width of a level i short corral, that is, $1 + \Theta(\epsilon(4k + 2)^{i-1})$.

To specify the final obstacle, we consider the motion of the robot in a bottom-up fashion with respect to the transformed boundaries in which horizontal segments represent arcs of circles centered at T, and vertical segments represent segments of rays from T. As before, we use the same names for the transformed constructions, C_i^S, C_i^L, and so on. As in the proof of Theorem 6.1 we divide the robot's motion into phases. The robot begins phase 1 inside one of the two possible level 1 long corral sequences, where the level 1 long corral has its long arc in the left recursive region if the robot first leaves the corral at the left exit, and in the right recursive region otherwise. The starting point S is shown in Figure 6.7; in this example, the robot exits the level 1 long corral at the left exit.

Assume that at the start of phase i, $1 < i < k$, we have constructed a path consisting of a particular level $(i - 1)$ long corral sequence, B_{i-1}^L, and the choice of long and short corrals is completely specified within the path. Assume also that the robot has just executed *move-to-T*() from some point s on one of the bottom segments of B_{i-1}^L and that this is the first action the robot has taken so far that would result in it reaching a point not on B_{i-1}^L.

Consider the level i long corral C_l given by embedding B_{i-1}^L into the left recursive region of a simple corral and embedding a copy of B_{i-1}^S into the right region. To

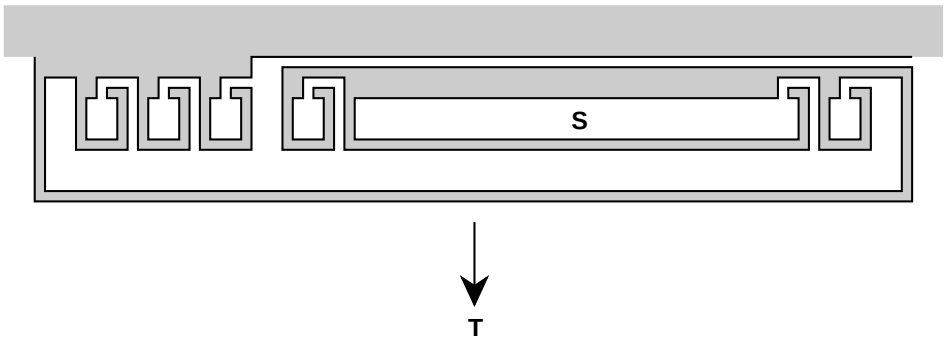


FIG. 6.7. *A level* 2 *long corral with a level* 1 *long corral in its right recursive region. The level* 1 *long corral has a long arc in its left recursive region.*

escape the corral in phase i, the robot must eventually reach one of two exit points, l or r. Since it is MONOTONE, the robot must eventually traverse every point on the boundary between the hitpoint and whichever exit it reaches. (This property is not true of a ONE-WAY algorithm on this boundary.)

If the robot reaches the left exit, then we have guaranteed that the robot retraces all of B_{i-1}^L during phase i. Suppose it only reaches the right exit, r. Then consider the level i long corral C_r given by embedding B_{i-1}^L into the *right* recursive region of a simple corral and embedding a copy of B_{i-1}^S into the left region. Since the robot has not ventured outside of B_{i-1}^L prior to the start of phase i, its behavior up until the start of phase i is the same regardless of whether it starts in C_l or C_r. Furthermore, the robot's behavior between the start of phase i and its reaching r must also be the same whether it is started in C_l or C_r. This follows from two observations: first, the initial hitpoint is on a segment of constant distance, and so the robot cannot tell from which side it departed at the start of phase i; second, the ground track given by traversing the recursive region is the same whether the region is long or short. Hence on obstacle C_r the robot still reaches the right exit and so retraces all of B_{i-1}^L during phase i. We set C_i^L to be whichever of C_r or C_l is chosen. In Figure 6.7 we show an example in which the phase 1 choice is the left exit and the phase 2 choice is the right exit.

Having chosen a particular level i long corral C_i^L, we use it to construct the long corral sequence B_i^L by adding k copies of C_i^S to both sides of C_i^L. Consider the behavior of the robot on B_i^L. The robot executes some sequence σ of *follow-boundary(LEFT)* and *follow-boundary(RIGHT)* instructions. It ends with a *follow-boundary(·)* instruction that will take it beyond one of the ends of the boundary path, or it executes *move-to-T*() from the bottom of one of the level i corrals.

Suppose the robot executes a *follow-boundary(·)* instruction that will take it past the left (respectively, right) end of B_i^L. Then the robot must have traveled at least once through each of the k level i short corrals between C_i^L and the left (right) end of B_i^L. We choose the final obstacle Ob to be just C_i^L, extended to wrap entirely around the target. The robot cannot distinguish the ground track observed while executing instruction sequence σ on this obstacle from the ground track observed on while executing σ on B_i^L. Since σ includes k traversals of level i corrals, the robot

must make k traversals of the boundary of obstacle Ob, giving an excess distance ratio of $\Omega(\log n/\log logn)$.

On the other hand, suppose the robot executes *move-to-T*() before exiting from B_i^L. At this point, phase i ends and phase $i+1$ starts. Note that the assumptions for the start of a phase are satisfied. If the robot completes k phases, let the final obstacle Ob be the long corral C_k^L that is constructed in phase k, with an arc joining the exit point to the entry point. Let $P(i,L)$ denote the path length of the Cartesian boundary C_i^L. For this quantity we can show $P(i,L) = \Theta(i+(4k+2)^{i-1}\epsilon)$. If $P'(i,L)$ is the path length of the transformed boundary C_i^L, then we have

$$(1-3i\epsilon)P(i,L) \le P'(i,L) \le P(i,L).$$

We can choose $\epsilon = 1/k^{\beta k}$ small enough that $P'(i,L) = \Theta(i)$. As shown above, the obstacle is constructed so that in phase i the robot retraces the whole path C_{i-1}^L. Hence the total distance traveled by the robot is $\Omega(k^2)$, and the excess distance ratio is $\Omega(k) = \Omega(\log n/\log\log n)$. The maximum value of k is determined by the maximum allowed complexity of the obstacle, n. The complexity of a level i corral, denoted $C(i)$, is described by a recurrence analogous to that in the proof of Theorem 6.1 and can be seen to be $\Theta(k^{\alpha k})$ for a small constant α. Hence $C(i) \le n$ for an appropriate choice of c_1 in the equation $k = \log n/c_1 \log\log n$. □

7. Navigation and circumnavigation in the outdoor setting. In this section we present an algorithm, FAR-NEAR, for reachable-target navigation in the outdoor setting. FAR-NEAR has an excess distance ratio bounded by 3. A variant of FAR-NEAR performs circumnavigation in the reachable-target outdoor setting, traveling around the obstacle boundary at most two times.

7.1. The FAR-NEAR algorithm. The robot starts by moving toward T. If any action brings it to T, it halts, having reached T. When the robot collides with obstacle Ob, it searches left until it reaches T or until its ground track forms a *circumnavigating departure sequence.*

A circumnavigating departure sequence has the form

$$h\alpha_0 Q_1 \alpha_1 P_1 \alpha_2 Q_2 \alpha_3 P_2,$$

where P_1 and P_2 are unblocked-towards break points of minimum distance and Q_1 and Q_2 are unblocked-away break points of maximum distance, where maximum and minimum are taken over all break points in the sequence.[4] The robot departs from P_2.

LEMMA 7.1. *Suppose the robot collides with normal obstacle Ob at hitpoint h. Traveling clockwise from h, either the robot will reach T or its ground track will eventually form a circumnavigating departure sequence.*

Proof. If T is on the boundary of Ob, then the robot will reach T by the time it has circumnavigated Ob from h. If T is not on the boundary of Ob, then Properties 3.18, 3.19, and 3.20 of section 3 imply the existence of points P, Q such that P is a break point on the boundary at minimum distance to T and Q is a break point at maximum distance to T. Starting at P, two circumnavigations of the boundary of Ob give the desired ground track. (The lemma is not true if the obstacle is a room wall.) □

Thus, the algorithm FAR-NEAR is in the class MONOTONE. The next lemma shows that if the robot encounters a departure sequence of the above form, it has completely circumnavigated the obstacle boundary.

[4] Thus perhaps the algorithm should be called FAR-NEAR-FAR-NEAR.

LEMMA 7.2. *Let Ob be a normal obstacle with boundary β. In any proper subsegment s of β there is no sequence of distinct points P_1, Q_1, P_2, Q_2 (or Q_1, P_1, Q_2, P_2) in order, such that P_1 and P_2 are unblocked-towards and of minimum distance (within s) to T, and Q_1 and Q_2 are unblocked-away and of maximum distance (within s) to T.*

Proof. Without loss of generality we may assume $T \notin Ob$. To see this, suppose there is a counterexample to the lemma consisting of an obstacle Ob containing T and a proper subsegment s of its boundary with the required properties. By rerouting the boundary of Ob, we can construct an obstacle Ob' such that $T \notin Ob'$ and Ob' and s also constitute a counterexample to the lemma. If T is in the interior of Ob, we take a segment of the boundary of Ob disjoint from s and cut a narrow channel from there into the interior of Ob that includes T, so that $T \notin Ob$. If T is on the boundary of Ob, then T is not in s (otherwise, $P_1 = P_2 = T$, contradicting the distinctness of P_1 and P_2) and we slightly reroute the boundary of Ob near T to ensure $T \notin Ob'$. Because the points in s are unaffected by the rerouting in either case, Ob' and s give a counterexample to the lemma in which $T \notin Ob'$. Thus, we assume $T \notin Ob$ for the remainder of the proof.

The proof is by contradiction. Consider a scene containing only the given obstacle and suppose there is a proper subsegment s of the boundary of the obstacle that starts with point P_1, contains point Q_1, then point P_2, and ends with point Q_2, where P_1 and P_2 are unblocked-towards and of minimum distance to T among points of s and Q_1 and Q_2 are unblocked-away and of maximum distance to T among points of s.

Let s_{11} denote the segment of s from P_1 to Q_1, let s_{12} denote the segment of s from Q_1 to P_2, and let s_{22} denote the segment of s from P_2 to Q_2. Also, let s' denote the rest of the boundary, that is, the segment from Q_2 back to P_1.

We define three circles centered at T: C_{ex}, which completely contains the obstacle in its interior; C_{max}, which contains the points Q_1 and Q_2; and C_{min}, which contains the points P_1 and P_2 (see Figure 7.1.) We now argue that Q_2 must be interior to the boundary b consisting of line segments P_1T, P_2T and boundary segments s_{11} and s_{12}. Since every point interior to this boundary has distance to T strictly smaller than $|Q_1T|$, this contradicts our assumption that Q_2 is also at maximum distance to T in the segment s.

To argue that Q_2 must be interior to b, we construct two paths to divide the disc D_{ex} bounded by C_{ex} using the following pieces. Let R_1 be the point on C_{ex} that is beyond Q_1 on the ray TQ_1. Since T is exterior to the obstacle by assumption, there exists a simple path p to T from outside D_{ex} that does not intersect the boundary of the obstacle. There must be a segment u of p that intersects C_{min} in some point R_3, intersects C_{max} in some point R_4, and otherwise is contained strictly between the two circles C_{min} and C_{max}. Let R_2 be the point on C_{ex} beyond R_4 on the ray TR_4. Since R_4 is different from Q_1, R_2 must be different from R_1.

Now we construct a simple path β_1 in D_{ex} from R_2 to R_1 consisting of line segment R_2R_4, path segment u, line segment R_3T, line segment TP_1, boundary segment s_{11}, and line segment Q_1R_1. The path β_1 does not contain the points P_2 or Q_2 and divides the disc D_{ex} into two regions. P_2 and Q_2 must be in the same region of D_{ex} with respect to β_1, because the boundary segment s_{22} joining them does not intersect β_1. To see this, note that s_{22} lies between the circles C_{min} and C_{max} and cannot intersect path segment u (by construction) or boundary segment s_{11}. Since these are the only portions of β_1 between C_{min} and C_{max}, s_{22} does not intersect β_1.

We construct another simple path β_2 from R_2 to R_1 consisting of line segment

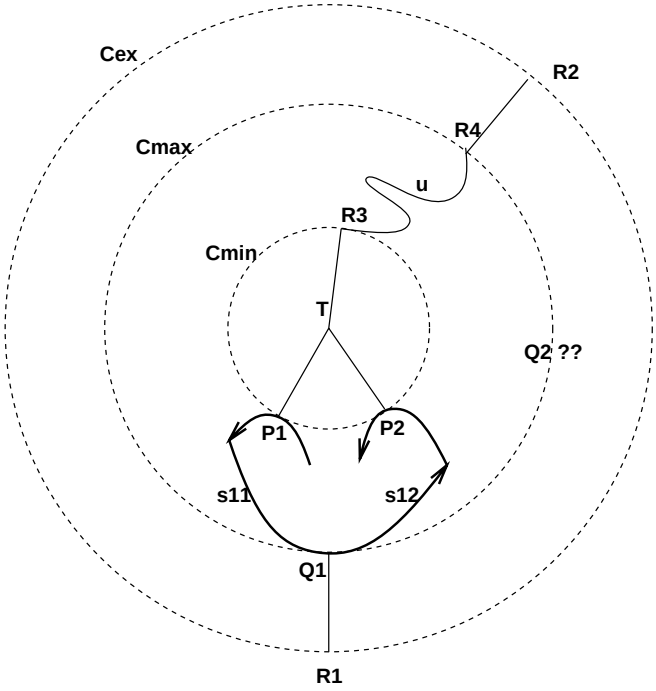


FIG. 7.1. *Figure for proof of Lemma* 7.2.

R_2R_4, path segment u, line segment R_3T, line segment TP_2, boundary segment s_{12}, and line segment Q_1R_1. Then β_2 does not contain points P_1 or Q_2 and it also divides the disc D_{ex} into two regions. We now argue that P_1 and Q_2 must be in the same region of D_{ex} with respect to β_2. To do so, we consider the segment s' of obstacle boundary from Q_2 to P_1. Although s remains between circles C_{min} and C_{max}, we have no such guarantee for s'. However, we argue that s' must cross the path β_2 an even number of times, which means P_1 and Q_2 are in the same region of D_{ex} with respect to β_2.

Considering each piece of the path β_2 in turn, we have the following. The segment R_2R_4 joins two points exterior to the obstacle and must be crossed by the boundary of the obstacle an even number of times. Because s lies between the circles C_{min} and C_{max}, the only crossings of R_2R_4 must be by s'. The path segment u does not intersect the boundary of the obstacle at all, so there are zero crossings of it by s'. The segment R_3T joins two points exterior to the obstacle and must be crossed by the boundary of the obstacle an even number of times. The only portion of the boundary that can intersect this segment is s'. By Property 3.22, the P_2T-probe, which we denote by P_2X, is exterior to the obstacle except for the point P_2, because P_2 is unblocked-towards and a local minimum of distance to T. Since T and X are both exterior to the obstacle, the subsegment TX must be crossed by the boundary of the obstacle an even number of times, and the only part of the boundary that can intersect it is s'. Since the segment P_2X is exterior to the obstacle except for P_2, it does not intersect s' at all. The boundary segment s_{12} does not intersect the boundary segment s' at all. Finally, by Property 3.23, the Q_1R_1-probe, which we denote by Q_1Y, is exterior to the obstacle except for the point Q_1 (because Q_1 is unblocked-away and a local

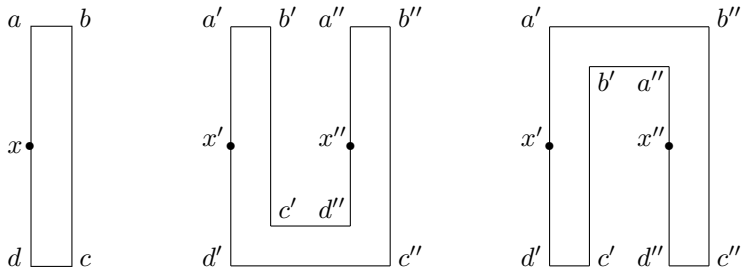


FIG. 7.2. *Obstacles for circumnavigation.*

maximum of the distance to T). Then s' does not intersect Q_1Y at all, and it crosses YR_1 an even number of times, because Y and R_1 are exterior to the obstacle.

Since each portion of the path β_2 either is not intersected by s' or is crossed an even number of times, P_1 and Q_2 are in the same region of D_{ex} with respect to β_2.

Finally, since Q_2 and P_2 are in the same region of D_{ex} with respect to β_1 and also Q_2 and P_1 are in the same region of D_{ex} with respect to β_2, Q_2 is in the region bounded by b consisting of s_{11}, s_{12}, and line segments TP_1 and TP_2. Clearly Q_2 cannot be a point of the boundary b, so it must be interior to b, which is the contradiction we sought. The case in which the points appear in the order Q_1, P_1, Q_2, and P_2 is handled analogously. □

THEOREM 7.3. *Let $\mathcal{O}$ be the set of obstacles with a point at distance at most $|ST|$ from T. Let P_{Ob} be the length of the boundary of $Ob \in \mathcal{O}$. The length of the path generated by the FAR-NEAR navigation algorithm is bounded by $|ST|+\sum_{Ob\in\mathcal{O}} 3P_{Ob}$.*

Proof. The robot collides with any obstacle Ob at most once, since by Lemma 7.2, the departure sequence circumnavigates Ob at least once.

Traveling from the hitpoint to Q_1 is at most one traversal of the obstacle boundary, from Q_1 to Q_2 is at most a second, and from Q_2 to P_2 is at most a third. As remarked in [4, 8], if the robot leaves an obstacle by a circumnavigating departure sequence, it will never hit that obstacle again. □

Thus, the FAR-NEAR algorithm has excess distance ratio ≤ 3. Compare this to the lower bound of 1 from Lumelsky and Stepanov [4]. A variant of the navigation algorithm can be used to detect circumnavigation.

THEOREM 7.4. *There is a deterministic algorithm to circumnavigate any normal obstacle that does not contain the target in its interior, using at most two complete traversals of its boundary.*

Proof. By Lemma 7.2, if the robot detects a sequence of break points P_1, Q_1, P_2, and Q_2 (or Q_1, P_1, Q_2, and P_2) such that P_1 and P_2 are unblocked-towards and of minimum distance to T among the boundary points traversed so far, and Q_1 and Q_2 are unblocked-away and of maximum distance to T among the boundary points traversed so far, then it has circumnavigated the obstacle. Moreover, if T is exterior to the obstacle, then such a sequence of points will be detected in at most two traversals of the boundary, using Properties 3.19 and 3.20. In the case that T is on the boundary of the obstacle, this fact will be detected in one boundary traversal, and, by the time T is reached again, the robot has circumnavigated the obstacle. □

The optimality of two boundary traversals for deterministic circumnavigation of normal obstacles not containing T in the interior is shown in the next subsection.

7.2. A lower bound for detecting complete circumnavigation.

THEOREM 7.5. *For any deterministic boundary-following circumnavigation algorithm and for any $\epsilon > 0$, there exists a normal obstacle not containing the target for which the robot's path length is at least $2 - \epsilon$ times the perimeter of the obstacle.*

Proof. Suppose we are given any deterministic algorithm for boundary-following circumnavigation in our model and any $\epsilon > 0$. We first analyze the behavior of the algorithm on the three obstacles depicted in Figure 7.2 to show that when it is started at point x on the rectangle, every possible computation path must make, in addition to an initial traversal of half the edge ad, at least two complete traversals of one of the vertical edges and one complete traversal of the other. If we take the horizontal dimension of the rectangle to be sufficiently small compared to the vertical dimension, this implies that the robot's path length will be at least $7/4 - \epsilon$ times the perimeter of the rectangle in every computation path. (As before, the vertical segments of the diagram represent segments aligned along rays to T and the horizontal segments represent arcs of a circle centered at T.)

Consider any computation path of the given algorithm when started at point x on the rectangle. The initial segment of the robot's path must traverse half the edge ad to reach either a or d. Suppose the point reached is a; then we focus on the $\sqcup$-shaped object; a corresponding argument will hold for the $\sqcap$-shaped object if the robot reaches d instead.

Note that the only points in the $\sqcup$-shaped object that are distinguishable to the robot from their counterparts in the rectangle are the points c' and d''. Clearly, to complete a circumnavigation of the rectangle, the robot must completely traverse both edge ad and edge bc. (Recall that in our model, the robot has no way of stopping in the middle of traversing a vertical edge.) Suppose for the sake of contradiction that these are the only two full traversals of vertical edges of the rectangle. Then in this computation path, either edge ad is traversed first and then edge bc or vice versa.

In the case that the edge ad is traversed first, consider what can happen when the robot, making the same choices, is started at point x' on the $\sqcup$-shaped obstacle. The initial segment of the path takes it to point a'. Then it traverses the edge $a'd'$ and then the edge $b''c''$ and halts without traversing any other vertical edges and without reaching either of the distinguishing points c' and d''. Thus, in this computation, the robot erroneously halts without circumnavigating the $\sqcup$-shaped object.

A similar argument holds in the case that the computation path of the robot traverses edge bc and then edge ad. In this case, there is a corresponding computation path for the robot when started at point x'' in the $\sqcup$-shaped obstacle that first proceeds to a'', traverses edge $b''c''$ and then edge $a'd'$, and halts without circumnavigating the $\sqcup$-shaped obstacle. Thus in either case our assumption that the path contained only two full traversals of vertical edges leads to a contradiction.

The other case is that in which the computation path we are considering takes the robot first to point d. Analogously, the assumption that the robot performs only two further full traversals of vertical edges in the rectangle leads to a contradiction by considering the behavior of the robot using the same choices in the $\sqcap$-shaped object, starting either at x' or x''.

To improve the lower bound to $2 - \epsilon$, start the robot at point x on the rectangle and observe whether it proceeds to point a or point d. If it proceeds to point a, then we slide the rectangle up until the point x is very close to the point d. If the robot begins by proceeding to point d, then we slide the rectangle down until point x is very close to point a. Then in either case, the initial segment traversed is very nearly the whole vertical edge ad. The argument that we have at least three more

vertical traversals to go still holds, which gives a lower bound of $2 - \epsilon$ times the perimeter. □

8. Remarks. Although our definition of a scene specifies a finite collection of obstacles, our results hold as long as a finite number of obstacles intersect the disk centered at the target with radius equal to the distance from start to target.

If the robot is given an a priori upper bound on the number of break points on any obstacle, then it can detect circumnavigation of an obstacle by simple counting. Our negative result of section 4 will not hold. Our algorithms will still be preferable, however, when the average number of break points on an obstacle is substantially less than the upper bound. Our algorithms do not need to save their entire ground track; they can be implemented using $O(1)$ registers.

An important problem in vision-based tracking is that the robot's view of the target is sometimes occluded by intervening obstacles. Taylor and Kriegman classify obstacles into transparent obstacles, which are invisible to the robot until it actually collides with them, and opaque obstacles, which the robot can detect because they intervene in the line of sight [8]. Although the robot can detect the existence of opaque obstacles, it cannot determine the distance to such an obstacle until, once again, it actually collides with it. In practice, transparent obstacles are things such as tables and boxes that are lower than the robot camera, and opaque obstacles are tall cabinets, bookcases, and partitions. Taylor and Kriegman's procedures work even if the target is obscured at various positions, as long as the target is visible from the start point.

Our algorithm for navigation within a room can be modified to cope with regions of obscured visibility, provided the target is visible from the starting point. The key intuition is that whenever the robot moves in open space it moves directly toward T, so when it encounters an obstacle, T is visible at the hit point. This ensures that the obstacle has some unblocked-towards point that is nearer to T than the hitpoint and from which T is visible. The above bounds still hold, but the definition of break point must be extended to include points at the end of boundary segments from which the target is continuously visible.

There are several interesting problems left open by our work. Is there a deterministic algorithm for navigation in the indoor setting that achieves an excess distance ratio of $o(\log n / \log \log n)$? Our lower bound results (section 6) show that if there is such an algorithm, it must be neither MONOTONE nor ONE-WAY, that is, it must travel in both clockwise and counterclockwise directions, and it must sometimes depart obstacles from break points that are farther from the target than the prior hitpoint. Can the excess distance ratio for navigation in the free space problem be reduced below 3? Do these results extend to other surfaces besides the plane? Is there a significant advantage in using randomized algorithms for these problems? We note that the lower bound of $7/4 - \epsilon$ for circumnavigation given in section 7.2 holds against randomized algorithms (for an appropriate definition of randomization). We believe that the impossibility results of section 4 and the lower bounds of section 6 can also be extended to randomized models.

Appendix. Proof of Theorem 4.1.

We begin by proving some technical lemmas, and then we restate Theorem 4.1 and give its proof.

DEFINITION A.1. *Let $p = (r(t), \theta(t))$, $t \in [0, 1]$, be an arbitrary path. Let j and k be integers $j \geq 0$ and $k > 0$. Then $\Gamma(p, j, k)$ is the path $(r'(t), \theta'(t))$ for $t \in [0, 1]$,*

where

$$r'(t) = r(t),$$
$$\theta'(t) = \frac{1}{k}(\theta(t) + 2\pi j).$$

$\Gamma(p, j, k)$ is an angularly scaled and rotated copy of the path p. It preserves distances from the origin but divides angular displacements by k, and it also rotates the start of the path counterclockwise around the origin by $2\pi j/k$. We use $\Gamma(p, j, k)(t)$ to indicate the point on the curve $\Gamma(p, j, k)$ given by parameter t. The following technical lemmas will be useful.

LEMMA A.2. *Given paths p_1 and p_2 let $p_1' = \Gamma(p_1, j_1, k)$ and $p_2' = \Gamma(p_2, j_2, k)$. If $p_1'(t_1) \equiv p_2'(t_2)$ for some $t_1, t_2 \in [0, 1]$, then $p_1(t_1) \equiv p_2(t_2)$.*

Proof. Let $p_1 = (r_1(t), \theta_1(t))$ and let $p_2 = (r_2(t), \theta_2(t))$. By hypothesis and the definition of Γ,

$$r_1(t_1) = r_1'(t_1) = r_2'(t_2) = r_2(t_2)$$

and

$$(\theta_1(t_1) + 2\pi j_1)/k = (\theta_2(t_2) + 2\pi j_2)/k + 2\pi i$$

for some integer i. Hence $\theta_1(t_1) = \theta_2(t_2) + 2\pi(j_2 - j_1 + ik)$. This implies $p(t_1) \equiv p(t_2)$, since j_1, j_2, i, and k are integers. Note that p_1 and p_2 may be the same path. □

LEMMA A.3. *Given paths p_1 and p_2, suppose $p_1(t_1) \equiv p_2(t_2)$ for some $t_1, t_2 \in [0, 1]$. Then for any nonnegative integer $j_1 < k$ there exists a nonnegative integer $j_2 < k$ such that $p_1'(t_1) = \Gamma(p_1, j_1, k)(t_1) \equiv \Gamma(p_2, j_2, k)(t_2) = p_2'(t_2)$.*

Proof. Let $p_1 = (r_1(t), \theta_1(t))$ and let $p_2 = (r_2(t), \theta_2(t))$. For any choice of j_1, j_2,

$$r_1'(t_1) = r_1(t_1) = r_2(t_2) = r_2'(t_2)$$

by hypothesis and definition of the transform Γ. By hypothesis, for some integer i,

$$\theta_1(t_1) = \theta_2(t_2) + 2\pi i.$$

Given integer j_1 with $0 \le j_1 < k$, let $j_2 = (i + j_1) \pmod k$. Then for some integer m, $j_2 = (i + j_1) + mk$, and

$$\theta_2'(t_2) = \frac{\theta_2(t_2) + 2\pi j_2}{k} = \frac{\theta_2(t_2) + 2\pi i + 2\pi j_1 + 2\pi mk}{k} = \frac{\theta_1(t_1) + 2\pi j_1}{k} + 2\pi m$$
$$\equiv \theta_1'(t_1). \qquad \square$$

LEMMA A.4. *Given path $p = (r(t), \theta(t))$ let $p_1 = \Gamma(p, j_1, k)$ and $p_2 = \Gamma(p, j_2, k)$ for $0 \le j_1, j_2 < k$. If $p_1(t_0) \equiv p_2(t_0)$ for some single value of the parameter $t = t_0$, then $j_1 = j_2$.*

Proof. By hypothesis $\theta_1(t_0) = \theta_2(t_0) + 2\pi i$ for some integer i. Hence

$$(\theta(t_0) + 2\pi j_1)/k = (\theta(t_0) + 2\pi j_2)/k + 2\pi i.$$

This implies $(j_1 - j_2)/k = i$. But since $0 \le j_1, j_2 < k$, equality can only hold if $j_1 = j_2$ and $i = 0$. □

DEFINITION A.5. *Let $p = (r(t), \theta(t))$, $t \in [0, 1]$, be an arbitrary curve, and let $k > 0$ be an integer. Then $Z(p, k)$ is the path given by the union of the family of*

curves $\Gamma(p, j, k)$ *for* $0 \le j < k$. *That is, the angular part of* p *is scaled by* k, *and then* k *copies of the result are placed around the origin, the* i*th copy starting at angle* $2\pi i/k$ *and ending at angle* $2\pi(i+1)/k$. $Z(p,k)$ *can be written as*

$$(r(\{s\}), (\theta(\{s\}) + 2\pi\lfloor s \rfloor)/k),$$

$s \in [0, k]$ *(where* $\{s\} = s - \lfloor s \rfloor$ *denotes the fractional part of* s*).*

Let Ob be an obstacle containing T in its bounded domain (i.e., interior). We assume the target is at the origin $(0,0)$ and that the boundary of Ob, $\beta = (r(s), \theta(s))$, $s \in [0,1]$, is parameterized so that $\theta(0) = 0$, $\theta(1) = 2\pi$, and θ increases in a counterclockwise direction. Let $\beta^* = Z(\beta, k)$. The next sequence of lemmas establishes that β^* is a simple closed curve. Furthermore, β^* can be used to define an obstacle Ob^* such that the sequence of ground track entries produced by one circumnavigation of Ob^* is exactly the same as the sequence of ground track entries produced by k circumnavigations of Ob. Figure 4.1 shows an example obstacle and Figure 4.2 shows the result of applying Z to that obstacle with $k = 6$.

LEMMA A.6. *The curve* $\beta^* = Z(\beta, k)$ *is simple and closed.*

Proof. That $\beta^*(0) \equiv \beta^*(k)$ follows from the definition of $\Gamma(\cdot,\cdot,\cdot)$ and the fact that $r(0) = r(1)$, $\theta(0) = 0$, and $\theta(1) = 2\pi$. Hence β^* is closed.

To show that β^* is simple, we show that for any s_1, s_2 such that $0 \le s_1 < s_2 < k$, $\beta^*(s_1) \not\equiv \beta^*(s_2)$. (Since $\beta^*(0) \equiv \beta^*(k)$ all cases with $s_2 = k$ are covered by cases with $s_1 = 0$.)

Let $j_1 = \lfloor s_1 \rfloor$ and $j_2 = \lfloor s_2 \rfloor$. Then $\beta^*(s_1)$ is the point on curve $\Gamma(\beta, j_1, k)$ given by parameter $\{s_1\}$, and $\beta^*(s_2)$ is the point on curve $\Gamma(\beta, j_2, k)$ given by parameter $\{s_2\}$. If $\beta^*(s_1) \equiv \beta^*(s_2)$, then by Lemma A.2, $\beta(\{s_1\}) \equiv \beta(\{s_2\})$. Since β is a simple closed curve by definition, this is only possible if $\{s_1\} = \{s_2\}$. Applying Lemma A.4 we conclude $j_1 = j_2$, and hence $s_1 = s_2$, a contradiction. □

Since β^* is a simple closed curve, it divides the plane into two regions, one of which contains T.

LEMMA A.7. *The target* T *is in the interior (bounded) domain of* β^*.

Proof. Suppose T is not in the bounded domain. Then there is a path q^* from T to a point Q arbitrarily far from T such that q^* does not intersect β^*. Define $q^* = (r'(t), \theta'(t))$, $t \in [0,1]$, and let q be the path $(r'(t), k \cdot \theta'(t))$.

Since Q can be made arbitrarily far from T, $q(1)$ is in the unbounded region of Ob. Therefore, path q must cross β at some point $\beta(s_0) \equiv q(t_0)$. By definition, therefore,

$$r'(t_0) = r(s_0), \tag{A.1}$$

$$k \cdot \theta'(t_0) = \theta(s_0) + 2\pi i \tag{A.2}$$

for some integer i. Now let $j = i \pmod k$, and consider the curve $\Gamma(\beta, j, k)$ at point s_0. By definition $\Gamma(\beta, j, k)(s_0) = (r(s_0), (\theta(s_0) + 2\pi(i \pmod k))/k)$. On the other hand, A.2 yields

$$\theta'(t_0) = \frac{\theta(s_0) + 2\pi(kx + i \pmod k)}{k} = \frac{\theta(s_0) + 2\pi(i \pmod k)}{k} + 2\pi x$$

for some integer x. Therefore $q^*(t_0) \equiv \Gamma(\beta, j, k)(s_0)$. This contradicts the assumption that q^* does not touch β^*, since $\Gamma(\beta, j, k)(s_0) \subset \beta^*$. □

If Ob is a normal obstacle, then let Ob^* be the obstacle with boundary β^* whose obstructed region is the bounded domain containing T. If Ob is a room wall, then let the obstructed region of Ob^* be the unbounded domain not containing T. Next we study the relationship between the break points of Ob and those of Ob^*.

LEMMA A.8. *Point $P^* = \beta^*(s)$ is a local minimum or local maximum with respect to distance if and only if point $P = \beta(\{s\})$ is a local minimum or local maximum, respectively.*

Proof. Since distances are unchanged, the distances of the points in a small neighborhood of P^* are the same as the distances of the points in the corresponding neighborhood of P. The only special case occurs when $\{s\} = 0$, but in this case the points on β^* in the distance $s - \varepsilon$ are the same as the points on β in the distance $1 - \varepsilon$. □

LEMMA A.9. *Point $P^* = \beta^*(s_0)$ is unblocked-towards if and only if $P = \beta(\{s_0\})$ is unblocked-towards.*

Proof. For convenience, assume for the moment that Ob is a normal obstacle. We show that if P is unblocked-towards, then P^* is also unblocked-towards. By definition, if P is unblocked-towards, the PT-probe is a segment PR of nonzero length that is contained entirely in free space. Let Q be some point in free space at distance greater than any boundary point of Ob. Since Q is in free space (the unbounded region is free with respect to Ob) there is a path connecting R to Q that lies entirely in free space. Let q denote a path consisting of the segment from P to R followed by the path from R to Q.

Let $j_0 = \lfloor s_0 \rfloor$. Let $q' = \Gamma(q, j_0, k)$. Then $P^* \equiv q'(0)$. Note that q' begins with a segment P^*R^* of nonzero length that lies along the ray P^*T. We show that q' lies entirely in free space, which implies that the segment P^*R^* is in free space. This then implies that P^* is unblocked-towards.

To show that q' lies entirely in free space, we observe that by Lemma A.2, the curve q' can only intersect β^* at point $q'(0) \equiv P^*$. Since the point Q^* is at a distance greater than any point on Ob^*, this point must be in free space, and since q' does not cross the boundary of Ob^*, all other points on q' must be in the same region as Q^*.

Suppose on the other hand that point P is blocked-towards. We construct a path q that begins at P, starts with a small segment PR along ray PT, and then continues from R to T along a path entirely contained within the obstacle. We apply the same transformation to q to derive a path from T to P^* lying entirely within one domain of the obstacle boundary β^*. Since T is in the obstructed domain by definition, this path is entirely within the obstructed region and hence shows that P^* is blocked-towards.

If the obstacle is a room wall rather than a normal obstacle, then the exterior domain is the obstructed region. In this instance, we reverse the cases of unblocked-towards and blocked-towards, since if a point is blocked-towards, then there is a path within the obstacle to some point Q very far away, and if it is unblocked-towards, then there is a path in free space to T. □

LEMMA A.10. *Point $P^* = \beta^*(s_0)$ is unblocked-away if and only if $P = \beta(\{s_0\})$ is unblocked-away.*

Proof. The proof is analogous to that of Lemma A.9, the only difference being that the probes point away from T rather than toward T. □

COROLLARY A.11. *Point $P^* = \beta^*(s_0)$ is a break point if and only if $P = \beta(\{s_0\})$ is a break point. Furthermore, if $P^* = \beta^*(s_0)$ is a break point, then the ground track entry generated at P^* is identical to the ground track entry generated at P.*

Proof. The corollary is a consequence of Lemmas A.8, A.9, and A.10. □

THEOREM 4.1. *In the distance query model, if there may exist an obstacle whose interior contains the target (either a normal obstacle with the target in its obstructed region, or a room wall with the target in its complement), then there is no deterministic algorithm for circumnavigation. Furthermore, there are arbitrarily complex obstacles*

which any given deterministic algorithm fails to circumnavigate.

Proof. Let A be any circumnavigation algorithm. Let Ob be any sufficiently complex obstacle containing T in its bounded domain (i.e., interior). We assume the target is at the origin and the boundary of Ob, $\beta = (r(s), \theta(s))$, $s \in [0,1]$, is parameterized so that $\theta(0) = 0$ and $\theta(1) = 2\pi$. The robot is started at point $\beta(0) = (r(0), \theta(0))$ using algorithm A.

If algorithm A fails to circumnavigate Ob, either by halting incorrectly or never halting, then the theorem holds immediately. Otherwise we construct an obstacle Ob^* which A fails to circumnavigate.

Choose a constant k as follows. Suppose the robot halts after receiving m ground track entries. Then $rp(m) = (\rho(t), \vartheta(t))$, $t \in [0,1]$, is the complete path followed by the robot while performing its circumnavigation.

Choose any integer k such that $k > \max_t\{|\vartheta(t)|\}/\pi$. Let $\beta^* = Z(\beta, k)$, and let Ob^* be the obstacle with boundary β^* whose interior region contains T. Lemmas A.8–A.10 and Corollary A.11 hold of the obstacle Ob^* and its boundary β^*. We claim that on Ob^* the robot will follow the path $p^* = (\rho(t), \vartheta(t)/k)$. If this claim is true, then for all s,

$$-\frac{\max_s\{|\vartheta(s)|\}}{k} \le \vartheta^*(s) \le \frac{\max_s\{|\vartheta(s)|\}}{k}.$$

Since $k > \max_s\{|\vartheta(s)|\}/\pi$, this implies $-\pi < \vartheta^*(s) < \pi$. But since every ray from T intersects Ob^* at some point there must be a point at angle π, which is not reached by the robot in traveling over path p^*. Hence the robot fails to circumnavigate Ob^*.

It remains to prove the claim. The proof is by induction on the number of ground track entries generated as the robot travels path p. We show that

(i) if $rp(\ell)$ on Ob is $(\rho(t), \vartheta(t))$, $t \in [0,1]$ then $rp^*(\ell)$ on Ob^* is $(\rho(t), \vartheta(t)/k)$.

To do this, we show that two additional invariants hold:

(ii) For all ℓ, the first ℓ ground track entries generated while running on Ob^* are identical to the first ℓ entries generated while running on Ob.

(iii) If $rp^*(\ell) = \beta^*(j_0 + s_0)$, $s_0 \in [0,1)$, and $0 \le j_0 < k$, then $rp(\ell) = \beta(s_0)$.

For the basis case, observe $rp(0) \equiv rp^*(0) = S$, and all the invariants are trivially true. Now suppose the robot has generated ℓ ground track entries while running on Ob^*. By invariant (ii), on both Ob and Ob^*, the navigation algorithm, having seen the same set of ground track entries, will take the same action.

Case 1. The robot executes *follow-boundary(CCW)*. On both Ob and Ob^*, the robot's path is coincident with the respective obstacle boundary until reaching the first break point in the CCW direction. On Ob, $rp(\ell+1) = rp(\ell) + \beta([s_0, s_1])$. Now set

$$j_1 = \begin{cases} j_0 & \text{if } s_0 < s_1 < 1, \\ j_0 + 1 \bmod k & \text{if } 0 \le s_1 < s_0. \end{cases}$$

By Corollary A.11, $\beta^*(j_1 + s_1)$ is the first break point the robot reaches on Ob^*. Therefore invariants (ii) and (iii) hold. Either path $rp^*(\ell+1) = rp^*(\ell) + \Gamma(\beta, j_0, k)([s_0, s_1])$ or $rp^*(\ell+1) = rp^*(\ell) + \Gamma(\beta, j_0, k)([s_0, 1]) + \Gamma(\beta, j_1, k)([0, s_1])$, depending on whether $s_0 < s_1 < 1$. By the definition of $\Gamma(\beta, j, k)$, invariant (i) holds.

Case 2. The robot executes *follow-boundary(CW)*. This is analogous to Case 1.

Case 3. The robot executes *move-to-T*(). Let $Q = \beta(s_0)$ be the point on Ob from which the robot departs. By invariant (iii), the robot will depart Ob^* at $Q^* = \beta^*(s_0 + j_0)$. Consider the ray $q = TQ$ and the transformed ray $q^* = S(q, j_0, k)$. From

Ob the robot travels inbound on ray q and from Ob^* the robot travels inbound on ray q^*. Let P be the hitpoint reached on Ob and let P^* be the hitpoint reached on Ob^*. We argue that $P = \beta(s_1)$ and $P^* = \beta^*(s_1 + j_1)$ for some j_1. This will immediately establish invariants (ii) and (iii). Invariant (i) also follows, since from both Ob and Ob^* the robot follows an inbound ray, without changing its angular coordinate, until stopping at a point at the same radius in both cases.

Since ray q intersects β at P, it follows from Lemma A.3 that ray q^* intersects β^* at some point $R^* = \beta^*(s_1 + j_1)$. Since P is blocked-towards, R^* is blocked-towards (Lemma A.9). If $R^* \equiv P^*$ we are done. Suppose therefore that $R^* \not\equiv P^*$. Then P^* must have a radius coordinate greater than R^*. Since ray q^* intersects β^* at P^*, it follows from Lemma A.2 that ray q intersects β at some point $R = \beta(s_2)$. Furthermore, R has the same radius coordinate as R^* and is also blocked-towards (Lemma A.9). But this contradicts the assumption that P is the first hitpoint encountered when the robot heads in along ray q. Therefore it must be the case that $R^* = P^*$, as desired. □

Acknowledgments. We thank David Kriegman and C. J. Taylor for bringing this problem to our attention, and Drew McDermott, Greg Hager, Stan Eisenstat, and the two anonymous referees for many helpful comments.

REFERENCES

[1] P. Berman, *On-line searching and navigation*, in Online Algorithms: The State of the Art, A. Fiat and G. J. Woeginger, eds., Springer, New York, 1998, pp. 232–241.

[2] C. Icking and R. Klein, *Competitive strategies for autonomous systems*, in Modelling and Planning for Sensor Based Intelligent Robot Systems, H. Bunke, T. Kanade, and H. Noltemeier, eds., World Scientific, River Edge, NJ, 1995, pp. 23–40.

[3] V. Lumelsky and A. Stepanov, *Dynamic path planning for a mobile automaton with limited information on the environment*, IEEE Trans. Automat. Control, AC-31 (1986), pp. 1058–1063.

[4] V. Lumelsky and A. Stepanov, *Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape*, Algorithmica, 2 (1987), pp. 403–430.

[5] V. Lumelsky and S. Tiwari, *An algorithm for maze searching with azimuth input*, in Proceedings of the IEEE Conference on Robotics and Automation, San Diego, CA, 1994, pp. 111–116.

[6] J. S. B. Mitchell, *Geometric shortest paths and network optimization*, in The Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier, New York, to appear.

[7] M. H. A. Newman, *Elements of the Topology of Plane Sets of Points*, Cambridge University Press, Cambridge, UK, 1961.

[8] C. J. Taylor and D. J. Kriegman, *Vision-based motion planning and exploration algorithms for mobile robots*, IEEE Trans. Robotics and Automation, 14 (1998), pp. 417–426.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 145–160

PREEMPTIVE SCHEDULING OF PARALLEL JOBS ON MULTIPROCESSORS*

XIAOTIE DENG†, NIAN GU‡, TIM BRECHT§, AND KAICHENG LU¶

Abstract. We study the problem of processor scheduling for n parallel jobs applying the method of competitive analysis. We prove that for jobs with a single phase of parallelism, a preemptive scheduling algorithm without information about job execution time can achieve a mean completion time within $2 - \frac{2}{n+1}$ times the optimum. In other words, we prove a competitive ratio of $2 - \frac{2}{n+1}$. The result is extended to jobs with multiple phases of parallelism (which can be used to model jobs with sublinear speedup) and to interactive jobs (with phases during which the job has no CPU requirements) to derive solutions guaranteed to be within $4 - \frac{4}{n+1}$ times the optimum. In comparison with previous work, our assumption that job execution times are unknown prior to their completion is more realistic, our multiphased job model is more general, and our approximation ratio (for jobs with a single phase of parallelism) is tighter and cannot be improved. While this work presents theoretical results obtained using competitive analysis, we believe that the results provide insight into the performance of practical multiprocessor scheduling algorithms that operate in the absence of complete information.

Key words. processor scheduling, parallel programs, competitive analysis, unknown information

AMS subject classifications. 68M20, 68Q22

PII. S0097539797315598

1. Introduction.

The CPU scheduling problem for computer systems distinguishes itself from general scheduling problems (e.g., job shop scheduling) in its variety of requirements of the system and variety of performance metrics. While minimizing makespan (the time at which the last job completes execution) is usually a natural objective function for many general scheduling problems, a number of different possibilities exist for CPU schedulers in a general purpose multiuser computing environment [31]. Nevertheless, minimizing the mean completion time (the sum of the times at which each job completes, divided by the number of jobs) is a commonly used objective function [22], [18], [34], [35], [27]. We can equivalently just consider the sum of the completion times. In this paper, the phrase *completion times* is used to imply that all jobs are available for execution at time zero, while the phrase *response time* implies that there are new job arrivals.

Several recent analytic results have been obtained for the problem of minimizing mean completion times using nonpreemptive scheduling algorithms which assume that

*Received by the editors January 27, 1997; accepted for publication (in revised form) May 27, 1999; published electronically April 25, 2000. A preliminary version of this paper appears in *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, January, 1996, pp. 159–167. This research was funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) and a strategic research grant and an RGC CERG grant from the City University of Hong Kong.

http://www.siam.org/journals/sicomp/30-1/31559.html

†Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Kowloon, Hong Kong SAR, P.R. China (deng@cs.cityu.edu.hk).

‡Department of Computer Science, York University, Toronto, Canada M3J 1P3 (nian_gu@yahoo.com).

§Department of Computer Science, University of Waterloo, Waterloo, ON Canada N2L 3G1 (brecht@cs.uwaterloo.ca).

¶Department of Computer Science, TsingHua University, Beijing, China, 100084 (lkc-dcs@mail.tsinghua.edu.cn).

job information is completely known [35], [34], [27], [18]. These results initiated the first (theoretical) step toward understanding the general problem. Our work takes the next significant step and is distinguished from these results in that *we remove the unrealistic assumption that the job execution time is known.* Knowledge of execution times for some jobs may be obtained, but this is clearly not a valid assumption for all jobs in general purpose computing environments.

Using the terminology of Feitelson et al. [9], the work by Turek et al. [35] and Schwiegelshohn et al. [27] examines nonpreemptive scheduling policies using a rigid job model. That is, the number of processors required by a job is defined by the job and must be assigned to the job for its lifetime. Subsequent work [34], [18] relaxes the job model to consider moldable jobs. A moldable job [9] is one which can be run on any number of processors provided that once the processors have been allocated they remain allocated to that job for the duration of its execution. In this case the scheduler is free to determine the number of processors to allocate to each job. However, once a job is assigned processors it cannot be preempted. Additionally, some of these studies [34], [18] explicitly model jobs with sublinear speedup.

By comparison, in this work we model malleable jobs in order to examine dynamic preemptive scheduling policies. That is, we model jobs that are capable of executing with a changing number of processors and scheduling algorithms that can modify the number of processors allocated to jobs during their execution in order to adjust to changing requirements or system loads. Note that in using this model, threads of a job can migrate (i.e., they can be suspended on one processor and resumed at a later time on another processor). We also explicitly study jobs with multiple phases of parallelism in order to approximate jobs with sublinear speedups. During each phase of execution a job is capable of executing with perfect speedup until a maximum degree of parallelism is reached. The addition of extra processors beyond this limit neither increases nor decreases the execution time of the job. While each phase executes with linear speedup (up to the maximum degree of parallelism), multiple phases of execution with different maximum degrees of parallelism can be combined to produce an overall model of sublinear speedup.

In this paper we show that the dynamic equipartition (DEQ) policy [33], [38] produces mean completion times that are no more than $2 - \frac{2}{n+1}$ times the optimum for any set of n parallel jobs with one phase of parallelism, and that no policy can guarantee a better competitive ratio without a priori knowledge of job execution times. Although the competitive ratio turns out to be the same as in the sequential problem (not necessarily by accident), our result requires a completely different and rather difficult proof. In fact, *the ratio of* $2 - \frac{2}{n+1}$ *cannot be further improved mathematically* for jobs with a single phase of parallelism.

This result provides a theoretical foundation for analyzing and understanding the performance of the DEQ policy, which, along with its various derivatives, has been shown to be superior to nonpreemptive algorithms in recent simulation and experimental studies [33], [17], [38], [16], [20], [21].

The remainder of the paper is organized as follows. We complete section 1 with a further description of the problem and a discussion of related work. In section 2, we give a formal definition of the DEQ allocation policy. Then we establish a lower bound on the optimal total completion time for parallel jobs by extending the squashed area bound and the height bound [34], using a completely different approach from those used previously. In section 3, we give a formal proof that the total completion time of DEQ is no more than twice the optimal total completion time for any job set. The

mathematical induction used in this proof requires a delicate balance of the squashed area and height bounds on the work needed to be executed by each job. Furthermore, in section 4, we show that our results can be extended to jobs which may change the number of processors required during their execution, including interactive jobs which may block and therefore need not be assigned to a CPU while waiting for user input. In section 5, we present theoretical results which demonstrate that DEQ is robust in the presence of faulty jobs. We consider the case where there are faulty jobs which may execute infinitely and show that, in this case, DEQ achieves the optimal competitive ratio for makespan. In section 6, we conclude the paper.

1.1. Preemptive scheduling. Schedulers in most general purpose computer systems are preemptive for several reasons [31]. First, job execution times may not be known prior to their completion. Thus, when nonpreemptive scheduling algorithms are used, short jobs may be penalized by long jobs which utilize the CPUs for long periods of time. Second, interactive jobs require some processing, and preemptive (time-sharing) scheduling policies allow them to execute by providing them with a slice of CPU time. Third, some jobs may execute infinitely, due to programming errors. If a nonpreemptive scheduling policy is used they may execute forever and exclude other jobs from being processed. Obviously, the preemptive execution of jobs incurs some overhead. For multiprocessor systems, this may become more expensive. However, these overheads can be absorbed in a time-sharing scheme by choosing a scheduling quantum that is sufficiently large or by dynamically space-sharing processors instead [33], [38], [20], [26]. This is consistent with the trend toward coarse grained machines for general purpose parallel computations, as suggested in the LogP model [3]. Setup costs can then be absorbed by pipeline routing if the size of a problem is sufficiently large in comparison with the number of processors in the system [36]. Independently, there have been extensive empirical studies on the preemptive cost caused by time/space-sharing scheduling policies [38], [20], [37]. Even for some cases when the preemption cost is relatively high, simulation and experimental studies support preemptive over nonpreemptive scheduling policies [38], [20], [23], [24].

1.2. Competitive analysis. We make the assumption that job execution times are not known prior to their completion. This is quite realistic for modern general purpose multiprocessors. Since execution times are not known at the time jobs are scheduled, it is possible that any given scheduling policy may not perform very well on some specific job set. For this reason, we use competitive analysis to study policies that do not deviate from the optimal solution (which has and uses complete information about the job set) by more than a constant factor. The competitive analysis of algorithms is a measure of algorithms operating with incomplete information, first introduced in the study of system memory management [32], [13], [19]. Policies for this problem are required to handle future unknown requests. The competitive ratio of a policy is defined to be the worst case ratio of the cost of the policy (which is different for different problems) to the optimal cost for the same input sequence. In the CPU scheduling problem the situation is similar in that the execution time of a job is unknown until its execution is completed. The competitive ratio of a scheduling policy S is thus defined as the worst case ratio of the mean completion time (or makespan), $S(\mathcal{J})$, of the policy on a set of jobs, $\mathcal{J}$, over the minimum mean completion time (or makespan) $OPT(\mathcal{J})$ on the same job set $\mathcal{J}$: $\max_{all\ \mathcal{J}} \frac{S(\mathcal{J})}{OPT(\mathcal{J})}$. An algorithm is said to be *f(n)-competitive* in mean completion time (or makespan) if $S(\mathcal{J}) \leq f(n)OPT(\mathcal{J})$. The goal is to find an algorithm which leads to the minimum

competitive ratio. Shmoys, Wein, and Williamson studied the optimal competitive ratio in the makespan of sequential jobs being scheduled on parallel machines [30]. For minimizing the mean completion time of sequential jobs, Motwani, Phillips, and Torng have shown that a preemptive time-sharing policy, round-robin, achieves the optimal competitive ratio. It guarantees a mean completion time which is within $2 - \frac{2}{n+1}$ times optimal, and no policy can guarantee a better competitive ratio [22]. As further evidence that preemptive policies should be preferred to nonpreemptive policies, it is not hard to see that any nonpreemptive policy can result in a mean completion time that is $\Omega(n)$ times the optimum when n jobs are scheduled.

1.3. The job model. The most detailed description of a parallel program's execution on a multiprocessor is a data-dependency directed acyclic graph (DAG), where edges represent data dependencies between the data (nodes). The DAG is revealed as the computation proceeds as a result of data-dependent conditional statements. Using a delay model introduced by Papadimitriou and Yannakakis [25], Deng and Koutsoupias show that given uniform communication delay, τ, for any scheduler, there exists a DAG for which the scheduler will produce a schedule whose execution is at least $\frac{\tau}{\log \tau}$ times the optimal execution time of that DAG [4]. The same claim holds for both the bulk synchronous parallelism (BSP) and the LogP models. That is, for a parallel system with communication latency L between processors, the competitive ratio of any scheduler is at least $\frac{L}{\log L}$. This work shows that it is not possible for a compiler to optimally (or near-optimally) execute all jobs for distributed memory parallel systems, and it calls for the characterization of parallel jobs and the use of these characteristics in scheduling policies.

We characterize a parallel job, J_i, using two parameters: its execution time, h_i, and its parallelism, P_i. P_i is the number of processors a job is capable of using during its execution, and h_i is the time that the job needs to complete execution if it is allocated P_i processors. When less than P_i processors are allocated to job J_i, we assume that the job's execution will be prolonged proportionally. That is, if p_i processors ($p_i < P_i$) are allocated to J_i, its actual execution time is $\frac{P_i}{p_i} h_i$. For a job, (P_i, h_i), its parallelism P_i is known to the scheduler but the execution time h_i is unknown prior to its completion. Therefore, jobs are considered malleable and the scheduling algorithms are dynamic and preemptive, since they can adjust the number of processors allocated to a job during its execution [9].

In general, we can use parallelism profiles to characterize parallel jobs. A parallelism profile is defined as the number of processors an application is capable of using at any point in time during its execution [15]. During execution, if the parallelism of an application varies with time, it is said to have multiple phases of parallelism. Note that although our job model assumes linear speedup within each phase of parallelism, jobs with multiple phases of parallelism will execute with sublinear speedup. We also consider interactive jobs by introducing phases during which a job does not require access to a processor because it is blocked while waiting for user input.

1.4. Related results. Motwani, Phillips, and Torng [22] show that, for uniprocessor systems, the mean completion time of the round-robin scheduling policy is $2 - \frac{2}{n+1}$ times the optimum and that without a priori information about job execution times, no policy can guarantee mean completion times within a better approximation factor of the optimum (called the competitive ratio [32], [13], [19]).

The problem of minimizing the mean completion time of parallel jobs executing on multiprocessors is also of interest, and here a number of positive results exist.

Recently, there have been several analytic results which assume that job information is completely known. The first significant work is that of Turek et al. [35] who introduce an approximation algorithm of 32 times the optimum for a nonpreemptive scheduling model. This result for nonpreemptive algorithms has been subsequently improved and extended [34], [27], [18].

A number of different preemptive policies have been proposed and studied for scheduling parallel jobs in multiprocessors [33], [29], [38], [20], [28], [26], [23], [24], [2]. In particular, experimental and simulation studies have shown that the DEQ algorithm yields low mean completion times under a variety of workloads and is reported to possess desirable properties of a good scheduler [33], [17], [16]. DEQ was first introduced to parallel scheduling by Tucker and Gupta as a *process control* policy [33] and was modified by Zahorjan and McCann [38]. The main idea behind this approach is to distribute processors evenly among jobs, provided they have sufficient parallelism.

One of the main drawbacks of DEQ, when compared with other approaches like gang scheduling, is that it requires each job to be implemented in such a way that the number of processes allocated to the job can change during its execution. It also requires significant coordination between the operating system and the run-time system. While these requirements might seem restrictive, studies have shown that it is relatively simple to write malleable applications, that coordination between the scheduler and run-time system is not prohibitive, and that performance is improved significantly when compared with other techniques [20], [10], [23], [24]. In addition, the DEQ algorithm is simple to implement and requires no information about job execution times. Therefore, this work examines the DEQ scheduling algorithm.

Our proof that DEQ is $2 - \frac{2}{n+1}$-competitive uses the notion of a *squashed area bound*, introduced for the nonpreemptive scheduling of parallel jobs [18], [35], [27], [34]. Turek et al. show that the minimum completion time for a set of jobs, $\mathcal{J} = \{(P_1, h_1), (P_2, h_2), \ldots, (P_n, h_n)\}$, is no more than the minimum completion time of the job set $\mathcal{J}_{squash} = \{(P, \frac{P_1 h_1}{P}), (P, \frac{P_2 h_2}{P}), \ldots, (P, \frac{P_n h_n}{P})\}$ [35]. The squashed area bound is the total completion time (the product of the number of jobs and the mean completion time) for $\mathcal{J}_{squash}$ under the least work first (LWF) policy. Sevcik shows that the LWF policy is optimal if all jobs have the same parallelism P [28].

2. Preliminaries. Consider n jobs in a system of P processors. Job J_i is characterized by the parallelism-time pair (P_i, h_i), and the amount of work is $w_i = P_i h_i$, $1 \leq i \leq n$. Denote the job set by $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$. Suppose that under a scheduler S the actual completion time of job J_i is t_i, $1 \leq i \leq n$. The total completion time of $\mathcal{J}$, denoted by $TC_S(\mathcal{J})$, is defined as $\sum_{i=1}^{n} t_i$. Then the mean completion time $MC_S(\mathcal{J})$ is defined as $\frac{TC_S(\mathcal{J})}{n}$. The height bound $H(\mathcal{J})$ is defined as $\sum_{i=1}^{n} h_i$ [35]. Since J_i requires at least h_i units of execution time, $1 \leq i \leq n$, $H(\mathcal{J})$ is an obvious lower bound on the optimal total completion time. Let the jobs be ordered according to their total work $w_1 \leq w_2 \leq \cdots \leq w_n$. The squashed area bound $A(\mathcal{J})$ is then defined as $\sum_{i=1}^{n} (n-i+1)\frac{w_i}{P}$ [35]. Notice that any preemptive scheduling of the job set $\mathcal{J}$ can be obtained by a preemptive scheduling of the job set $\{(P, \frac{w_1}{P}), (P, \frac{w_2}{P}), \ldots, (P, \frac{w_n}{P})\}$. It follows that $OPT(\mathcal{J}) \geq OPT\{(P, \frac{w_1}{P}), (P, \frac{w_2}{P}), \ldots, (P, \frac{w_n}{P})\}$. Since each job has the same parallelism, the shortest job first (or LWF) strategy gives the optimal solution for the total completion time, easily provable as in sequential systems (see [35], [28] for details). This gives exactly the squashed area bound.

Our main result utilizes a nontrivial extension to the squashed area bound and the height bound. Suppose each job (P_i, h_i) is divided into two parts: (P_i, h_{i1}) and (P_i, h_{i2}) such that $h_i = h_{i1} + h_{i2}$. Let $\mathcal{J}(1) = \{(P_i, h_{i1}) : 1 \leq i \leq n\}$ and $\mathcal{J}(2) =$

$\{(P_i, h_{i2}) : 1 \le i \le n\}$. We have the following lemma for a lower bound on the optimal total completion time of the job set $\mathcal{J}$.

LEMMA 2.1. $OPT(\mathcal{J}) \ge A(\mathcal{J}(1)) + H(\mathcal{J}(2))$.

Proof of Lemma 2.1. Consider the optimal scheduling algorithm on the input $\mathcal{J}$. Let t_{i1} be the time when the remaining portion of J_i is (P_i, h_{i2}) and t_{i2} be the time when J_i completes execution $1 \le i \le n$. Obviously, $t_{i2} - t_{i1} \ge h_{i2}$. The total completion time for the optimal scheduler is

$$OPT(\mathcal{J}) = \sum_{i=1}^{n} t_{i2} \ge \sum_{i=1}^{n} t_{i1} + \sum_{i=1}^{n} h_{i2} \ge A(\mathcal{J}(1)) + H(\mathcal{J}(2)),$$

where the last inequality is derived from the height bound and the squashed area bound [35]. □

We formally define the DEQ allocation policy recursively as follows:

(1) If $P_i \ge \frac{P}{n}$ for all $i : 1 \le i \le n$, each job is assigned $\frac{P}{n}$ processors.
(2) Otherwise, each job J_i with parallelism $P_i < \frac{P}{n}$ is allocated P_i processors. Update n and P. If $n > 0$, recursively apply DEQ.

Obviously, this schedule is valid only when $\frac{P}{n}$ is an integer. In practice, if $\frac{P}{n}$ is a rational number, and larger than 1, we can take its integer part $\lfloor \frac{P}{n} \rfloor$ and ignore its fraction part $\frac{P}{n} - \lfloor \frac{P}{n} \rfloor$. The result will be affected by a small constant factor. If $\frac{P}{n}$ is a fractional number smaller than 1, we view all the parallel jobs as sequential jobs and apply the round-robin policy so that in unit time, a fraction $\frac{P}{n}$ of one processor's CPU time is assigned to one job. To simplify our proof, we allow a fractional number of processors to be assigned to a job, $\frac{P}{n}$, as long as that number is smaller than the parallelism of the job. Let $\mathcal{J}_{para}$ be the set of jobs that are allocated P_i processors, and the rest of the jobs form the set $\mathcal{J}_{equi}$ (which are each assigned an equal number of processors, denoted by $\bar{p}$).

LEMMA 2.2. *If there are no idle processors, then* $\sum_{J_i \in \mathcal{J}_{para}} P_i + |\mathcal{J}_{equi}|\bar{p} = P$, $(\forall J_i \in \mathcal{J}_{para}) P_i \le \bar{p}$, *and* $\bar{p} \ge \frac{P}{n}$.

Consider the execution of jobs under the DEQ policy. Each job (P_i, h_i) is divided into two modes of execution: It is in *full-parallelism mode* if P_i processors are assigned, and it is in *equipartition mode* if less than P_i processors are assigned. It is not difficult to see that under the DEQ allocation policy, once a job enters full-parallelism mode, it will stay in that mode until completion. Let $h_i(f)$ be the length of execution of job i in full-parallelism mode, and $h_i(e) = h_i - h_i(f)$. Let $\mathcal{J}(f) = \{(P_i, h_i(f)) : 1 \le i \le n\}$ and $\mathcal{J}(e) = \{(P_i, h_i(e)) : 1 \le i \le n\}$. In the next section, we prove

$$TC_{DEQ}(\mathcal{J}) \le \left(2 - \frac{2}{n+1}\right)[A(\mathcal{J}(e)) + H(\mathcal{J}(f))]. \tag{2.1}$$

Combining this with Lemma 2.1, we have the following theorem.

THEOREM 2.3. $TC_{DEQ}(\mathcal{J}) \le (2 - \frac{2}{n+1})OPT(\mathcal{J})$. □

It is not hard to extend the lower bound for the competitive ratio of sequential jobs by Motwani, Phillips, and Torng [22] to this situation. In fact, we can replace each job in their proof with a parallel job of the same execution time with parallelism P, the number of processors in the system. Thus, this competitive ratio is optimal.

3. Minimizing mean completion time. In this section, we prove that (2.1) holds. Suppose jobs are initially divided into $\mathcal{J}_{para}$ and $\mathcal{J}_{equi}$ according to the discussion in section 2. We need the following lemma.

LEMMA 3.1. *If there are no idle processors, then*

$$(n+1)P|\mathcal{J}_{equi}| \le (n-1)P|\mathcal{J}_{para}| + n\bar{p}|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1).$$

Proof of Lemma 3.1.

$$\begin{aligned} RHS &= P(|\mathcal{J}_{equi}| + |\mathcal{J}_{para}| - 1)|\mathcal{J}_{para}| + n\bar{p}|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1) \\ &\ge P|\mathcal{J}_{equi}||\mathcal{J}_{para}| + P(|\mathcal{J}_{para}|-1)|\mathcal{J}_{para}| + P|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1) \\ &\ge P|\mathcal{J}_{equi}|(|\mathcal{J}_{para}| + |\mathcal{J}_{equi}| + 1) = P|\mathcal{J}_{equi}|(n+1). \end{aligned}$$

In the above, the first inequality follows from Lemma 2.2, and the second inequality follows from the fact $(|\mathcal{J}_{para}|-1)|\mathcal{J}_{para}| \ge 0$. □

When one of the jobs, say J_1, finishes its execution under DEQ, we will reallocate processors according to DEQ. Every job in $\mathcal{J}_{para}$ will still be assigned the same number of processors as its parallelism. Let $\mathcal{J}'_{para} \subseteq \mathcal{J}_{equi}$ be the subset of jobs in $\mathcal{J}_{equi}$ which are assigned the same number of processors as their parallelism after the reallocation of processors. Let $\mathcal{J}'_{equi} = \mathcal{J}_{equi} - \mathcal{J}'_{para}$. Thus, jobs in $\mathcal{J}'_{equi}$ are now assigned the same number of processors.

Proof of (2.1). For simplicity of presentation, let $C = 2 - \frac{2}{n+1}$. In order to avoid case-by-case analysis in the proof, we prove the claim by induction on the number of jobs which have nonzero execution time, n.

Since $2 - \frac{2}{n+1}$ is an increasing function of n and jobs of zero length would change neither the squashed area bound nor the height bound, the claim would also hold when we allow n to be the number of total jobs, including jobs of zero length. Therefore, we can simply prove the claim for the case when all jobs have nonzero lengths while allowing the induction hypothesis to include the case when jobs of zero length are present.

For the base case $n = 1$, if the parallelism, P_1, of job J_1 is less than or equal to P ($P_1 \le P$), it is assigned P_1 processors ($h_1(f) = h_1$). Otherwise, P processors are assigned to the job ($h_1(e) = h$) and its execution ends in time $\frac{P_1 h_1}{P}$, which is the same as the squashed area bound.

Assume the claim holds when the number of jobs of nonzero length is less than n. Consider the case of n jobs, all of nonzero length, $\mathcal{J} = \{(P_i, h_i) : 1 \le i \le n\}$. If there are idle processors, the claim follows immediately. So we assume there are no idle processors. Without loss of generality, let $J_1 = (P_1, h_1)$ be the first to finish and let τ denote its completion time. Therefore, the remaining portion of jobs $J_i \in \mathcal{J}_{equi}$ is $(P_i, h_i - \frac{\tau\bar{p}}{P_i})$, and the remaining portion of jobs $J_i \in \mathcal{J}_{para}$ is $(P_i, h_i - \tau)$. Therefore, the total completion time is

$$TC_{DEQ}(\mathcal{J}) = n\tau + TC_{DEQ}\left(\left\{\left(P_i, h_i - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\} \cup \{(P_i, h_i - \tau) : i \in \mathcal{J}_{para}\}\right), \tag{3.1}$$

where the first term is the completion time of J_1 plus the time that the other $n-1$ jobs have been in the system so far, and the second term is needed in recursion for the remaining portion of the $n-1$ jobs. By the induction hypothesis,the second term

in (3.1) is bounded by C times

$$A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\}\right) + H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \sum_{i\in\mathcal{J}_{para}} (h_i - \tau), \tag{3.2}$$

where $h_i(e) \geq \frac{\tau\bar{p}}{P_i}$ for each $i \in \mathcal{J}_{equi}$. DEQ will redistribute processors among jobs in $\mathcal{J}_{equi}$ after the departure of J_1. Let $\mathcal{J}'_{para} \subseteq \mathcal{J}_{equi}$ be the subset of $\mathcal{J}_{equi}$ such that for each $i \in \mathcal{J}'_{para}$, J_i is assigned P_i processors after the redistribution. Let $\mathcal{J}'_{equi}$ be the rest of the jobs in $\mathcal{J}_{equi}$. Since jobs in $\mathcal{J}'_{para}$ will stay in full parallelism mode, we have

$$\forall i \in \mathcal{J}'_{para} h_i(e) = \frac{\tau\bar{p}}{P_i}. \tag{3.3}$$

Similarly,

$$\forall i \in \mathcal{J}'_{equi} h_i(e) > \frac{\tau\bar{p}}{P_i}. \tag{3.4}$$

From (3.3), we have

$$A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\}\right) = A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}'_{equi}\right\}\right). \tag{3.5}$$

Let the jobs in $\mathcal{J}'_{equi}$ be ordered as $j_1, j_2, \ldots, j_k$, $k = |\mathcal{J}'_{equi}|$, according to the increasing order of the amount of remaining work $P_i(h_i(e) - \frac{\tau\bar{p}}{P_i})$, $i \in \mathcal{J}'_{equi}$, which is the same as the increasing order of $P_i h_i(e)$, $i \in \mathcal{J}'_{equi}$. Then,

$$\begin{aligned} A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}'_{equi}\right\}\right) &= \frac{1}{P}\sum_{i=1}^{k}(k-i+1)P_{j_i}\left(h_{j_i}(e) - \frac{\tau\bar{p}}{P_{j_i}}\right) \\ &= \frac{1}{P}\sum_{i=1}^{k}(k-i+1)P_{j_i}h_{j_i}(e) - \frac{1}{P}\sum_{i=1}^{k}(k-i+1)\tau\bar{p}. \end{aligned}$$

Therefore,

$$A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}'_{equi}\right\}\right) = A(\{(P_i, h_i(e)) : i \in \mathcal{J}'_{equi}\}) - \frac{k(k+1)}{2P}\tau\bar{p}. \tag{3.6}$$

We also have

$$H(\mathcal{J}_{equi}(f)) + \sum_{i\in\mathcal{J}_{para}} (h_i - \tau) = H(\mathcal{J}(f)) - |\mathcal{J}_{para}|\tau. \tag{3.7}$$

Combining (3.1) and (3.2), we know that $TC_{DEQ}(\mathcal{J})$ is no more than

$$n\tau + C \cdot \left(A\left(\left\{\left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i}\right) : i \in \mathcal{J}_{equi}\right\}\right) + H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \sum_{i\in\mathcal{J}_{para}} (h_i - \tau)\right).$$

By (3.5), this is the same as

$$n\tau + C \cdot \left(A\left(\left\{ \left(P_i, h_i(e) - \frac{\tau\bar{p}}{P_i} \right) : i \in \mathcal{J}'_{equi} \right\} \right) + H(\{(P_i, h_i(f)) : i \in \mathcal{J}_{equi}\}) + \sum_{i \in \mathcal{J}_{para}} (h_i - \tau) \right).$$

Now by substituting (3.6) and (3.7), we have the following upper bound for $TC_{DEQ}(\mathcal{J})$:

$$n\tau - C \cdot \frac{k(k+1)\tau\bar{p}}{2P} - C \cdot |\mathcal{J}_{para}|\tau + C \cdot A(\mathcal{J}'_{equi}) + C \cdot H(\mathcal{J}(f)).$$

Since

$$A(\mathcal{J}_{equi}(e)) = A(\mathcal{J}'_{equi}(e)) + \sum_{i=1}^{|\mathcal{J}'_{para}|} \frac{(k + |\mathcal{J}'_{para}| - i + 1)\tau\bar{p}}{P},$$

the above upper bound is equal to

$$n\tau - C \cdot \frac{k(k+1)\tau\bar{p}}{2P} - C \cdot |\mathcal{J}_{para}|\tau - C \cdot \sum_{i=1}^{|\mathcal{J}'_{para}|} \frac{(k + |\mathcal{J}'_{para}| - i + 1)\tau\bar{p}}{P} + C \cdot (A(\mathcal{J}_{equi}(e)) + H(\mathcal{J}(f))).$$

To show that this is no more than

$$C \cdot A(\mathcal{J}(e)) + C \cdot H(\mathcal{J}(f)) = C \cdot A(\mathcal{J}_{equi}(e)) + C \cdot H(\mathcal{J}(f)),$$

it is sufficient to show that

$$n \le C \cdot \frac{k(k+1)\bar{p}}{2P} + C \cdot |\mathcal{J}_{para}| + C \cdot \sum_{i=1}^{|\mathcal{J}'_{para}|} \frac{(k + |\mathcal{J}'_{para}| - i + 1)\bar{p}}{P}.$$

This inequality is the same as

$$nP \le C \cdot |\mathcal{J}_{para}| P + \frac{C}{2} |\mathcal{J}_{equi}| (|\mathcal{J}_{equi}| + 1)\bar{p}.$$

Equivalently,

$$|\mathcal{J}_{equi}| P \le (C-1) |\mathcal{J}_{para}| P + \frac{C}{2} |\mathcal{J}_{equi}| (|\mathcal{J}_{equi}| + 1)\bar{p}.$$

Since $C = 2 - \frac{2}{n+1}$, the above inequality follows from Lemma 3.1. □

4. Multiphased parallelism and interactive jobs. At any point in time some jobs are assigned a number of processors equal to their parallelism (those in full-parallelism mode) and others are assigned $\bar{p}$ processors (those in equipartition mode). Both the parallelism of the jobs and $\bar{p}$ may change over time. We continue to use the notation $\mathcal{J}(f)$ for the portion of jobs in $\mathcal{J}$ in full-parallelism mode, and $\mathcal{J}(e)$ for the portion of jobs in equipartition mode.

THEOREM 4.1. *For jobs with multiple phases of parallelism, we have*

$$TC_{DEQ}(\mathcal{J}) \leq \left(2 - \frac{2}{n+1}\right) A(\mathcal{J}(e)) + \left(2 - \frac{2}{n+1}\right) H(\mathcal{J}(f)). \tag{4.1}$$

Therefore, DEQ is $4 - \frac{4}{n+1}$ *competitive for mean job completion time.*

Proof. The conclusion of $4 - \frac{4}{n+1}$ competitiveness follows immediately from the fact that both the squashed area bound and the height bound are lower bounds on the optimal total completion time. Our focus is thus on (4.1). We consider an inductive proof using the result for single-phased jobs as the base case. A difficulty in this case is that the order of jobs in the squashed area bound may change as execution of the jobs (according to DEQ) proceed. To deal with this problem, we divide the execution time of the jobs into intervals such that in each interval, the parallelism of all jobs does not change; and the order, according to which the squashed area bound is applied, of the total remaining work to be executed under the equipartition mode of all jobs does not change. Thus between two consecutive intervals, either some job changes its parallelism or two jobs have the same amount of remaining work to be executed under the equipartition mode. We prove our claim by induction on the number of such intervals.

For the base case, the parallelism of all jobs is the same and the claim follows from our result on jobs with a single phase of parallelism in section 3. To apply the inductive proof, consider the execution of all jobs for τ time units in the interval during which no job changes its parallelism. The case with idle processors is trivial. So we assume there are no idle processors. For jobs in full-parallelism mode during this period of time (denoted by $\mathcal{J}_{para}$), their height decreases by τ. For jobs in equipartition mode during this period of time (denoted by $\mathcal{J}_{equi}$), their work decreases by $\bar{p}\tau$. Applying the induction hypothesis to the remaining portions after the first time interval, we have

$$\begin{aligned} TC_{DEQ}(\mathcal{J}) \leq n\tau + C \cdot \left[A(\mathcal{J}(e)) - \sum_{i=1}^{|\mathcal{J}_{equi}|} \frac{(n-i+1)\tau\bar{p}}{P} \right] \\ + C \cdot \left[H(\mathcal{J}(f)) - \sum_{i=1}^{|\mathcal{J}_{para}|} \tau \right], \end{aligned}$$

where $C = 2 - \frac{2}{n+1}$. The condition that the order of total work in $\mathcal{J}(e)$ does not change is crucial in the above formulation. At the end of the interval, it is possible that two jobs may be tied in the remaining portion of $\mathcal{J}(e)$. We can exchange their order without changing the squashed area bound. The new order would then be used for the next interval. To obtain our proof, it is sufficient to show

$$n\tau \leq C \cdot \sum_{i=1}^{|\mathcal{J}_{equi}|} \frac{(n-i+1)\tau\bar{p}}{P} + C \cdot |\mathcal{J}_{para}|\tau.$$

This holds if we have

$$n\tau \leq C \cdot \sum_{i=1}^{|\mathcal{J}_{equi}|} \frac{i\tau\bar{p}}{P} + C \cdot |\mathcal{J}_{para}|\tau,$$

which is equivalent to

$$nP \leq C \cdot \frac{|\mathcal{J}_{equi}|(|\mathcal{J}_{equi}|+1)\bar{p}}{2} + C \cdot |\mathcal{J}_{para}|P.$$

This can be shown in the same way as in the proof of Theorem 2.3 by applying Lemma 3.1. In fact, Lemma 3.1 holds independently of the fact that jobs contain a single phase of parallelism or multiple phases of parallelism. □

Interactive jobs can be formulated as jobs alternating between periods of being blocked while waiting for input from a user (requiring no processor), and periods of processing. Our result above also applies to such interactive jobs. We assume that users respond to each request for input in a finite amount of time. The degenerate case, where a user may not respond to an input request, is considered in detail in section 5. Thus, we have the following corollary.

COROLLARY 4.2. *DEQ is* $4 - \frac{4}{n+1}$ *competitive for the mean completion time of interactive jobs.* □

This result immediately carries over to sequential job scheduling problems and produces the same competitive ratio for the round-robin policy. However, in this case we have a better result.

THEOREM 4.3. *Round-robin is* $3 - \frac{2}{n+1}$ *competitive for the mean completion time of interactive jobs on sequential machines.*

Proof. Let $W(\mathcal{J}) = \sum_{i=1}^{n}(n-i+1)t_i$ for a job set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$, where t_i is the cumulative time job J_i requires CPU processing, $1 \leq i \leq n$, and $t_1 \leq t_2 \leq \cdots \leq t_n$. Let $H(\mathcal{J}) = \sum_{i=1}^{n} h_i$, where h_i is the cumulative time job J_i does not require CPU processing (i.e., when it is blocked). We now show

$$TC_{RR}(\mathcal{J}) \leq \left(2 - \frac{2}{n+1}\right) W(\mathcal{J}) + H(\mathcal{J}). \tag{4.2}$$

The theorem follows from (4.2) since both $W(\mathcal{J})$ and $H(\mathcal{J})$ are lower bounds for the optimal total completion time.

In a fashion similar to the above proof of Theorem 4.1, we use the round-robin policy to divide the execution of jobs into a finite number of intervals in which the order of remaining cumulative CPU times of jobs does not change and each job is in the same phase (ready to execute or be blocked). We apply an inductive proof to the number of such intervals. The base case follows from the result of Motwani, Phillips, and Torng [22]. Consider one such interval of length τ; let K be the set of jobs ready to execute, $k = |K|$. The case $k = 0$ is trivial and we thus assume $k \geq 1$. Each such job is executed for $\frac{\tau}{k}$ time units. The cumulative blocked time for each of the other jobs is decreased by τ. Thus we have

$$TC_{RR}(\mathcal{J}) \leq n\tau + \left(2 - \frac{2}{n+1}\right)\left[W(\mathcal{J}) - \sum_{i \in K}(n-i+1)\frac{\tau}{k}\right] + H(\mathcal{J}) - (n-k)\tau,$$

which is less than or equal to $(2 - \frac{2}{n+1})W(\mathcal{J}) + H(\mathcal{J})$ if

$$n \leq \left(2 - \frac{2}{n+1}\right)\sum_{i \in K}(n-i+1)\frac{1}{k} + (n-k).$$

This last inequality holds even for the worst choice of set K. Therefore, the theorem follows. □

5. Robustness of DEQ. Competitive analysis has been successfully applied in multiple processor scheduling problems to minimize the makespan [30]. There are, however, some objections to using makespan as a computer system's performance measure. One of them is that makespan does not distinguish one policy from another. Under our model, any work-conserving policy (i.e., no processors are idle as long as there are jobs available for execution [6]) has a competitive ratio of two, which is asymptotically optimal [1]. On the other hand, when there are new arrivals and the scheduler has no a priori information about the execution time, using mean response time as the performance objective, Motwani, Phillips, and Torng [22] have shown that no scheduling policy can achieve a performance ratio that is better than $n^{1/3}$. In this case, mean response time is a performance metric that is extremely difficult to minimize without any a priori information, since the adversary is able to choose arrival times and job sizes that can defeat any scheduler. Rather than using competitive analysis one might consider using mathematical analysis or simulation to compare various scheduling algorithms. In addition, Kellerer, Tautenhahn, and Woeginger [14] have shown that mean response time is also difficult to minimize even in the presence of complete job information. Unfortunately, these approaches require either knowing or making assumptions about job arrival and execution times. The power of competitive analysis is that a positive result applies without the need to understand or justify the workload model. That is, it applies for all workloads.

Since it is not possible to obtain a positive result for minimizing the competitive ratio for mean response times when there are new arrivals, as a compromise, we examine competitive scheduling policies for minimizing makespan. These policies are robust in the presence of infinite (possibly faulty) jobs. More precisely, we assume that there are up to K infinite jobs in the system but the scheduler does not know which jobs they are. Let $T_A(\mathcal{J})$ be the completion time of the last finished finite job under the scheduling policy A. Let $OPT(\mathcal{J})$ be the optimal completion time with full information about finite jobs (and information about which are infinite jobs). Notice that from now on, $OPT(\mathcal{J})$ refers to the optimal makespan instead of mean completion time. Obviously, infinite jobs are not executed under the optimal scheduler. The competitive ratio is defined as $\max_{\mathcal{J}} \frac{T_A(\mathcal{J})}{OPT(\mathcal{J})}$.

Since the same issue arises in uniprocessor systems, we first consider this case.

THEOREM 5.1. *In a system with K infinite jobs, when there are new arrivals, the competitive ratio of the makespan of the round-robin policy is $K+1$, which is optimal.*

Proof. To show that (K+1) is a lower bound on the competitive ratio, consider a case of $K+1$ jobs including K infinite jobs and one finite job with execution time t. No matter what scheduling algorithm is used, the adversary always assigns the finite job to execute last. Thus, the total time required to complete the finite job is at least $(K+1)t$. An optimal schedule with complete information will take only t time units, running only the finite job in t time units and not even running the infinite jobs. Therefore, the competitive ratio is at least $K+1$ for any scheduling algorithm.

For the upper bound, without loss of generality, we assume that all K faulty jobs are present in the system at time t_0, and there are N other jobs, whose execution time is $x_1, x_2, \ldots, x_N$, arriving at time $t_1, t_2, \ldots, t_N$, respectively. We consider the following two cases.

Under the optimal scheduling algorithm with complete information, if new jobs always arrive before the last finite job in the system has finished, no processor will be left idle, and the optimal makespan will be $\sum_{i=1}^{N} x_i$. A round-robin scheduler will

always execute at least $\frac{1}{K+1}$ of the finite jobs, and all of the finite jobs will complete by time $\frac{1}{K+1}\sum_{i=1}^{N} x_i$. Therefore, the competitive ratio is bounded above by $K+1$ in this case.

The second case to consider under the optimal scheduling algorithm is when new jobs arrive some time after the last finite job in the system has finished execution. Let indices be defined according to job arrival orders. In this case, denote by m the maximum index when a job arrives at time t_m and finds no remaining jobs in the system (all previous jobs have completed their execution). Let $t^* = t_m$. In this case, the optimal completion time will be $OPT = t^* + \sum_{i=m}^{N} x_i$. Consider the last consecutive interval $[\tau_1, \tau_2]$, $0 \le \tau_1 < \tau_2 \le t^*$, during which there is at least one finite job executing under the round-robin policy. The total length of finite jobs arriving during this interval will be no more than $\tau_2 - \tau_1$, since an optimal scheduler (with complete information) will finish all of them. Since the round-robin policy will assign at least $\frac{1}{K+1}$ of the total processing power to one finite job whenever there is one in the system, by time τ_2 the remaining total length of jobs is no more than $\frac{K}{K+1}(\tau_2 - \tau_1) \le \frac{K}{K+1}t^*$. From then on, the processor will always devote a fraction (at least $\frac{1}{K+1}$) of its time to finite jobs. Therefore, round-robin will finish all of the finite jobs within at most $(K+1)(\frac{K}{K+1}t^* + \sum_{i=m}^{N} x_i)$ time units after t^*. Thus the completion time of all finite jobs will be at most $(K+1)(t^* + \sum_{i=m}^{N} x_i)$, which is $(K+1)OPT$. Therefore, the round-robin policy achieves a competitive ratio of $K+1$. □

A similar argument can be applied to parallel job scheduling on multiprocessors.

THEOREM 5.2. *In a multiprocessor system with new job arrivals with multiple phases of parallelism and K infinite jobs, the competitive ratio for the makespan of DEQ is $K+1$, which is the best possible competitive ratio.*

Proof. Without loss of generality, assume that all K faulty jobs are present in the system at time $t_0 = 0$, and there are N other finite jobs, $J_1, J_2, \ldots, J_N$, which arrive at time $t_1, t_2, \ldots, t_N$, respectively. Similarly, we examine the last finished finite job J_i according to the DEQ policy. Again, we divide the execution of this job J_i into two parts: let t_{para} be the total time during which the number of processors allocated to the job is equal to its parallelism, and let t_{equi} be the total time during which it is assigned its fair share of processors according to DEQ. Let W_{equi} be the total amount of work executed by J_i during the period it is assigned its fair share of processors (i.e., during t_{equi}). Since there are at most K infinite jobs, the total amount of work performed on the infinite jobs during the period when J_i is assigned its fair share of processors is no more than KW_{equi}. Let W' be the total work performed on finite jobs during the same period. Then the completion time of DEQ is bounded by $t_i + t_{para} + \frac{KW_{equi}+W'}{P}$. On the other hand, for the optimal completion time, we have $t_i + t_{para} + \frac{W_{equi}}{P} \le OPT$, which is the minimum time to complete J_i, ignoring all other jobs. Furthermore, $\frac{(K-1)W_{equi}}{P} \le (K-1)OPT$, and $\frac{W'}{P} \le OPT$. This concludes our proof that DEQ has a competitive ratio of $K+1$ when the system has new arrivals of jobs with multiple phases of parallelism. □

6. Remarks and discussion. We started our work on competitive analysis for the parallel scheduling problem by facing two obstacles with this approach: the lower bound of Deng and Koutsoupias on scheduling an arbitrary DAG [4] and the lower bound of Motwani, Phillips, and Torng for scheduling with new job arrivals. While the former points out that it is impossible to obtain a general on-line strategy that

schedules arbitrary jobs on parallel machines (with a given communication latency) near-optimally, the latter points out that it is impossible to obtain a general online preemptive strategy that schedules sequential jobs on a uniprocessor to minimize mean response time if job arrivals are unpredictable [22]. (This result can also be extended to parallel jobs.) These two results raise serious doubts about the possibility of obtaining a near-optimal scheduling strategy in realistic computing environments for parallel jobs with new arrivals.

In this paper, we avoid the first difficulty by utilizing a special class of parallel jobs. This class includes jobs with sublinear speedups by modeling them using multiple phases of parallelism. With Edmonds and Chinn we have also recently extended some of our positive results to larger classes of jobs and to models of systems which more explicitly account for communication delays [7]. It seems that the second difficulty may be much harder to overcome since it holds even for single processor scheduling. If the objective of computer system scheduling is indeed to minimize mean response time, more empirical workload studies will help to understand arrival times and job sizes in typical computer systems. (Recent work has begun to examine workload characteristics on multiprocessor systems [8], [11].) Workload information could be used to more accurately reflect typical arrival times and job sizes and to perhaps avoid scenarios that prevent competitive ratios from being obtained for algorithms that perform well in practice. Alternatively, we may study other objective functions. The explicit definition of interactive jobs introduced in this paper and the related constant competitive ratio (for minimizing makespan) is an effort in this direction. Under these conditions (including the presence of infinite jobs), the DEQ policy stands out among other parallel scheduling policies in that it achieves the optimal competitive ratio for makespan. An alternative approach to this second difficulty has been proposed by Kalyanasundaram and Pruhs [12]. They show that a moderate increase in the speed of the processor used by a nonclairvoyant scheduler can effectively give this processor power equal to clairvoyance.

A central question that must be considered when developing and comparing scheduling algorithms for multiprocessors is, What is the objective function being used to determine how well the algorithm is performing? For example, is it more desirable to minimize mean response time than to minimize makespan, or should maximizing throughput be the main goal of the scheduler? As well, real systems must also be careful to provide quick turnaround time to interactive programs. Additional consideration must also be given to the fact that in some cases knowing or deriving a competitive ratio for an algorithm does not mean that the complexity of the algorithm is acceptable or that an algorithm can be easily constructed. These issues are quite similar to solution concepts in the game theoretical framework for the study of sharing economic resources, and it might be interesting to explore possible links between them [5].

Acknowledgment. We would like to thank the anonymous referees whose comments helped to significantly improve this paper.

REFERENCES

[1] T. Brecht, X. Deng, and N. Gu, *Competitive dynamic processor allocation for parallel applications*, Parallel Process. Lett., 7 (1997), pp. 89–100.

[2] T. Brecht, and K. Guha, *Using parallel program characteristics in dynamic processor allocation policies*, Performance Evaluation, 27–28 (1996), pp. 519–539.

[3] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian,

AND T. VON EICKEN, *LogP: Towards a realistic model of parallel computation*, in Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, New York, 1993, pp. 1–12.

[4] X. DENG AND E. KOUTSOUPIAS, *Competitive implementation of parallel programs*, in Proceedings of the Fourth Annual ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1993, pp. 455–461.

[5] X. DENG AND C. PAPADIMITRIOU, *On the complexity of cooperative game solution concepts*, Math. Oper. Res., 19 (1994), pp. 257–266.

[6] D. EAGER, J. ZAHORJAN, AND E. LAZOWSKA, *Speedup versus efficiency in parallel systems*, IEEE Trans. Comput., 38 (1989), pp. 408–423.

[7] J. EDMONDS, D. CHINN, T. BRECHT, AND X. DENG, *Non-clairvoyant multiprocessor scheduling of jobs with changing execution characteristics*, in Proceedings of the 22nd ACM Symposium on the Theory of Computing, ACM, New York, 1997, pp. 120–129.

[8] D.G. FEITELSON AND B. NITZBERG, *Job characteristics of a production parallel scientific workload on the NASA AMES iPSC/*860, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 1291, Springer-Verlag, Berlin, New York, Heidelberg, 1997, pp. 1–34.

[9] D.G. FEITELSON, L. RUDOLPH, U. SCHWIEGELSHOHN, K.C. SEVCIK, AND P. WONG, *Theory and practice in parallel job scheduling*, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 949, Springer-Verlag, Berlin, New York, Heidelberg, 1995, pp. 337–360.

[10] A. GUPTA, A. TUCKER, AND S. URUSHIBARA, *The impact of operating system scheduling policies and synchronization methods on the performance of parallel applications*, in Proceedings of the ACM SIGMETRICS Conference on the Measurement and Modeling of Computer Systems, New York, 1991, pp. 120–132.

[11] S. HOTVOY, *Workload evolution on the Cornell theory center IBM SP*2, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 1162, Springer-Verlag, Berlin, New York, Heidelberg, 1996, pp. 27–40.

[12] B. KALYANASUNDARAM AND K. PRUHS, *Speed is as powerful as clairvoyance*, in Proceedings of the 36th Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 214–221.

[13] A. KARLIN, M. MANASSE, L. RUDOLPH, AND D. SLEATOR, *Competitive Snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[14] H. KELLERER, T. TAUTENHAHN, AND G.J. WOEGINGER, *Approximability and nonapproximability results for minimizing total flowtime on a single machine*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, ACM, New York, 1996, pp. 418–426.

[15] M. KUMAR, *Measuring parallelism in computation-intensive scientific/engineering applications*, IEEE Trans. Comput., 37 (1988), pp. 1088–1098.

[16] S. LEUTENEGGER AND R. NELSON, *Analysis of Spatial and Temporal Scheduling Policies for Semi-Static and Dynamic Multiprocessor Environments*, Technical Report RC 17086 75594, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1991.

[17] S. LEUTENEGGER AND M. VERNON, *The performance of multiprogrammed multiprocessor scheduling policies*, in Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1990, pp. 226–236.

[18] W. LUDWIG AND P. TIWARI, *The Power of Choice in Scheduling Parallel Tasks*, Report CS TR1190, Computer Science Department, University of Wisconsin, Madison, WI, 1993.

[19] M. MANASSE, L. MCGEOCH, AND D. SLEATOR, *Competitive algorithms for on-line problems*, in Proceedings of the 30th ACM Annual Symposium on the Theory of Computing, ACM, New York, 1988, pp. 322–333.

[20] C. MCCANN, R. VASWANI, AND J. ZAHORJAN, *A dynamic processor allocation policy for multiprogrammed shared memory multiprocessors*, ACM Trans. Comput. Systems, 11 (1993), pp. 146–178.

[21] C. MCCANN AND J. ZAHORJAN, *Scheduling memory constrained jobs on distributed memory parallel computers*, in Proceedings of the International Joint Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS 95 and Performance 95, ACM, New York, 1995, pp. 208–219.

[22] R. MOTWANI, S. PHILLIPS, AND E. TORNG *Non-clairvoyant scheduling*, Theoret. Comput. Sci., 130 (1994), pp. 17–47.

[23] T. NGUYEN, R. VASWANI, AND J. ZAHORJAN, *Using runtime measured workload characteristics in parallel processor scheduling*, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 1162, Springer-Verlag,

Berlin, New York, Heidelberg, 1996, pp. 175–199.

[24] T. Nguyen, R. Vaswani, and J. Zahorjan, *Maximizing speedup through self-tuning of processor allocation*, in Proceedings of the 10th International Parallel Processing Symposium, Honolulu, HI, 1996, pp. 463–468.

[25] C. Papadimitriou and M. Yannakakis, *Towards an architecture-independent analysis of parallel algorithms*, in Proceedings of the 20th ACM Symposium on the Theory of Computing, ACM, New York, 1988, pp. 510–513.

[26] E. Parsons and K. Sevcik, *Multiprocessor scheduling for high-variability service time distributions*, in Job Scheduling Strategies for Parallel Processing, D.G. Feitelson and L. Rudolph, eds., Lecture Notes in Comput. Sci. 949, Springer-Verlag, Berlin, New York, Heidelberg, 1995, pp. 127–145.

[27] U. Schwiegelshohn, W. Ludwig, J.L. Wolf, J. Turek, and P.S. Yu, *Smart SMART bounds for weighted response time scheduling*, SIAM J. Comput., 28 (1999), pp. 237–253.

[28] K. Sevcik, *Application scheduling and processor allocation in multiprogrammed multiprocessors*, Performance Evaluation, 9 (1994), pp. 107–140.

[29] K. Sevcik, *Characterizations of parallelism in applications and their use in scheduling*, in Proceedings of the 1989 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1989, pp. 171–180.

[30] D.B. Shmoys, J. Wein, and D.P. Williamson, *Scheduling parallel machines on-line*, SIAM J. Comput., 24 (1995), pp. 1313–1331.

[31] A. Silberschatz and P. Galvin, *Operating System Concepts*, Addison–Wesley, New York, 1994.

[32] D. Sleator and R. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. Assoc. Comput. Mach., 28 (1985), pp. 202–208.

[33] A. Tucker and A. Gupta, *Process control and scheduling issues for multiprogrammed shared-memory multiprocessors*, in Proceedings of the 12th ACM Symposium on Operating Systems Principles, ACM, New York, 1989, pp. 159–166.

[34] J. Turek, W. Ludwig, J.L. Wolf, L. Fleischer, P. Tiwari, J. Glasgow, U. Schwiegelshohn, and P. Yu, *Scheduling parallelizable tasks to minimize average response time*, in Proceedings of the Sixth Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1994, pp. 200–209.

[35] J. Turek, U. Schwiegelshohn, J.L. Wolf, and P. Yu, *Scheduling parallel tasks to minimize average response time*, in Proceedings of the Fifth ACM–SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1994, pp. 112–121.

[36] L. Valiant, *A bridging model for parallel computation*, Comm. Assoc. Comput. Mach., 33 (1990), pp. 103–111.

[37] C. Wu, *Processor Scheduling in Multiprogrammed Shared-Memory NUMA Multiprocessors*, M.Sc. thesis, University of Toronto, Toronto, ON, Canada, October, 1993.

[38] J. Zahorjan and C. McCann, *Processor scheduling in shared memory multiprocessors*, in Proceedings of the 1990 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, ACM, New York, 1990, pp. 214–225.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 161–190

NONUNIFORM DISCRETIZATION FOR KINODYNAMIC MOTION PLANNING AND ITS APPLICATIONS*

JOHN H. REIF† AND HONGYAN WANG†

Abstract. The first main result of this paper is *a novel nonuniform discretization approximation method for the kinodynamic motion-planning problem.* The kinodynamic motion-planning problem is to compute a collision-free, time-optimal trajectory for a robot whose accelerations and velocities are bounded. Previous approximation methods are all based on a uniform discretization in the time space. On the contrary, our method employs a nonuniform discretization in the configuration space (thus also a nonuniform one in the time space). Compared to the previously best algorithm of Donald and Xavier, the running time of our algorithm reduces in terms of $1/\varepsilon$, roughly from $O((1/\varepsilon)^{6d-1})$ to $O((1/\varepsilon)^{4d-2})$, in computing a trajectory in a d-dimensional configuration space, such that the time length of the trajectory is within a factor of $(1+\varepsilon)$ of the optimal. More importantly, our algorithm is able to take advantage of the obstacle distribution and is expected to perform much better than the analytical result. This is because our nonuniform discretization has the property that it is coarser in regions that are farther from all obstacles. So for situations where the obstacles are sparse, or the obstacles are unevenly distributed, the size of the discretization is significantly smaller.

Our second main result is the *first known polynomial-time approximation algorithm for the curvature-constrained shortest-path problem in three and higher dimensions.* We achieved this by showing that the approximation techniques for the kinodynamic motion-planning problem are applicable to this problem.

Key words. robotic motion planning, nonholonomic motion planning, kinodynamic motion planning, nonuniform discretization

AMS subject classifications. 68W25, 68W40, 65K10

PII. S0097539798331975

1. Introduction.

Nonholonomic motion planning involves planning a collision-free path (or trajectory) for a robot subject to *nonholonomic* constraints on its dynamics. A *holonomic* constraint is one that can be expressed as an equation of the robot's configuration parameters (a placement of a robot with k degrees of freedom can be uniquely specified by k such parameters), while a *nonholonomic* one can only be expressed as a *nonintegrable* equation involving also the derivatives of the configuration parameters (see [30] for a more detailed discussion on nonholonomic constraints). Examples of nonholonomic constraints are bounds on velocities, accelerations, and curvatures. Although there has been considerable recent work in the robotics literature (see [2, 3, 4, 5, 6, 7, 19, 24, 27, 28, 29, 31, 33, 34, 39, 42, 45, 47, 48] and references therein) on nonholonomic motion-planning problems, relatively little theoretical work has been done on these problems. Nonholonomic motion planning is considerably harder than holonomic. For one thing, a robot with k degrees of freedom cannot be described completely by k parameters. A complete description has to include the k parameters and their derivatives. The *configuration-space* approach, which is widely used for the holonomic motion-planning problems, does not apply to the problems with nonholonomic constraints because such constraints are not expressed by the

*Received by the editors December 31, 1997; accepted for publication (in revised form) March 18, 1999; published electronically April 25, 2000. This work was supported by NSF grant NSF-IRI-91-00681, Rome Labs contracts F30602-94-C-0037, ARPA/SISTO contracts N00014-91-J-1985, and N00014-92-C-0182 under subcontract KI-92-01-0182.

http://www.siam.org/journals/sicomp/30-1/33197.html

†Department of Computer Science, Duke University, Box 90129, Durham, NC 27708-0129 (reif@cs.duke.edu, hongyan@cs.duke.edu).

configuration-space representation. On the other hand, these problems bear major significance in robotic engineering. In reality, a robot arm has to move not only in a collision-free fashion, but also in conformation of the dynamic bounds due to limited force or torque from motors.

It is also desirable, in many cases, to minimize the cost of a motion plan under some cost function (e.g., path length, time length, number of turns, clearance, etc.). Finding an optimal solution is significantly harder than just finding any solutions. For example, the seemingly easy problem of finding the shortest path for a point robot in three dimensions among polyhedral obstacles is already NP-hard (see Canny and Reif [10]), even without constrained dynamics.

In this paper, we study two optimal nonholonomic motion-planning problems: the *kinodynamic motion-planning* problem and the *curvature-constrained shortest-path* problem. The kinodynamic motion-planning problem studies the problem of finding collision-free time-optimal trajectories for a robot whose motion is governed by Newtonian dynamics and whose accelerations and velocities are bounded. A *trajectory* of a robot with d degrees of freedom is a map $\Gamma\colon [0,T] \to \mathbb{R}^d \times \mathbb{R}^d$ given by $\Gamma(t) = (p(t), \dot{p}(t))$, where $p(t)$ and $\dot{p}(t)$ give the location and the velocity at time t, respectively, in a d-dimensional configuration space. $\ddot{p}(t)$ is the acceleration function, which determines a trajectory uniquely once an initial state is fixed. The constraints on dynamics are given by bounding the norms of the accelerations and velocities. The most studied norms are the L_∞-norm (called the *decoupled* case) and the L_2-norm (called the *coupled* case). In this paper, we study the coupled kinodynamic problem, which happens to be harder than the decoupled one. The decoupled case is simpler because each dimension is independent of the others, and a d-dimensional problem can be reduced to a one-dimensional (1D) one.

The curvature at one point on a path describes how fast the direction of the path changes at that point. Generally, a curvature constraint requires that a path has a curvature of at most c at every point along the path, where $c > 0$ is a given parameter. The curvature-constrained shortest-path problem is to compute a shortest collision-free path such that the path satisfies the given curvature constraint.

Our major contribution in this paper is a *nonuniform* discretization approximation method for the kinodynamic motion-planning problem. The discretization is nonuniform in the sense that it is coarser in regions which are farther from all obstacles. The intuition behind this is that in regions that are far away from all obstacles, even with a coarse discretization, we are still able to find an approximation trajectory which does not intersect obstacles. The nonuniform discretization on one hand reduces the search space and the running time, and on the other hand still enables us to obtain a collision-free trajectory whose time length is within a given factor of the optimal time length. Nonuniform discretization is widely used in solving PDEs (see Miller et al. [35, 36] and references therein), but research in that area focuses on quite different issues. Applying the idea to kinodynamic motion planning was first suggested by Xavier [49] but without a rigorous proof. As it happens, provably good bounds given by us are quite intriguing to obtain.

Our nonuniform discretization is based on a box decomposition of the configuration space. Other geometric algorithms using similar box decompositions include the work of Mitchell, Mount, and Suri [38] on ray-shooting problems and that of Hershberger and Suri [21] on two-dimensional (2D) shortest-path problems without constraints. Using nonuniform discretization for geometric planning with nonholonomic constraints presents a new challenge.

Instead of using a deterministic discretization, Kavraki et al. [25, 26] developed a *random sampling technique*, where in preprocessing, grid-points (called *milestones*) are chosen *randomly* and are connected by feasible paths to form a network. The method is similar to ours in that both do a preprocessing to obtain a set of valid paths (or trajectories) which are used later in answering planning queries. However, they have to preprocess for each new environment, where in our method, the preprocessing is only done once for some fixed parameters, and can be used repeatedly for different environments.

1.1. Previous work on the kinodynamic motion-planning problem. With the exceptions of 1D and 2D cases (see Ó'Dúnlaing [40] and Canny, Rege, and Reif [9]), there are no exact solutions for the kinodynamic motion-planning problem. In fact, as an implication of the result of [10], the problem is at least NP-hard in three and higher dimensions. In light of this lower bound, most study has been focusing on finding approximation solutions. The earlier approximation algorithms of Sahar and Hollerbach [44] did not guarantee goodness of their solutions. Moreover, the running time was exponential in the resolution. Canny et al. [8, 16] developed the first provably good, polynomial-time approximation algorithm for the decoupled kinodynamic case. Their work was followed up by a series of work in which Donald and Xavier [11, 15] improved the running time for the decoupled case, Heinzinger et al. [20] and Donald and Xavier [13, 14] investigated the problem for open chain manipulators, and independent work of Donald and Xavier [12, 13, 15] and Reif and Tate [43] gave approximation algorithms for the coupled kinodynamic problem. The best-known result for the coupled case, given by [15], is that given an $\varepsilon > 0$, one can compute in time $O(c(d)p(n, \varepsilon, d)L^d(1/\varepsilon)^{6d-1})$ an approximation trajectory whose time length is at most $(1+\varepsilon)$ times the time length of an optimal safe trajectory (roughly speaking, a safe trajectory is one that can be perturbed without intersecting obstacles), where d is the dimension, L is the size of the configuration space to which the robot is confined, n is the number of equations describing the configuration obstacles, $c(d)$ is a function depending solely on d, and $p(n, \varepsilon, d)$ is a lower-order polynomial in n, ε, and d.

1.2. Previous work on the curvature-constrained shortest-path problem. Dubins [17] was perhaps the first to study the curvature-constrained shortest paths who gave a characterization of the shortest paths in 2D in the absence of obstacles. Reeds and Shepp [41] extended the obstacle-free characterization to robots that can make reversals. (Boissonnat, Cerezo, and Leblond [4] gave an alternative proof for both cases, using ideas from control theory.) In the presence of obstacles, Fortune and Wilfong [18] gave a $2^{poly(n,m)}$-time algorithm, where n is the number of vertices and m is the number of bits of precision with which all points are specified; their algorithm only decides whether a path exists, without necessarily finding one. Jacobs and Canny [23] gave an $O((\frac{n+L}{\varepsilon})^2 + (\frac{n+L}{\varepsilon})n^2 \log n)$-time algorithm that computes an approximation path whose length is at most $(1+\varepsilon)$ times the length of an optimal path, where n is the number of obstacle vertices and L is the total edge length of the obstacles. This running time was later improved significantly by Wang and Agarwal [46] to $O((n^2/\varepsilon^2)\log n)$. Agarwal, Raghavan, and Tamak [1] studied a special case when the boundaries of obstacles are also constrained to have a curvature of at most 1. There has also been work on computing curvature-constrained paths when the robot is allowed to make reversals [2, 32, 37]. However, no result has been obtained for the curvature-constrained problem in three or higher dimensions, even in the absence of obstacles.

1.3. Models and results. Let B be a point robot moving in a d-dimensional configuration space. The velocity of B is bounded by $v_{\max}$ in L_2-norm and the acceleration of B is bounded by $a_{\max}$ in L_2-norm, where $v_{\max} > 0$ and $a_{\max} > 0$ are arbitrary. In section 2.7, we show that we can always scale $v_{\max}$ and $a_{\max}$ to 1 by scaling both time and the size of the configuration space. Therefore, without loss of generality, we will set $v_{\max} = 1$ and $a_{\max} = 1$.

A *state* x of B is a pair $(\text{LOC}(x), \text{VEC}(x))$, where $\text{LOC}(x)$ is a point representing the location of B in the d-dimensional configuration space and $\text{VEC}(x)$ is a vector representing the velocity of B. A trajectory is an $(\bar{a}, \bar{v})$-trajectory if at any time during the trajectory, the acceleration is bounded by $\bar{a}$ and the velocity is bounded by $\bar{v}$, both in L_2-norm. Thus only $(1, 1)$-trajectories are valid trajectories for robot B. Let Ω be a set of configuration obstacles. A trajectory is *collision free* if its path does not intersect the interior of Ω. A trajectory from state x to state y is *optimal* if it is a collision-free $(1, 1)$-trajectory with a minimum time length, where the minimum is taken over all collision-free $(1, 1)$-trajectories from x to y. The kinodynamic motion-planning problem is to compute such optimal trajectories.

Since we have seen that computing exact solutions is hard, we focus on developing fast approximation algorithms. To discuss approximation solutions, the notion of a safe trajectory needs to be introduced. A point p has a *clearance* of μ if for any point $q \in \Omega$, $\|p - q\|_\infty \geq \mu$, where $\|\cdot\|_\infty$ denotes the L_∞-norm operation. A path is *μ-safe* if for any point p along the path, p has a clearance of μ. A trajectory is a *μ-safe trajectory* if its path is μ-safe. A trajectory is an *optimal μ-safe* trajectory from state x to state y if its time length is the minimum over all μ-safe $(1, 1)$-trajectories from x to y.

The first main result of this paper is a faster algorithm for computing approximations to optimal safe trajectories. Let W be a d-dimensional configuration space of size L (without loss of generality, we can assume that W is a d-dimensional cube), where the point robot is confined to move. Let Ω be a set of configuration obstacles defined by a total of n algebraic equations, each of $O(1)$ degrees. Given the initial and the final states i and f and two parameters $l > 0$ and $\varepsilon > 0$, we can compute an approximation to the optimal $3l$-safe $(1, 1)$-trajectory from i to f in time $O(nN + N \log N(1/\varepsilon)^{4d-2})$, where $N = O((L/l)^d)$. The time length of the computed trajectory is at most $(1 + \varepsilon)$ times the time length of the optimal trajectory. The computed trajectory connects two states which are close to i and f, respectively. More precisely, if the computed trajectory is from state i' to state f', then $\|\text{LOC}(i') - \text{LOC}(i)\|_\infty \leq \varepsilon l$, $\|\text{VEC}(i') - \text{VEC}(i)\|_\infty \leq \varepsilon$, $\|\text{LOC}(f') - \text{LOC}(f)\|_\infty \leq \varepsilon l$, and $\|\text{VEC}(f') - \text{VEC}(f)\|_\infty \leq \varepsilon$.

Here we introduce two parameters ε and l, which are independent except that in general $\varepsilon < l$. ε describes how close the approximation should be to the optimal safe trajectory in time, while l describes how safe the optimal trajectory is. If we require that l be as small as ε, the running time of our algorithm in terms of ε is roughly $O((1/\varepsilon)^{5d-2})$, which improves over the running time $O((1/\varepsilon)^{6d-1})$ of the previously best algorithm of Donald and Xavier. If we choose a big l, our algorithm may fail to find a trajectory because $3l$-safe trajectories connecting the initial state to the final state may not exist due to, for example, the crowded obstacles. However, we can still choose the parameters such that $l \gg \varepsilon$, with both parameters being small. In this case, the running time of our algorithm in terms of ε is $O((1/\varepsilon)^{4d-2})$. Moreover, because of our nonuniform discretization, there are cases when the number of boxes is much smaller than the worst-case bound of $N = O((L/l)^d)$. In these cases, our

algorithm is expected to perform much better than the given time bound. Such cases include sparse obstacle distributions and uneven obstacle distributions. As will be pointed out in the conclusion section, one future work is to look for more accurate bounds on the number of boxes with respect to different obstacle distributions.

By showing that the approximation techniques for kinodynamic motion planning are applicable to the curvature-constrained shortest-path problem, we are able to obtain the second main result of this paper—the first known polynomial-time approximation algorithm for the curvature-constrained shortest-path problem in three and higher dimensions. The detailed model and result are presented in section 3.

2. The kinodynamic motion-planning problem.

2.1. Preliminaries. In this paper, we generally use $\|\cdot\|_\infty$ to denote the L_∞-norm operation and $\|\cdot\|$ the L_2-norm one.

Given a trajectory Γ, let $T(\Gamma)$ be the time length of Γ. Let $p_\Gamma(t), v_\Gamma(t), a_\Gamma(t)$ for $0 \le t \le T(\Gamma)$ be the location, the velocity, and the acceleration of Γ, respectively, at time t.

Let Π be a path and A a subset of the configuration space. We say that $d(\Pi, A) \le \rho$ for some real number $\rho \ge 0$ if for every point $p \in \Pi$, there is a point $q \in A$ such that $\|p - q\|_\infty \le \rho$. $d(\Pi, A) = 0$ if every point $p \in \Pi$ is also in A. Similarly, for two paths Π and Π', we say that $d(\Pi', \Pi) \le \rho$ if for every point $p \in \Pi'$, there is a point $q \in \Pi$ such that $\|p - q\|_\infty \le \rho$. Let Γ and Γ' be two trajectories and let Π and Π' be the paths of Γ and Γ', respectively. $d(\Gamma', \Gamma) \le \rho$ if $d(\Pi', \Pi) \le \rho$. Also $d(\Gamma', A) \le \rho$ if $d(\Pi', A) \le \rho$, where A is a subset of the configuration space.

For a state x, define

$$ngb(x, \rho, \nu) = \{x' \mid \|\textsc{loc}(x') - \textsc{loc}(x)\|_\infty \le \rho \text{ and } \|\textsc{vec}(x') - \textsc{vec}(x)\|_\infty \le \nu\}$$

for some $\rho, \nu > 0$. Thus $ngb(x, \rho, \nu)$ defines a set of states which are close to x. Notice that if $x' \in ngb(x, \rho, \nu)$, then $x \in ngb(x', \rho, \nu)$.

Lemma 2.1 (time-rescaling lemma, Hollerbach [22]). *Let Π be the path of an $(\bar{a}, \bar{v})$-trajectory Γ. Π can be traversed by an $(\bar{a}/(1+\varepsilon)^2, \bar{v}/(1+\varepsilon))$-trajectory Γ' in time $(1+\varepsilon)\,T(\Gamma)$ for any $\varepsilon > 0$.*

The time-rescaling lemma shows a trade-off between time and the dynamic bounds. Later we will see that in our approximation algorithm, we first compute a trajectory whose acceleration and velocity bounds are slightly bigger than 1. Then applying the time-rescaling lemma, we can reduce the dynamic bounds to 1 by sacrificing the time length by a small factor.

The TC-graph method developed in [13] by Donald and Xavier applies a graph-searching technique to compute approximations to time-optimal trajectories (for a brief description of the method, see the appendix). The following corollary states a result of the TC-graph method.

Corollary 2.2 (see [13]). *Let W be a d-dimensional configuration space of size L. Let Γ be a $(1,1)$-trajectory from state i to state f such that Γ lies inside W. Given any $\varepsilon > 0$ and $\rho = O(1)$, applying the TC-graph method with appropriate parameters, we can compute in time $O(L^d(1/\varepsilon)^{6d-1})$ a $(1,1)$-trajectory Γ' from a state i' to a state f' such that $i' \in ngb(i, \varepsilon\rho/2, \varepsilon/2)$, $f' \in ngb(f, \varepsilon\rho/2, \varepsilon/2)$, $T(\Gamma') \le (1+\varepsilon)\,T(\Gamma)$, and $d(\Gamma', W) \le \varepsilon\rho/2$.*

The above corollary states that by applying the TC-graph method, one can compute in polynomial time (in terms of L and ε) a trajectory which obeys the dynamic constraints and which approximates the given trajectory in the following ways. The

computed trajectory connects two states which are close to the given initial and final states, respectively. The time length of the computed trajectory is within $(1+\varepsilon)$ times that of the given trajectory. Also the path of the computed trajectory stays close to the region W, within which the path of the given trajectory lies.

We will see that in later sections we apply the TC-graph method to precompute a set of trajectories which in turn become the building-block trajectories in our approximation method. Here we present the result for cases where there are no obstacles. This is sufficient for our purpose, though the TC-graph method works also in the presence of obstacles.

In the rest of the paper, we fix $l > 0$ and $\epsilon > 0$ unless otherwise stated, where l gives the size of the smallest box in the box decomposition and ϵ describes the fineness of the discretization.

2.2. Overview of the algorithm. Following the framework of many motion-planning algorithms, our algorithm reduces the problem to that of computing and then searching on a graph. The nodes of the graph correspond to a set of states (see section 2.3) which are induced by a nonuniform discretization over the configuration space. Therefore, we term our method the CS-graph method, where C stands for the configuration space and S the state space.

Section 2.5 describes how to compute the edge set of the graph. Each edge corresponds to a collision-free, near-optimal trajectory between two nodes. It is shown in this section that the graph satisfies the property that if there is an optimal safe trajectory, then there exists a path in the graph that corresponds to a collision-free trajectory whose time length is near optimal. Thus the problem is reduced to a shortest-path search on the computed graph.

In order to compute the edge set efficiently, we perform precomputations. We derive a set of canonical trajectories (section 2.4), which are sufficient for building the graph. These trajectories, once computed, can be used repeatedly for different problem instances with different distributions of obstacles.

A technical lemma, the correcting lemma (Lemma 2.15), is described in section 2.6. This lemma is a precise statement of our intuition that the discretization can be coarser in regions farther away from all obstacles.

2.3. Nonuniform grids. The set of nonuniform grids is generated based on a box decomposition of the configuration space (see section 2.3.1). The decomposition is such that the size of a box is roughly proportional to the smallest distance between the box and the obstacles. Each box contributes the same number of grids, despite its size; the distances among grids induced by larger boxes are larger. Thus the grids are nonuniform (see section 2.3.2).

2.3.1. A box decomposition. Let $\tilde{\Xi}(s)$ be a set $\{p = (p_1, \ldots, p_d) \mid -s/2 \le p_1, \ldots, p_d \le s/2\}$. $\tilde{\Xi}(s)$ is called a d-dimensional *canonical box* of size s. Each set of $\{p \mid p \in \tilde{\Xi}(s) \text{ and } p_j = -s/2\}$ and $\{p \mid p \in \tilde{\Xi}(s) \text{ and } p_j = s/2\}$ for $1 \le j \le d$ is called a *face* (of size s) of $\tilde{\Xi}(s)$. Ξ is a d-dimensional *box* of size s if it is a region which can be obtained from $\tilde{\Xi}(s)$ by translation and rotation (reflection is not necessary because of the symmetry of the box). Similarly we can define the faces of Ξ. Notice that a face of a d-dimensional box Ξ of size s is really a $(d-1)$-dimensional box of size s. A d-dimensional box has $2d$ faces.

Given a d-dimensional box of size s, we can decompose it into 2^d boxes, each of size $s/2$. Each of them can be further decomposed into 2^d boxes of size $s/4$ and so on. We stop further decomposing until certain conditions hold. We refer to this procedure,

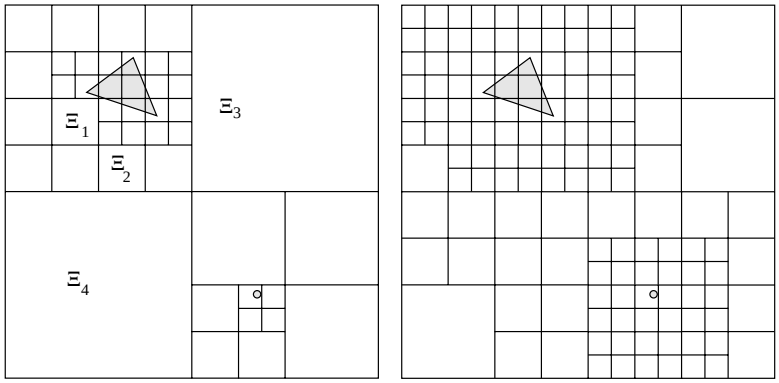


FIG. 2.1. (a) *A box decomposition satisfying* C1; (b) *one also satisfying* C2 *and* C3.

as well as the collection of nondecomposed boxes obtained, as a *box decomposition*. When we refer to a box of a decomposition, we mean a nondecomposed box.

Let W be a d-dimensional configuration space (assuming without loss of generality that W is a d-dimensional box) of size L, with a set Ω of configuration obstacles. (We consider the initial and final locations also as obstacle vertices.) A box is *free* if it does not intersect the interior of Ω. A box is *occupied* if it is a subset of Ω. Otherwise a box is called *partially occupied*. A box B_1 is *adjacent* to a box B_2 (or B_1 and B_2 are *neighbors*) if B_1 and B_2 share common points. We perform a box decomposition on W until the following conditions are satisfied:

C1. The size of every partially occupied box is l.

C2. If a free box is adjacent to a nonfree box, or to a free box with nonfree boxes as neighbors, its size is l.

C3. The decomposition is *balanced*, i.e., a free box has only adjacent free boxes whose sizes are either twice as large or half as small.

Figure 2.1 shows an example of a box decomposition in two dimensions. The shaded triangle is an obstacle, and the shaded disk represents the initial location. Figure 2.1(a) gives a box decomposition only satisfying C1. In Figure 2.1(b), boxes are further decomposed to satisfy C2 (e.g., boxes Ξ_1 and Ξ_2) and C3 (e.g., boxes Ξ_3 and Ξ_4).

We use a tree structure to represent the box decomposition, where an internal node represents a decomposed box and a leaf node a nondecomposed one. Each internal node has 2^d children. We can perform the box decomposition in two stages. In the first stage, decompose until C1 is satisfied. Let M be the number of boxes obtained after this stage. It is easy to see that this stage can be computed in time $O(nM)$, where n is the number of obstacle constraints. In the second stage, we further decompose certain boxes until C2 and C3 are satisfied. Basically, we check all the leaf nodes in a bottom-up manner, working from small boxes to large ones. For each leaf node, find its neighbors and further decompose them if their sizes are too large. Let N be the size of the final box decomposition. We claim that the time spent in this stage is $O(Nlog(L/l))$. Since the size of the smallest box is l, the box-decomposition tree has a depth of $O(\log(L/l))$. Thus finding the neighbors of a box takes at most $O(\log(L/l))$ time. Since we find neighbors for at most N boxes, the total running time is $O(N \log(L/l))$.

Notice that in the worst case, $N = O((L/l)^d)$, but N tends to be much smaller for cases where the obstacles are sparse.

Let $\mathcal{B}$ be the final box decomposition of W. Later, when we refer to a box (resp., a face) of $\mathcal{B}$, we mean a nondecomposed box (resp., face) of $\mathcal{B}$.

2.3.2. Discretization. Let $\mathcal{B}$ be the box decomposition of W (with respect to Ω) satisfying C1–C3. For each face Δ of $\mathcal{B}$ such that Δ is not further decomposed (recall that Δ is a $(d-1)$-dimensional box), we select $(2/\epsilon)^{d-1}$ uniformly spaced points on Δ, with spacing $\epsilon s/2$, where s is the size of Δ. The set of points obtained in this way are called the *CS grid points* of $\mathcal{B}$. Notice that the number of grid points on each face is the same, while the spacing of grid points grows linearly with the size of the faces. Thus we obtain a nonuniform discretization of the configuration space.

Let

$$\mathcal{V} = \{v = (j_1\epsilon, \ldots, j_d\epsilon) \mid j_1, \ldots, j_d \in \mathbb{Z} \text{ and } \|v\| \leq 1\}. \tag{2.1}$$

$\mathcal{V}$ is called the set of *grid velocities*. A state a is called a *CS grid state* of $\mathcal{B}$ if $\textsc{loc}(a)$ is a *CS* grid point of $\mathcal{B}$ and $\textsc{vec}(a) \in \mathcal{V}$. The grid states of $\mathcal{B}$ (including the initial and the final states) are the nodes of the graph to be constructed.

For a state a that lies on a face Δ, let $s(a)$ be the function returning the size of the face Δ. Define $ngb(a) = ngb(a, \epsilon s(a)/4, \epsilon/2)$. By the construction of the grid states, it is easy to show the following result.

LEMMA 2.3. *If a is a state such that $\textsc{loc}(a)$ lies on a face Δ of $\mathcal{B}$, there exists a CS grid state a' such that $a \in ngb(a')$.*

2.4. Precomputing canonical trajectories. In this section, we derive a set of *canonical trajectories*, which once computed can be used repeatedly for different problem instances with different obstacle distributions. These canonical trajectories are from grid states to grid states which lie on the faces of an *extended box*. The extended boxes are to guarantee that each canonical trajectory has a length at least proportional to the size of the box it lies in (see section 2.4.1). Section 2.4.2 describes the canonical trajectories and how to compute them.

2.4.1. Extended boxes. Let $\mathcal{B}$ be the box decomposition of W. Given a state a, define $\xi(a)$ to be the *extended box* of a, which is the smallest region composed of boxes such that (i) the boundary of the union of these boxes forms a closed $(d-1)$-dimensional surface and (ii) for any point q that lies on the closed surface, $\|\textsc{loc}(a) - q\|_\infty \geq s(a)/2$.

Let Δ be a face of $\mathcal{B}$ of size s and let Ξ be the box that contains Δ (pick one arbitrarily if both of the two boxes containing Δ have size s). Decompose Δ into 2^{d-1} $(d-1)$-dimensional boxes, each of size $s/2$, and label them $\Delta_1, \ldots, \Delta_{2^{d-1}}$. Given any two states a and b, if both $\textsc{loc}(a)$ and $\textsc{loc}(b)$ lie in the interior of Δ_j, then $\xi(a)$ and $\xi(b)$ are the same. This implies that we can extend the definition of extended boxes for faces. Define $\xi(\Delta_j)$ to be the extended boxes of Δ_j, which are the same as the extended boxes of $\xi(a)$, for any state a such that $\textsc{loc}(a)$ lies in the interior of Δ_j. $\xi(\Delta_j)$ is called an extended box of *size* $s/2$, since Δ_j is a face of size $s/2$.

An extended box (resp., a box) is said to have a *clearance* of μ if for any point p that lies in the extended box (resp., the box), p has a clearance of μ. Notice that a free box has a clearance of s if all its neighbors are free boxes and have a size of at least s. The following two lemmas describe the clearance properties of extended boxes of different sizes.

LEMMA 2.4. *An extended box of size $s \geq 2l$ has a clearance of at least $s/2$; an extended box of size l has a clearance of at least l.*

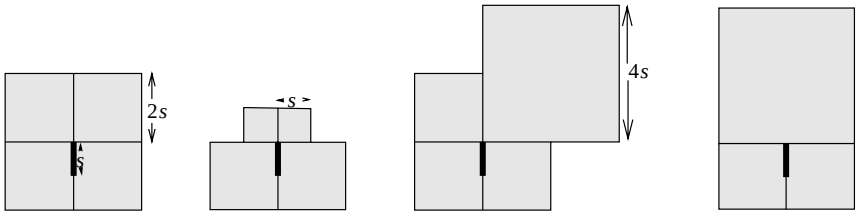


FIG. 2.2. *Some canonical extended boxes of* $\tilde{\Delta}(s)$*, where* $\tilde{\Delta}(s)$ *is a line segment drawn in thick lines.*

Proof. Suppose that the extended box is an extended box of face Δ_j whose size is s. Let Ξ be the box containing Δ_j and the size of Ξ be $2s$. Each other box in the extended box is a neighbor of Ξ. Thus their sizes are at least s, by the balanced property of the box decomposition.

If $s \geq 2l$, for any box Ξ' in the extended box, its neighbors must have sizes $\geq s/2$. By property C2 of the box decomposition, these neighbor boxes must be free. Otherwise, the size of Ξ' cannot be larger than l. Thus Ξ' has a clearance of $\geq s/2$. This implies that the extended box has a clearance of $\geq s/2$.

If $s = l$, then Ξ is a box of size $2l$. Since the box decomposition satisfies C2, we know that all Ξ's neighbors and all their neighbors are free boxes whose sizes are at least l. This implies that the extended box has a clearance of l. This completes the proof. □

LEMMA 2.5. *Let* $\xi(\Delta_j)$ *be an extended box of a face* Δ_j *of size* $l/2$*. If there exists a point* p *on* Δ_j *such that* p *has a clearance of* $3l$*, then the clearance of the extended box is at least* l*.*

Proof. It suffices to prove that for each point q on the boundary of $\xi(\Delta_j)$, q has a clearance of l. If q lies on the face of a box of size l, then $\|q - p\|_\infty \leq 2l$. Since the clearance of p is $3l$, the clearance of q is at least $3l - 2l = l$. If q lies on the face of a box of size $\geq 2l$, its clearance is at least l, since the clearance of this box is at least l. □

Let $\tilde{\Delta}(s/2) = \{p = (p_1, \ldots, p_d) \,|\, p_1 = -s/2 \text{ and } 0 \leq p_2, \ldots, p_d \leq s/2\}$. $\tilde{\Delta}(s/2)$ is called the *canonical face* of size $s/2$. We can transform Δ_j, Ξ, and $\xi(\Delta_j)$ by translation, rotation, and reflection such that Ξ becomes $\tilde{\Xi}(s)$, the canonical box of size s, and Δ_j becomes $\tilde{\Delta}(s/2)$. The transformed extended box $\xi(\Delta_j)$ is called a *canonical extended box* of $\tilde{\Delta}(s/2)$, or a canonical extended box of size $s/2$. Since the box decomposition $\mathcal{B}$ is balanced, the number of canonical extended boxes of a fixed size is a function depending only on the number of dimensions d, but not on the size or the location. Let $\xi(s)$ be the set of canonical extended boxes of $\tilde{\Delta}(s)$. We can give an order to $\xi(s)$ and let $\xi(s, j)$ be the jth canonical extended box of $\tilde{\Delta}(s)$ for some valid j. Figure 2.2 gives some examples of canonical extended boxes in two dimensions.

Let $\mathcal{P}(s)$ be the set of $(1/\epsilon)^{d-1}$ uniformly spaced points on $\tilde{\Delta}(s)$ with spacing ϵs. Define $\mathcal{Q}(s, j)$ as follows. If Δ is a face of size s' belonging to the boundary of $\xi(s, j)$, include the set of $(2/\epsilon)^{d-1}$ uniformly spaced points with spacing $\epsilon s'/2$ on Δ in $\mathcal{Q}(s, j)$. Let $\mathcal{I}(s) = \mathcal{P}(s) \times \mathcal{V}$ and $\mathcal{F}(s, j) = \mathcal{Q}(s, j) \times \mathcal{V}$. A state in the set $\mathcal{I}(s)$ is called a *canonical starting state* and a state in the set $\mathcal{F}(s, j)$ is called a *canonical ending state* of $\xi(s, j)$. It is easy to see that $|\mathcal{I}(s)| = O((1/\epsilon)^{2d-1})$. It also holds that $|\mathcal{F}(s, j)| = O(d(1/\epsilon)^{2d-1})$. This is because of the balanced property of the box decomposition. The boundary of $\xi(s, j)$ can contain at most $O(d)$ faces, each contributing $O((1/\epsilon)^{2d-1})$ states.

2.4.2. Computing connection tables. Let $\xi(s,j)$ be the jth canonical extended box of $\tilde{\Delta}(s)$ for some valid s and j. Let $CT(s,j)$ be the *connection table* of $\xi(s,j)$. The connection table contains precomputed trajectories which connect canonical starting states and canonical ending states. In this section we describe how to compute the connection tables.

First we introduce our correcting lemma (whose proof can be found in section 2.6), which states a result essential to the proofs of other lemmas.

LEMMA 2.6 (correcting lemma). *Fix a constant $c \leq 1$ and let $\rho_c = \bar{v}^2/(c\bar{a})$, where $\bar{a}, \bar{v} > 0$ are arbitrary. Let Γ be an $(\bar{a},\bar{v})$-trajectory from i to f. Given any $\rho > 0$ and $\varepsilon > 0$ and given any two states g and h, if $\rho > \rho_c$, $\|\text{LOC}(f) - \text{LOC}(i)\|_\infty \geq \rho$, $g \in ngb(i, \varepsilon\rho/2, \varepsilon\bar{v}/2)$, and $h \in ngb(f, \varepsilon\rho/2, \varepsilon\bar{v}/2)$, we can construct a trajectory Γ' from g to h by correcting Γ such that Γ' satisfies the following properties:*

P1. $T(\Gamma') = T(\Gamma)$.

P2. *Γ' is a $((1+4c\sqrt{d}\varepsilon)^2\bar{a}, (1+4c\sqrt{d}\varepsilon)\bar{v})$-trajectory.*

P3. $d(\Gamma', \Gamma) \leq (17/16)\varepsilon\rho$.

Here onwards, we fix

$$c = \max(2/l, 1), \tag{2.2}$$

where l is the size of the smallest box. In our later application of this lemma, it always holds that $\bar{v}^2/\bar{a} = 1$, thus $\rho_c = 1/c$. This implies that

$$l/2 \geq \rho_c. \tag{2.3}$$

Also let

$$e = 4c\sqrt{d}. \tag{2.4}$$

Consider a pair of states (i,f) that lie on the boundary of an extended box ζ. A $(1,1)$-trajectory from i to f is called *legal* if its path lies inside ζ. The pair (i,f) is *legal* if there exists at least a legal trajectory from i to f. We will see later that these legal trajectories are the potential trajectories that we need to approximate. We call a pair of grid states (i,f) *good* if there exists at least a legal pair (g,h) such that $g \in ngb(i)$ and $h \in ngb(f)$. Recall that $ngb(i) = ngb(i, \epsilon s(i)/4, \epsilon/2)$, where $s(i)$ gives the size of the face where the location of state i lies. Roughly speaking, the next lemma (existence lemma) states that if (i,f) is a good pair, then there *exists* a trajectory Γ from i to f such that Γ approximates *any* legal trajectories for any legal pair (g,h), where $g \in ngb(i)$ and $h \in ngb(f)$. Thus for our purpose of approximation, it is enough to compute only the trajectories for good pairs. For a pair of legal states (a,b), let $\hat{T}(a,b)$ be the time length of the legal trajectory from a to b whose time length is the smallest among all legal trajectories from a to b.

LEMMA 2.7 (existence lemma). *Let $\xi(s,j)$ be the jth canonical extended box of $\tilde{\Delta}(s)$ for some valid s and j. For any $i \in \mathcal{I}(s)$ and $f \in \mathcal{F}(s,j)$, if (i,f) is good, then there exists a trajectory Γ from i to f such that the trajectory satisfies the following properties:*

P1. *For any $i' \in ngb(i)$ and $f' \in ngb(f)$, if (i',f') is legal, $T(\Gamma) \leq \hat{T}(i',f')$.*

P2. *Γ is a $((1+e\epsilon)^2, (1+e\epsilon))$-trajectory.*

P3. $d(\Gamma, \xi(s,j)) \leq (17/8)\epsilon s$.

Proof. Let $g \in ngb(i)$ and $h \in ngb(f)$ be such that (g,h) is legal and that $\hat{T}(g,h)$ is the smallest among those of all such legal pairs;let Γ' be the $(1,1)$-trajectory from g

to h whose time length is $\hat{T}(g,h)$. We will show the existence of a trajectory satisfying P1–P3 by constructing one from Γ'. There are three cases depending on whether $s(f)$ (the size of the face where f lies) is s, $2s$, or $4s$. We will prove the case when $s(f) = 4s$; this case gives the worst bounds. The other two cases can be handled in a similar way.

Let Γ be the trajectory obtained by correcting Γ'. Let $\bar{a} = 1$, $\bar{v} = 1$, and $\rho = 2s$ in the correcting lemma (Lemma 2.6). Since $s \geq l/2$, $\rho = 2s \geq l > \rho_c$. We can show that $\|\textsc{loc}(h) - \textsc{loc}(g)\|_\infty \geq 2s = \rho$. Since $g \in ngb(i)$, $g \in ngb(i, \epsilon\, s(i)/4, \epsilon/2)$. Since $\rho = 2s = s(i)$, it is also true that $g \in ngb(i, \epsilon\rho/2, \epsilon/2)$. This implies that $i \in ngb(g, \epsilon\rho/2, \epsilon/2)$. Similarly, $f \in ngb(h, \epsilon\rho/2, \epsilon/2)$. Thus the conditions of the correcting lemma are satisfied. This implies that $T(\Gamma) \leq \hat{T}(g,h)$, satisfying P1. Γ is a $((1+e\epsilon)^2, (1+e\epsilon))$-trajectory, and $d(\Gamma, \Gamma') \leq (17/16)\epsilon\rho = (17/8)\epsilon s$. Since $d(\Gamma', \xi(s,j)) = 0$, $d(\Gamma, \xi(s,j)) \leq (17/8)\epsilon s$, proving P3. □

For a good pair (i,f) (with respect to $\xi(s,j)$), our goal is to approximate (since we do not know how to compute exactly) such a trajectory Γ as stated in the existing lemma (Lemma 2.7). The approximation is done in two steps. First, we apply the TC-graph method to compute a trajectory Γ' which is close to Γ timewise and spatially. However, the TC-graph method does not guarantee that Γ' is from i to f. Instead, we only know that Γ' is from some i' close to i to some f' close to f. Second, we correct Γ' to obtain a trajectory Γ'' which is really from i to f. By the correcting lemma, Γ'' approximates Γ', and thus approximates Γ. In the first step, by setting $\rho = l/2$ and $\varepsilon = \epsilon$ in Corollary 2.2, we are able to guarantee that

- Γ' is a $((1+e\epsilon)^2, (1+e\epsilon))$-trajectory. (Notice that even though Corollary 2.2 is about $(1,1)$-trajectories, the results apply to $((1+e\epsilon)^2, (1+e\epsilon))$-trajectories.)
- $T(\Gamma') \leq (1+\epsilon)T(\Gamma)$.
- Γ' is from some $i' \in ngb(i, \epsilon l/4, \epsilon/2)$ to some $f' \in ngb(f, \epsilon l/4, \epsilon/2)$.
- $d(\Gamma', \xi(s,j)) \leq (17/8)\epsilon s + \epsilon l/2$. The existence of a trajectory satisfying this condition in the TC graph is due to the fact that $d(\Gamma, \xi(s,j)) \leq (17/8)\epsilon s$ and to the existence of a trajectory $\tilde{\Gamma}$ in the graph such that $d(\tilde{\Gamma}, \Gamma) \leq \epsilon l/2$.

In the second step, setting $\bar{a} = (1+e\epsilon)^2$, $\bar{v} = 1 + e\epsilon$, and $\rho = l/2 \geq \rho_c$, we can see that the conditions of the correcting lemma are satisfied. Thus Γ'' is a $((1+e\epsilon)^4, (1+e\epsilon)^2)$-trajectory, $T(\Gamma'') = T(\Gamma') \leq (1+\epsilon)T(\Gamma)$, and $d(\Gamma'', \Gamma') \leq (17/32)\epsilon l$. Since $d(\Gamma', \xi(s,j)) \leq (17/8)\epsilon s + \epsilon l/2$, $d(\Gamma'', \xi(s,j)) \leq (17/8)\epsilon s + (33/32)\epsilon l \leq 5\epsilon s$ (using the fact that $s \geq l/2$). In summary, we obtain the following result.

LEMMA 2.8 (loose-tracking lemma). *Let $\xi(s,j)$ be the jth canonical extended box of $\tilde{\Delta}(s)$ for some valid s and j. Let $i \in \mathcal{I}(s)$ and $f \in \mathcal{F}(s,j)$. If (i,f) is good, then Γ'' computed as above satisfies the following properties:*

P1. *For any $g \in ngb(i)$ and $h \in ngb(f)$, if (g,h) is legal, $T(\Gamma'') \leq (1+\epsilon)\,\hat{T}(g,h)$.*

P2. *Γ'' is a $((1+e\epsilon)^4, (1+e\epsilon)^2)$-trajectory.*

P3. *$d(\Gamma'', \xi(s,j)) \leq 5\epsilon s$.*

Fix a canonical extended box $\xi(s,j)$ for some valid s and j. For each $i \in \mathcal{I}(s)$ and $f \in \mathcal{F}(s,j)$, we precompute a trajectory from i to f as described above. We store the trajectory (i.e., the initial state i, the final state f, the acceleration function, and the time length) in the connection table $CT(s,j)$. A connection table computed in this way satisfies the following property.

LEMMA 2.9 (connection-table property). *Let $\xi(s,j)$ be the jth canonical extended box of $\tilde{\Delta}(s)$ for some valid s and j and let $CT(s,j)$ be the connection table for $\xi(s,j)$. For any canonical starting state $i \in \mathcal{I}(s)$ and any canonical ending state $f \in \mathcal{F}(s,j)$, if (i,f) is good, then $CT(s,j)$ contains a trajectory Γ from i to f and Γ satisfies the*

following properties:

P1. *For any* $g \in ngb(i)$ *and* $h \in ngb(f)$*, if* (g,h) *is good,* $T(\Gamma) \leq (1+\epsilon)\,\hat{T}(g,h)$.

P2. Γ *is a* $((1+e\epsilon)^4, (1+e\epsilon)^2)$*-trajectory.*

P3. $d(\Gamma, \xi(s,j)) \leq 5\epsilon s$.

Let $\xi(l/2,0)$ be the canonical extended box that consists of four boxes of size l. Apart from $CT(l/2,0)$, we maintain a special connection table CT_0 for this canonical extended box. For each $i \in \mathcal{I}(l/2)$, we construct a TC graph G rooted at i (for a description of the TC-graph method, see the appendix). We add a trajectory Γ from i to f in CT_0, if (i) f is a node of G, (ii) f either lies inside $\xi(l/2,0)$ or its L_∞ distance to the extended box is no more than $\epsilon l/2$, (iii) there is a path on G from i to f (whose corresponding trajectory is Γ). By the property of the TC-graph method, we can show the following.

LEMMA 2.10. *Let* $i \in \mathcal{I}(l/2)$*. For any pair of states* g *and* h *such that* $g \in ngb(i, \epsilon l/2, \epsilon/2)$*,* h *lies inside* $\xi(l/2,0)$*, and the optimal* $(1,1)$*-trajectory from* g *to* h *lies inside* $\xi(l/2,0)$*, there exists a* $(1,1)$*-trajectory in* CT_0 *from* i *to some* f *such that* $f \in ngb(h, \epsilon l/2, \epsilon/2)$ *and the time length of this trajectory is no more than* $(1+\epsilon)$ *times the time length of the optimal* $(1,1)$*-trajectory from* g *to* h*.*

To correct a trajectory takes only $O(1)$ time, since it basically involves computing and adding $O(1)$ number of corrective acceleration terms (see section 2.6). So the time complexity of computing a connection table $CT(s,j)$ is determined by how fast we can compute a trajectory Γ' for each pair of canonical starting and ending states. We can compute the trajectories using the TC-graph method in the following manner. For each canonical starting state i, compute the TC graph rooted at i. Expand the graph until all the canonical ending states of $\xi(s,j)$ are reached, or no new nodes can be added. A shortest graph path from i to a canonical ending state gives a trajectory between these two states. All these paths can be stored in a concise way by storing at each node its preceding node along the paths. Since the extended box is of size s, the graph has $O(s^d(1/\epsilon)^{6d-1})$ edges. (For a brief analysis on the number of TC-graph edges, see the end of the appendix.) This also bounds the time and space for computing the trajectories for a single canonical starting state. Since there are $O((1/\epsilon)^{2d-1})$ canonical starting states for an extended box, it takes $O(s^d(1/\epsilon)^{8d-2})$ time and space to compute a connection table $CT(s,j)$. A similar analysis shows that CT_0 can also be computed in $O(l^d(1/\epsilon)^{8d-2})$ time and space.

2.5. Approximation in the presence of obstacles.

2.5.1. The algorithm. In this section we present our approximation algorithm for computing collision-free near-optimal trajectories in the presence of obstacles. Let W be a d-dimensional configuration space of size L with a set Ω of obstacles. Let i and f be the initial and the final states, respectively. And let $\mathcal{B}$ be the box decomposition of W satisfying C1–C3 (recall that we presented C1–C3 and how to obtain such a box decomposition in section 2.3.1). Our approximation algorithm constructs a weighted directed graph $G = (V,E)$, where V includes i, f, and a subset of CS grid states induced by $\mathcal{B}$. A CS grid state a is included in V if (i) $s(a) \geq 2l$ or (ii) $s(a) = l$ and the clearance of $\xi(a)$ is at least l.

Let a and b be two states that lie within an extended box ζ. To see if there is a precomputed trajectory from a to b, we transform ζ to its canonical form, and also a and b with it. If $\mathcal{T}$ is the transformation, we say that there is a precomputed trajectory from a to b if there is a trajectory from $\mathcal{T} \circ a$ to $\mathcal{T} \circ b$ in the connection table of $\mathcal{T} \circ \zeta$. If Γ is the trajectory from $\mathcal{T} \circ a$ to $\mathcal{T} \circ b$, then $\mathcal{T}^{-1} \circ \Gamma$ is the trajectory

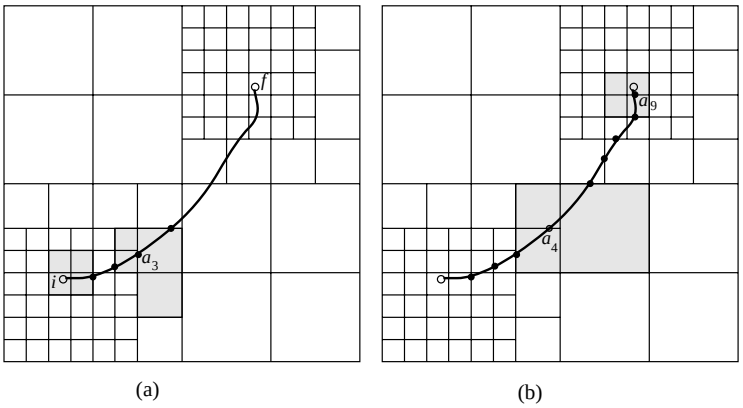


FIG. 2.3. (a) $\xi(a_0)$ *and* $\xi(a_3)$; (b) $\xi(a_4)$ *and* $\xi(a_9)$.

from a to b. It is easy to see that $\mathcal{T}^{-1} \circ \Gamma$ satisfies the connection-table properties P1–P3 (see Lemma 2.9) with respect to ζ.

We construct the edge set in the following way. For node i, add an edge from i to another node a if (i) i lies within $\xi(a)$ and (ii) there is a precomputed trajectory from[1] a^- to i^-. Similarly, for node f, add an edge from a node a to f if (i) f lies within $\xi(a)$ and (ii) there is a precomputed trajectory from a to f. For any other node a, add an edge from a to another node b if (i) b lies on the boundary of $\xi(a)$ and (ii) there is a precomputed trajectory from a to b. The weight of each edge is equal to the time length of its corresponding trajectory.

THEOREM 2.11 (safe loose-tracking theorem). *If there exists an optimal* $3l$*-safe* $(1,1)$*-trajectory* Γ *from state* i *to state* f*, then the graph* G*, constructed as above, contains a path whose corresponding trajectory* Γ' *satisfies the following properties:*

P1. $T(\Gamma') \leq (1+\epsilon)\,T(\Gamma)$.

P2. Γ' *is a* $((1+e\epsilon)^4, (1+e\epsilon)^2)$*-trajectory.*

P3. Γ' *does not intersect the interior of* Ω.

P4. Γ' *is from some* $i' \in ngb(i, \epsilon l/2, \epsilon/2)$ *to some* $f' \in ngb(f, \epsilon l/2, \epsilon/2)$.

Proof. Let Γ be divided into segments of trajectories $\Gamma_{a_0a_1} \| \Gamma_{a_1a_2} \| \dots \| \Gamma_{a_{m-1}a_m}$ such that $a_0 = i$, $a_m = f$, and each a_j is the state where Γ first exits $\xi(a_{j-1})$ for $1 \leq j \leq m-1$. Figure 2.3 gives an example in 2D. The optimal $(1,1)$-trajectory from i to f (drawn in thick curve) is divided into 10 segments (each dark circle represents an a_j for $1 \leq j < 10$). Some of the extended boxes are shown as shaded regions. We consider the following parts $\Gamma_{a_0a_1}$, $\Gamma_{a_1a_2} \| \dots \| \Gamma_{a_{m-2}a_{m-1}}$, and $\Gamma_{a_{m-1}a_m}$ separately.

Since each a_j, for $1 \leq j \leq m-1$, is on some face of $\mathcal{B}$, by Lemma 2.3, we can find a CS grid state b_j such that $a_j \in ngb(b_j)$. If $s(a_j) \geq 2l$, then $s(b_j) \geq 2l$ and b_j is in the node set V. Otherwise, since a_j is $3l$-safe, $\xi(a_j)$ has a clearance of at least l. Since $\xi(b_j) = \xi(a_j)$, $\xi(b_j)$ also has a clearance of at least l and b_j is in the node set V. We will show that the graph path $(b_1, b_2) \| \dots \| (b_{m-2}, b_{m-1})$ corresponds to a trajectory satisfying P1–P3. To this end, we show that there is an edge from b_{j-1} to b_j whose corresponding trajectory $\Gamma_{b_{j-1}b_j}$ satisfies P2, P3, and $T(\Gamma_{b_{j-1}b_j}) \leq (1+\epsilon)\,T(\Gamma_{a_{j-1}a_j})$ for $1 < j < m$.

By the way Γ is divided, $d(\Gamma_{a_{j-1}a_j}, \xi(a_{j-1})) = 0$. Thus each pair (a_{j-1}, a_j) is legal. This implies that each (b_{j-1}, b_j) is good. By the connection table property

[1] For a state a, we use a^- to denote a state of $(\text{LOC}(a), -\text{VEC}(a))$.

(Lemma 2.9), there is a precomputed trajectory $\Gamma_{b_{j-1}b_j}$ from b_{j-1} to b_j. Thus an edge from b_{j-1} to b_j is added in the construction of G. Also by the connection table property, $\Gamma_{b_{j-1}b_j}$ is a $((1+e\epsilon)^4,(1+e\epsilon)^2)$-trajectory, and its time length is no more than $(1+\epsilon)T(\Gamma_{a_{j-1}a_j})$. Next we need to show that $\Gamma_{b_{j-1}b_j}$ is collision free.

Let s be the size of $\xi(b_{j-1})$. We consider the three cases $s \geq 2l$, $s = l$, and $s = l/2$ separately. If $s \geq 2l$ (resp., $s = l$), the extended box $\xi(b_{j-1})$ has a clearance of $s/2$ (resp., l). On the other hand, $d(\Gamma_{b_{j-1}b_j}, \xi(b_{j-1})) \leq 5\epsilon s$. Thus if $\epsilon \leq 1/10$ is chosen small enough, $\Gamma_{b_{j-1}b_j}$ is collision free. When $s = l/2$, $\xi(b_{j-1})$ has a clearance of at least l. This is because a_{j-1} is $3l$-safe. By Lemma 2.4, $\xi(a_{j-1})$ has a clearance of at least l. This implies that $\xi(b_{j-1})$ has a clearance of at least l, since $\xi(b_{j-1}) = \xi(a_{j-1})$. Since $d(\Gamma_{b_{j-1}b_j}, \xi(b_{j-1})) \leq (5/2)\epsilon l$, $\Gamma_{b_{j-1}b_j}$ is collision free if ϵ is chosen small enough.

Now consider the part $\Gamma_{a_{m-1}a_m}$, which lies inside $\xi(a_{m-1})$. When constructing $\mathcal{B}$, $\textsc{loc}(f)$ is considered as an obstacle vertex. Thus the box containing $\textsc{loc}(f)$ and its neighbors all have size l. This means that $\xi(a_{m-1})$ is an extended box consisting of four boxes of size l. By Lemma 2.10, there exists in CT_0 a precomputed trajectory $\Gamma_{b_{m-1}f}$ from b_{m-1} to some $f' \in ngb(f, \epsilon l/2, \epsilon/2)$. Thus an edge from b_{m-1} to f is added in the construction of G. Also $\Gamma_{b_{m-1}f}$ is a $(1,1)$-trajectory whose time length is no more than $(1+\epsilon)\Gamma_{a_{m-1}a_m}$. To show that $\Gamma_{b_{m-1}f}$ is collision free, notice that the clearance of $\xi(a_{m-1})$ is at least l, since a_{m-1} is $3l$-safe. But $d(\Gamma_{b_{m-1}f}, \xi(b_{m-1})) \leq \epsilon l/2$, so this trajectory is also collision free. We can show similar results for $\Gamma_{a_0a_1}$. This completes the proof of this theorem. □

Thus the approximation problem is transformed to one of searching for a shortest path on the graph G. Notice that in order for this algorithm to be correct, each edge introduced in G must correspond to a collision-free trajectory. This is guaranteed by the way we choose the node set V and the connection table properties (a proof similar to the one showing that each $\Gamma_{b_{j-1}b_j}$ is collision free, used in the above theorem). We do not have to explicitly check whether each trajectory segment is collision free. The obtained shortest path corresponds to a $((1+e\epsilon)^4,(1+e\epsilon)^2)$-trajectory. Applying the time-rescaling lemma (Lemma 2.1) with a scaling factor of $(1+e\epsilon)^2$, we can obtain the following.

COROLLARY 2.12. *Let W be a d-dimensional configuration space of size L and Ω a set of obstacles. Let Γ be an optimal $3l$-safe $(1,1)$-trajectory from i to f. We can compute a $(1,1)$-trajectory Γ' from some $i' \in ngb(i, \epsilon l/2, 3e\epsilon)$ to some $f' \in ngb(f, \epsilon l/2, 3e\epsilon)$ such that $T(\Gamma')$ is at most $(1+3e\epsilon)$ times $T(\Gamma)$.*

2.5.2. The time complexity. The running time of the algorithm consists of the following components:

1. Time to generate the graph nodes (i.e., the grid states). If N is the total number of boxes in the final decomposition, the time to perform the decomposition is $O(nN + N\log N)$, where n is the number of constraints defining the configuration obstacles. Since each box contributes at most $O((1/\epsilon)^{2d-1})$ grid states, the time to generate the grid states, after a decomposition, is $O(N(1/\epsilon)^{2d-1})$.
2. Time to compute the graph edges. Since each node is connected to at most $O((1/\epsilon)^{2d-1})$ other nodes, the total number of edges is $O(N(1/\epsilon)^{4d-2})$. This bounds the time to compute the edges, since it takes $O(1)$ time to compute an edge.
3. Time to search for a shortest graph path. Using Dijkstra's algorithm with the priority queue implemented with a binary heap, this time is $O((|V| + |E|)\log|V|)$, where $|V|$ and $|E|$ are the number of nodes and edges, respec-

tively. Plugging in our numbers, the searching time is roughly $O(dN \log N (1/\epsilon)^{4d-2})$.

4. Time to rescale the obtained trajectory to a $(1,1)$-trajectory, which is $O(1)$.

Combining Corollary 2.12 and the time-complexity analysis, and choosing ϵ sufficiently small, we obtain the following result.

THEOREM 2.13. *Fix an $l > 0$ and an $\varepsilon > 0$. After some precomputation, we can achieve the following.*

Let W be a d-dimensional configuration space of size L. Let Ω be a set of configuration obstacles defined by a total of n algebraic equations, each of $O(1)$ degrees. Given any two states i and f, we can compute a collision-free $(1,1)$-trajectory from i' to f' such that the time length of the trajectory is at most $(1+\varepsilon)$ times the time length of an optimal $3l$-safe $(1,1)$-trajectory from i to f. Furthermore, $\|\textsc{loc}(i') - \textsc{loc}(i)\|_\infty \le \varepsilon l$, $\|\textsc{vec}(i') - \textsc{vec}(i)\|_\infty \le \varepsilon$, $\|\textsc{loc}(f') - \textsc{loc}(f)\|_\infty \le \varepsilon l$, and $\|\textsc{vec}(f') - \textsc{vec}(f)\|_\infty \le \varepsilon$.

The running time of our algorithm is $O(nN + N \log N (1/\varepsilon)^{4d-2})$, where $N = O((L/l)^d)$.

2.6. Correcting a trajectory. In this subsection we prove the correcting lemma (Lemma 2.6). The correcting lemma roughly states the following. Let Γ be an $(\bar{a}, \bar{v})$-trajectory from a state i to a state f. Given another pair of states g and h, we can construct a trajectory Γ' from g to h by correcting Γ. Furthermore, we show that if g and h are close to i and f, respectively, the correction is small. Note that in this subsection, the results are presented in the absence of obstacles.

For simplicity of illustration and analysis, we first look at the 1D case. Let $\delta a(t)$ be the corrective acceleration, i.e., $a_{\Gamma'}(t) = a_\Gamma(t) + \delta a(t)$ for $0 \le t \le T(\Gamma)$. Let $\delta v(t) = v_{\Gamma'}(t) - v_\Gamma(t)$ and $\delta p(t) = p_{\Gamma'}(t) - p_\Gamma(t)$. Also let $\Delta v_i = v_{\Gamma'}(0) - v_\Gamma(0) = \textsc{vec}(g) - \textsc{vec}(i)$, $\Delta v_f = v_{\Gamma'}(T) - v_\Gamma(T) = \textsc{vec}(h) - \textsc{vec}(f)$, $\Delta p_i = p_{\Gamma'}(0) - p_\Gamma(0) = \textsc{loc}(g) - \textsc{loc}(i)$, and $\Delta p_f = p_{\Gamma'}(T) - p_\Gamma(T) = \textsc{loc}(h) - \textsc{loc}(f)$. Thus,

$$\begin{aligned}\delta v(t) &= v_{\Gamma'}(0) + \int_0^t (a_\Gamma(\mu) + \delta a(\mu))\, d\mu - \left(v_\Gamma(0) + \int_0^t a_\Gamma(\mu)\, d\mu\right)\\ &= \Delta v_i + \int_0^t \delta a(\mu)\, d\mu\end{aligned}$$

and

$$\begin{aligned}\delta p(t) &= p_{\Gamma'}(0) + \int_0^t (v_\Gamma(\mu) + \delta v(\mu))\, d\mu - \left(p_\Gamma(0) + \int_0^t v_\Gamma(\mu)\, d\mu\right)\\ &= \Delta p_i + \int_0^t \delta v(\mu)\, d\mu\\ &= \Delta p_i + \int_0^t \left(\Delta v_i + \int_0^\mu \delta a(\nu)\, d\nu\right) d\mu\\ &= \Delta p_i + \Delta v_i t + \int_0^t \int_0^\mu \delta a(\nu)\, d\nu\, d\mu.\end{aligned}$$

Our object is to find $\delta a(t)$ such that

$$\Delta v_i + \int_0^T \delta a(\mu)\, d\mu = \Delta v_f \text{ and}$$

$$\Delta p_i + \Delta v_i T + \int_0^T \int_0^\mu \delta a(\nu)\, d\nu\, d\mu = \Delta p_f,$$

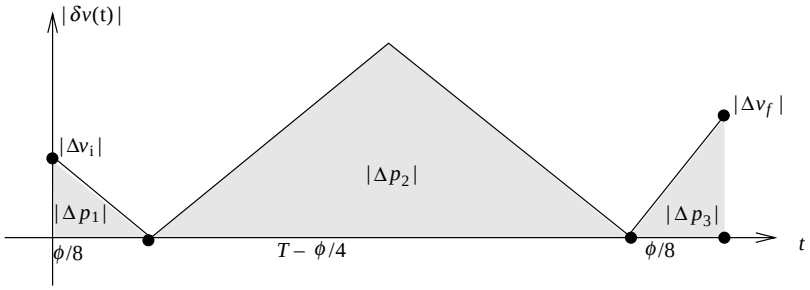


FIG. 2.4. *The correcting scheme.*

where $T = T(\Gamma)$. We also want to keep the absolute values of $\delta v(t)$, $\delta a(t)$, and $\delta p(t)$ small.

Fix a constant $c \geq 1$ and let

$$\rho_c = \bar{v}^2/(c\bar{a}).$$

Assume that

$$|\text{LOC}(f) - \text{LOC}(i)| \geq \rho \geq \rho_c$$

for some ρ, and let

$$\phi = \rho/\bar{v}.$$

Thus $T(\Gamma) \geq \phi$. Our correcting scheme is illustrated in Figure 2.4. Basically, we correct in three phases. In the first phase, we use a constant corrective acceleration to make $\delta v(t)$ become 0, while in the last phase, we use a constant corrective acceleration to make $\delta v(t)$ become Δv_f. The middle phase is used to correct the distance. Notice that at the beginning and the end of the second phase, $\delta v(t) = 0$.

The time length of the first and the last phases is $\phi/8$. Let Δa_1 be the constant corrective acceleration used in the first phase. Then

$$\Delta a_1 = -\frac{\Delta v_i}{\phi/8} = -\frac{8\Delta v_i}{\phi}.$$

If Δp_1 is the distance covered in this phase, then $\Delta p_1 = \Delta v_i \phi/16$. In this phase, $|\delta v(t)|$ is getting smaller and is upper bounded by $|\Delta v_i|$, while $|\delta p(t)|$ is upper bounded by $|\Delta p_i| + |\Delta p_1|$. Let Δa_3 be the constant corrective acceleration used in the last phase. Similarly, we have

$$\Delta a_3 = \frac{8\Delta v_f}{\phi}.$$

$\Delta p_3 = \Delta v_f \phi/16$ is the distance covered in this phase. In this phase, $|\delta v(t)|$ is upper bounded by $|\Delta v_f|$. At the beginning of this phase, $\delta p(t) = \Delta p_f - \Delta p_3$. Thus $|\delta p(t)|$ is upper bounded by $|\Delta p_f| + |\Delta p_3|$ during this phase.

Let $T' = T(\Gamma) - \phi/4$ be the time spent in the second phase. This phase is divided into two subphases of equal length. The corrective accelerations used in these two subphases have the same absolute value but opposite directions. The effect is that at the end of this phase, $\delta v(t)$ becomes 0 again. If Δp_2 is the distance covered in this

phase, $\Delta p_2 = \Delta p_f - \Delta p_i - \Delta p_1 - \Delta p_3$. Notice that $|\Delta p_2|$ and $\delta p(t)$ are both upper bounded by $|\Delta p_i| + |\Delta p_1| + |\Delta p_3| + |\Delta p_f|$. Let Δa_2 be the corrective acceleration used in the first subphase. Then

$$\Delta a_2 = \frac{4\Delta p_2}{(T')^2}.$$

In this phase, $|\delta v(t)|$ is upper bounded by $2|\Delta p_2|/T'$.

LEMMA 2.14. *For any $\varepsilon > 0$ and any two states g and h, if $g \in ngb(i, \varepsilon\rho/2, \varepsilon\bar{v}/2)$ and $h \in ngb(f, \varepsilon\rho/2, \varepsilon\bar{v}/2)$, where ρ is defined as above, then the trajectory Γ', constructed as above, satisfies the following properties:*

P1. $T(\Gamma') = T(\Gamma)$.

P2. Γ' *is a* $((1+4c\varepsilon)^2\bar{a}, (1+4c\varepsilon)\bar{v})$*-trajectory.*

P3. $d(\Gamma', \Gamma) \leq (17/16)\varepsilon\rho$.

Proof. P1 holds obviously. During the whole time period $T(\Gamma)$, the maximum difference in position is upper bounded by

$$\begin{aligned}
|\Delta p_i| + |\Delta p_1| + |\Delta p_3| + |\Delta p_f| &\leq \frac{1}{2}\varepsilon\rho + \frac{|\Delta v_i|\phi}{16} + \frac{|\Delta v_f|\phi}{16} + \frac{1}{2}\varepsilon\rho \\
&\leq \varepsilon\rho + \frac{(\varepsilon\bar{v})(\rho/\bar{v})}{16} \\
&\leq \left(1 + \frac{1}{16}\right)\varepsilon\rho \\
&\leq \frac{17}{16}\varepsilon\rho.
\end{aligned}$$

This implies that $|p_{\Gamma'}(t) - p_\Gamma(t)| \leq (17/16)\varepsilon\rho$ for $0 \leq t \leq T(\Gamma)$. By definition, $d(\Gamma', \Gamma) \leq (17/16)\varepsilon\rho$, proving P3. The maximum difference in velocity is upper bounded by

$$\begin{aligned}
\max\left(|\Delta v_i|, |\Delta v_f|, \frac{2|\Delta p_2|}{T'}\right) &\leq \max\left(\frac{\varepsilon\bar{v}}{2}, \frac{2(17/16)\varepsilon\rho}{(1-1/4)\phi}\right) \\
&\leq \max\left(\frac{\varepsilon\bar{v}}{2}, \frac{3\varepsilon\rho}{\phi}\right) \\
&\leq \max\left(\frac{\varepsilon\bar{v}}{2}, 3\varepsilon\bar{v}\right) \\
&\leq 4c\varepsilon\bar{v}.
\end{aligned}$$

The maximum difference in acceleration is upper bounded by

$$\begin{aligned}
\max\left(|\Delta a_1|, |\Delta a_2|, |\Delta a_3|\right) &\leq \max\left(\frac{8|\Delta v_i|}{\phi}, \frac{4|\Delta p_2|}{(T')^2}, \frac{8|\Delta v_f|}{\phi}\right) \\
&\leq \max\left(\frac{4\varepsilon\bar{v}}{\phi}, \frac{4(17/16)\varepsilon\rho}{((3/4)\phi)^2}\right) \\
&\leq \max\left(\frac{4\varepsilon\bar{v}}{\phi}, \frac{8\varepsilon\bar{v}}{\phi}\right) \\
&= \frac{8\varepsilon\bar{v}}{\phi} \\
&= \frac{8\varepsilon\bar{v}^2}{\rho}
\end{aligned}$$

$$\leq \frac{8\varepsilon\bar{v}^2}{\rho_c}$$
$$= 8c\varepsilon\bar{a}.$$

Thus the velocity of Γ' is upper bounded by $(1+4c\varepsilon)\bar{v}$ and the acceleration is bounded by $(1+4c\varepsilon)^2\bar{a}$, proving P2. □

In higher dimensions, we can add corrective accelerations for each dimension separately. Since the time length of the trajectory is not changed, the corrections can be carried out simultaneously without affecting each other. Let $v_\Gamma^j(t)$ be $v_\Gamma(t)$ projected in the jth dimension for $1 \leq j \leq d$. By Lemma 2.14, we have

$$|v_{\Gamma'}^j(t)| \leq 4c\varepsilon\bar{v} + |v_\Gamma^j(t)|.$$

By the triangle inequality,

$$\begin{aligned}\|v_{\Gamma'}(t)\| &\leq \|v_\Gamma(t)\| + \|v_{\Gamma'}(t) - v_\Gamma(t)\| \\ &\leq \bar{v} + \sqrt{\sum_{j=1}^{d}(v_{\Gamma'}^j(t) - v_\Gamma^j(t))^2} \\ &\leq \bar{v} + \sqrt{\sum_{j=1}^{d}(4c\varepsilon\bar{v})^2} \\ &\leq \bar{v} + 4c\sqrt{d}\varepsilon\bar{v} \\ &= (1 + 4c\sqrt{d}\varepsilon)\bar{v}.\end{aligned}$$

Similarly for the acceleration, we can obtain that

$$\|a_{\Gamma'}(t)\| \leq (1 + 4c\sqrt{d}\varepsilon)^2\bar{a}.$$

In summary, we have Lemma 2.6, reproduced here as Lemma 2.15.

LEMMA 2.15 (correcting lemma). *Fix a constant $c \geq 1$ and let $\rho_c = \bar{v}^2/(c\bar{a})$, where $\bar{a}, \bar{v} > 0$ are arbitrary. Let Γ be an $(\bar{a}, \bar{v})$-trajectory from a state i to a state f. Given any $\rho > 0$ and $\varepsilon > 0$ and given any two states g and h, if $\rho > \rho_c$, $\|\text{LOC}(f) - \text{LOC}(i)\|_\infty \geq \rho$, $g \in ngb(i, \varepsilon\rho/2, \varepsilon\bar{v}/2)$, and $h \in ngb(f, \varepsilon\rho/2, \varepsilon\bar{v}/2)$, then we can construct a trajectory Γ' from g to h by correcting Γ such that Γ' satisfies the following properties:*

P1. $T(\Gamma') = T(\Gamma)$.

P2. *Γ' is a $((1 + 4c\sqrt{d}\varepsilon)^2\bar{a}, (1 + 4c\sqrt{d}\varepsilon)\bar{v})$-trajectory.*

P3. $d(\Gamma', \Gamma) \leq (17/16)\varepsilon\rho$.

2.7. Scaling the dynamic bounds. So far, our presentation considers only the case when both the accelerations and the velocities are upper bounded by 1. This is sufficiently general because, as we mentioned in section 1.3, we can scale arbitrarily given bounds to 1 by scaling time and the configuration space. In this section, we will show how the scaling is done. Basically, for given arbitrary bounds, we scale the configuration space (also the locations and the velocities of the initial and final states) by some appropriate factors decided by the given bounds. We then apply the algorithm described in section 2.5 to compute a trajectory in the scaled configuration space with both dynamic bounds set to 1. Finally we "scale back" the computed trajectory. We will see that the scaled trajectory abides by the given dynamic bounds and is a close approximation to an optimal trajectory.

For a path Π, we use $L(\Pi)$ to denote the path length of Π. Let Π and Π' be two paths. We say that $\Pi' = \alpha\Pi$, or that Π' is Π scaled by a factor of α, if $L(\Pi') = \alpha\, L(\Pi)$ and $\Pi'(\ell) = \alpha\, \Pi(\ell/\alpha)$ for $0 \le \ell \le \alpha\, L(\Pi)$.

In the following, we first present the velocity-scaling lemma, which shows the scaling relation between the velocity and the time and the space. Next, the acceleration-scaling lemma shows how to scale the acceleration by scaling the time and the space. By combining these two lemmas, we obtain the acceleration-velocity scaling lemma. Note that unlike the time-rescaling lemma (Lemma 2.1), the acceleration and the velocity can be scaled by different and unrelated factors in our lemma.

LEMMA 2.16 (velocity-scaling lemma). *Let Γ be an $(\bar{a}, \bar{v})$-trajectory and Π be its path. The path $(1/\alpha^2)\Pi$ can be traversed by an $(\bar{a}, \bar{v}/\alpha)$-trajectory Γ' in time $T(\Gamma)/\alpha$. Moreover Γ is an optimal $(\bar{a}, \bar{v})$-trajectory if and only if Γ' is an optimal $(\bar{a}, \bar{v}/\alpha)$-trajectory.*

Proof. Define Γ' to be the following: $p_{\Gamma'}(0) = p_\Gamma(0)/\alpha^2$, $v_{\Gamma'}(0) = v_\Gamma(0)/\alpha$, and $a_{\Gamma'}(t) = a_\Gamma(\alpha t)$, $0 \le t \le T(\Gamma)/\alpha$. We will show that, for $0 \le t \le T(\Gamma)/\alpha$,

$$v_{\Gamma'}(t) = v_\Gamma(\alpha t)/\alpha \text{ and}$$

$$p_{\Gamma'}(t) = p_\Gamma(\alpha t)/\alpha^2.$$

This is because

$$\begin{aligned} v_{\Gamma'}(t) &= v_{\Gamma'}(0) + \int_0^t a_{\Gamma'}(s)\, ds \\ &= v_\Gamma(0)/\alpha + \int_0^t a_\Gamma(\alpha s)\, ds \\ &= v_\Gamma(0)/\alpha + \int_0^{\alpha t} a_\Gamma(s')(1/\alpha)\, ds' \\ &= v_\Gamma(0)/\alpha + \left(\int_0^{\alpha t} a_\Gamma(s')\, ds' \right) /\alpha \\ &= v_\Gamma(\alpha t)/\alpha \end{aligned}$$

and

$$\begin{aligned} p_{\Gamma'}(t) &= p_{\Gamma'}(0) + \int_0^t v_{\Gamma'}(s)\, ds \\ &= p_\Gamma(0)/\alpha^2 + \int_0^t v_\Gamma(\alpha s)/\alpha\, ds \\ &= p_\Gamma(0)/\alpha^2 + \int_0^{\alpha t} (v_\Gamma(s')/\alpha^2)\, ds' \\ &= p_\Gamma(0)/\alpha^2 + (1/\alpha^2) \int_0^{\alpha t} v_\Gamma(s')\, ds' \\ &= p_\Gamma(\alpha t)/\alpha^2. \end{aligned}$$

Let Π' be the path traversed by Γ', and let ℓ and t be such that$\Pi'(\ell) = p_{\Gamma'}(t)$.

Thus

$$\begin{aligned}\ell &= \int_0^t v_{\Gamma'}(s)ds \\ &= \int_0^t v_\Gamma(\alpha s)/\alpha ds \\ &= (1/\alpha^2)\int_0^{\alpha t} v_\Gamma(s)ds.\end{aligned}$$

This implies that $\Pi(\alpha^2\ell) = p_\Gamma(\alpha t)$. Since

$$\begin{aligned}\Pi'(\ell) &= p_{\Gamma'}(t) \\ &= p_\Gamma(\alpha t)/\alpha^2 \\ &= \Pi(\alpha^2\ell)/\alpha^2,\end{aligned}$$

therefore Π' is $(1/\alpha^2)\Pi$. The scaling preserves the optimality. This completes the proof of the lemma. □

LEMMA 2.17 (acceleration-scaling lemma). *Let Γ be an $(\bar{a}, \bar{v})$-trajectory and Π be its path. The path $(1/\alpha)\Pi$ can be traversed by an $(\bar{a}/\alpha, \bar{v}/\alpha)$-trajectory Γ' in time $T(\Gamma)$. Moreover Γ is an optimal $(\bar{a}, \bar{v})$-trajectory if and only if Γ' is an optimal $(\bar{a}/\alpha, \bar{v}/\alpha)$-trajectory.*

Proof. Define Γ' to be the following: $p_{\Gamma'}(0) = p_\Gamma(0)/\alpha$, $v_\Gamma(0) = v_\Gamma(0)/\alpha$, and $a_{\Gamma'}(t) = a_\Gamma(t)/\alpha$, $0 \le t \le T(\Gamma)$. We can show that, for any $0 \le t \le T(\Gamma)$,

$$v_{\Gamma'}(t) = v_\Gamma(t)/\alpha \text{ and}$$

$$p_{\Gamma'}(t) = p_\Gamma(t)/\alpha.$$

This is because

$$\begin{aligned}v_{\Gamma'}(t) &= v_{\Gamma'}(0) + \int_0^t a_{\Gamma'}(s)\, ds \\ &= v_\Gamma(0)/\alpha + \int_0^t a_\Gamma(s)/\alpha\, ds \\ &= v_\Gamma(0)/\alpha + \left(\int_0^t a_\Gamma(s)\, ds\right)/\alpha \\ &= v_\Gamma(t)/\alpha\end{aligned}$$

and

$$\begin{aligned}p_{\Gamma'}(t) &= p_{\Gamma'}(0) + \int_0^t v_{\Gamma'}(s)\, ds \\ &= p_\Gamma(0)/\alpha + \int_0^t v_\Gamma(s)/\alpha\, ds \\ &= p_\Gamma(0)/\alpha + \left(\int_0^t v_\Gamma(s)\, ds\right)/\alpha \\ &= p_\Gamma(t)/\alpha.\end{aligned}$$

Let Π' be the path traversed by Γ'. Similarly to the proof of Lemma 2.16, we can show that $\Pi' = (1/\alpha)\Pi$. This completes the proof of the lemma. ∎

LEMMA 2.18 (acceleration-velocity scaling lemma). *Let Γ be an $(\bar{a}, \bar{v})$-trajectory and let Π be its path. The path $(\beta/\alpha^2)\Pi$ can be traversed by an $(\bar{a}/\beta, \bar{v}/\alpha)$-trajectory Γ' in time $(\beta/\alpha)\,T(\Gamma)$. Moreover Γ is an optimal $(\bar{a}, \bar{v})$-trajectory if and only if Γ' is an optimal $(\bar{a}/\beta, \bar{v}/\alpha)$-trajectory.*

Proof. Let $\Pi'' = (1/\beta)\Pi$. By Lemma 2.17, Π'' can be traversed by an $(\bar{a}/\beta, \bar{v}/\beta)$-trajectory Γ'' in time $T(\Gamma)$.

Let $\gamma = \alpha/\beta$ and let $\Pi' = (1/\gamma^2)\Pi''$. By Lemma 2.16, Π' can be traversed by an $(\bar{a}/\beta, \bar{v}/(\beta\gamma))$-trajectory in time $T(\Gamma'')/\gamma$. Since

$$\Pi' = (1/\gamma^2)\Pi'' = (1/\gamma^2)(1/\beta)\Pi = (\beta/\alpha^2)\Pi,$$

$$\bar{v}/(\beta\gamma) = \bar{v}/\alpha,$$

and

$$T(\Gamma'')/\gamma = T(\Gamma)/\gamma = (\beta/\alpha)T(\Gamma),$$

therefore this trajectory is the one desired. ∎

By applying the acceleration-velocity scaling lemma, we obtain the following corollary.

COROLLARY 2.19. *Given a kinodynamic motion-planning problem with the following parameters: the size of the configuration space L, the dynamic bounds $\bar{a}$ and $\bar{v}$, the two parameters l and ε, the initial state i, and the final state f, we can scale the problem to one with $\bar{a} = 1$ and $\bar{v} = 1$ by scaling the initial and final velocities by a factor of $(1/\bar{v})$ and scaling the configuration space (including the obstacles, the parameter l, $\text{LOC}(i)$, and $\text{LOC}(f)$) by a factor of $(\bar{a}/\bar{v}^2)$. Let Γ be the trajectory obtained for the scaled problem. Then the trajectory Γ' such that*

$$a_{\Gamma'}(t) = \bar{a} \cdot a_{\Gamma}((\bar{a}/\bar{v})t)$$

for $0 \le t \le (\bar{v}/\bar{a})T(\Gamma)$ is the solution for the original problem.

3. The curvature-constrained shortest-path problem in three and higher dimensions.

3.1. Introduction. Let $P : I \to \mathbb{R}^d$ be a d-dimensional differentiable path parameterized by arclength $s \in I$. The *average curvature* of P in the interval $[s_1, s_2] \subseteq I$ is defined by $\|\dot{P}(s_1) - \dot{P}(s_2)\|/|s_1 - s_2|$. In the curvature-constrained shortest-path problem, we require that the path have an average curvature of at most 1 in every interval. Again, we can always scale an arbitrarily given curvature bound to 1 by scaling the configuration space, thus it is general enough to consider the case when the bound is 1. Notice that we use average curvature instead of curvature because the curvature of a differentiable path may not exist at certain points. Also, the curvature of a path, wherever it is defined, is bounded by 1 if and only if its average curvature is bounded by 1 for all intervals.

A *position* X is a pair $(\text{LOC}(X), \text{ORN}(X))$, where $\text{LOC}(X)$ is a point in the d-dimensional space and $\text{ORN}(X)$ is a vector representing an orientation in the d-dimensional space. Notice that $\|\text{ORN}(X)\| = 1$. Given a set Ω of obstacles in a d-dimensional space, an initial position I, and a final position F, the curvature-constrained shortest-path problem is to find a shortest path from I to F such that the path obeys the curvature constraint and does not intersect Ω.

It has been observed in [18] that the curvature-constrained shortest-path problem is a restricted case of the kinodynamic motion-planning problem, with the L_2-norms of the velocities fixed to be 1. However, if we require that the velocities of the approximation trajectories be fixed in L_2-norm, the techniques developed so far for the general kinodynamic case cannot be applied to this restricted case. As pointed out in [12], a necessary condition for the techniques to apply is that the set of *feasible instantaneous accelerations* (accelerations that can be applied without violating the dynamic constraints) spans d dimensions. On the other hand, if the velocities are fixed in L_2-norm, the set of instantaneous accelerations spans only $d-1$ dimensions, because they have to be perpendicular to the instantaneous velocity.

Our contribution is that we look at the curvature-constrained shortest-path problem from a different viewpoint, which enables us to overcome the difficulty mentioned above. Basically, instead of requiring that the L_2-norms of the velocities be fixed to be 1, we only force them to fall within a small range close to 1. In this way, we are able to obtain a path whose maximum curvature is slightly larger than, but can be arbitrarily close to, 1. It should be noted that under this same variation, the former approximation algorithms of Donald and Xavier [15] and Reif and Tate [43] can also be applied to compute approximating solutions for the curvature-constrained shortest-path problem.

In this section, we use lowercase letters to denote states as defined for the kinodynamic case, but uppercase letters (usually X, Y, U, V) to denote positions (i.e., $\|\textsc{orn}(X)\| = \|\textsc{orn}(Y)\| = \|\textsc{orn}(U)\| = \|\textsc{orn}(V)\| = 1$). A path is called a *c-constrained* path if its average curvature is at most c. We say that Γ is a $(1, \tilde{1})$-trajectory if $\|a_\Gamma(t)\| \leq 1$ and $\|v_\Gamma(t)\| = 1$ for $0 \leq t \leq T(\Gamma)$. Given a path Π, let $L(\Pi)$ be its path length. Notice that the notation of *ngb* extends to positions. Therefore, for a position X, a position $X' \in ngb(X, \rho, \nu)$ if $\|\textsc{loc}(X') - \textsc{loc}(X)\|_\infty \leq \rho$ and $\|\textsc{orn}(X') - \textsc{orn}(X)\|_\infty \leq \nu$.

3.2. Approximating in the absence of obstacles. The following lemma states a result similar to Corollary 2.2 but for the curvature-constrained case. This lemma enables us to apply the method developed in the previous sections to the curvature-constrained shortest-path problem.

LEMMA 3.1. *Let W be a d-dimensional configuration space of size L. Let Π be a 1-constrained path from position X to position Y and lying inside W. Given any $\varepsilon > 0$ and $\rho = O(1)$, we can compute in time $O(L^d(1/\varepsilon)^{6d-1})$ a path Π' such that Π' satisfies the following properties:*

P1. *Π' is $(1+\varepsilon)$-constrained.*
P2. *$L(\Pi') \leq (1+\varepsilon)L(\Pi)$.*
P3. *$d(\Pi', W) \leq \varepsilon\rho/2$.*
P4. *Π' is from some position $X' \in ngb(X, \varepsilon\rho/2, \varepsilon/2)$ to some position $Y' \in ngb(Y, \varepsilon\rho/2, \varepsilon/2)$.*

Proof. Let $\delta = \varepsilon/8$, $\eta_x = \varepsilon\rho/2$, and $\eta_v = \varepsilon/(8\sqrt{d})$.

We consider Π to be the path of a trajectory Γ such that Γ is from state $i = (\textsc{loc}(X), \textsc{orn}(X)/(1+\delta))$ to state $f = (\textsc{loc}(Y), \textsc{orn}(Y)/(1+\delta))$ and $\|v_\Gamma(t)\| = 1/(1+\delta)$. Notice that $T(\Gamma) = (1+\delta)\,L(\Pi)$. Since Π is a 1-constrained path, and at any time t, the curvature of Π is given by $\|a_\Gamma(t)\|/\|v_\Gamma(t)\|^2$, the acceleration of Γ is bounded by

$$\|a_\Gamma(t)\| \leq 1 \cdot \|v_\Gamma(t)\|^2 \leq 1/(1+\delta)^2.$$

This implies that Γ is a $(1/(1+\delta)^2, 1/(1+\delta))$-trajectory. Let $\mathcal{A}_\delta$ be the set of accel-

erations whose L_2-norms are bounded by $1/(1+\delta)^2$. This is the set of accelerations used by Γ.

Let $\mu = \kappa_l = \delta/(4\sqrt{d})$ and $\mathcal{A}_\mu$ be the set of accelerations as defined in the appendix (see (A.5) and (A.6)). Thus $\mathcal{A}_\mu$ has a uniform advantage of κ_l over $\mathcal{A}_\delta$. By the tracking lemma (see the appendix), there exists a $\tau = O(\varepsilon)$ (satisfying (A.4)) and a τ-bang trajectory Γ' using $\mathcal{A}_\mu$ such that $T(\Gamma') = T(\Gamma)$ and Γ' tracks Γ to a tolerance of (η_x, η_v). This implies that

$$\|v_{\Gamma'}(t) - v_\Gamma(t)\|_\infty \le \eta_v,$$

$$\|v_{\Gamma'}(t) - v_\Gamma(t)\| \le \sqrt{d}\eta_v,$$

$$\|v_\Gamma(t)\| - \sqrt{d}\eta_v \le \|v_{\Gamma'}(t)\| \le \|v_\Gamma(t)\| + \sqrt{d}\eta_v,$$

$$\frac{1}{1+\delta} - \sqrt{d}\frac{\varepsilon}{8\sqrt{d}} \le \|v_{\Gamma'}(t)\| \le \frac{1}{1+\delta} + \sqrt{d}\frac{\varepsilon}{8\sqrt{d}},$$

$$\frac{1}{1+\delta} - \frac{\varepsilon}{8} \le \|v_{\Gamma'}(t)\| \le \frac{1}{1+\delta} + \frac{\varepsilon}{8},$$

$$1 - \frac{\varepsilon}{4} \le \|v_{\Gamma'}(t)\| \le 1 + \frac{\varepsilon}{8}.$$

Let Π' be the path of trajectory Γ'. We will show that Π' satisfies P1–P4. Since $\|a_{\Gamma'}(t)\| \le 1$, this bounds the maximum curvature of Π' to be at most

$$\frac{1}{(1-\varepsilon/4)^2} \le 1 + \varepsilon,$$

proving P1. Since $T(\Gamma') \le T(\Gamma) = (1+\delta)L(\Pi)$, then

$$L(\Pi') \le \left(1 + \frac{\varepsilon}{8}\right) T(\Gamma') \le \left(1 + \frac{\varepsilon}{8}\right)(1+\delta)L(\Pi) \le (1+\varepsilon)L(\Pi),$$

proving P2, where $(1 + \frac{\varepsilon}{8})$ is the upper bound on $\|v_{\Gamma'}(t)\|$. P3 follows directly from the fact that Γ' tracks Γ to a tolerance of (η_x, η_v) with $\eta_x = \varepsilon\rho/2$, and that Γ lies inside W.

Let i' and f' be the initial and the final states of Γ', and X' and Y' be the initial and the final positions of Π'. Thus $\textsc{loc}(X') = \textsc{loc}(i')$, $\textsc{loc}(Y') = \textsc{loc}(f')$, $\textsc{orn}(X') = \textsc{vec}(i')/\|\textsc{vec}(i')\|$, and $\textsc{orn}(Y') = \textsc{vec}(f')/\|\textsc{vec}(f')\|$, and

$$\begin{aligned}
&\|\textsc{orn}(X') - \textsc{orn}(X)\|_\infty \\
&\le \|\textsc{orn}(X') - \textsc{orn}(X)\| \\
&= \|\textsc{orn}(X') - \textsc{vec}(i') + \textsc{vec}(i') - \textsc{vec}(i) + \textsc{vec}(i) - \textsc{orn}(X)\| \\
&\le \|\textsc{orn}(X') - \textsc{vec}(i')\| + \|\textsc{vec}(i') - \textsc{vec}(i)\| + \|\textsc{vec}(i) - \textsc{orn}(X)\| \\
&\le (1 - \|\textsc{vec}(i')\|) + \sqrt{d}\eta_v + \frac{\delta}{1+\delta} \\
&\le \frac{\varepsilon}{4} + \frac{\varepsilon}{8} + \frac{\varepsilon}{8} \\
&= \frac{\varepsilon}{2}.
\end{aligned}$$

Similarly we can show that $\|\textsc{orn}(Y') - \textsc{orn}(Y)\|_\infty \le \varepsilon/2$. This proves P4 for Γ'.

Such a trajectory Γ' (and thus a path Π') can be obtained by the TC-graph method. In constructing the graph, an edge is added if (i) it is a (μ, τ)-bang, (ii) the bang does not diverge from W by more than $\varepsilon\rho/2$, and (iii) the L_2-norms of the velocities of the bang fall into $[1-\varepsilon/4, 1+\varepsilon/8]$. Since μ and τ are $O(\varepsilon)$, the number of edges in this graph is bounded by $O(L^d(1/\varepsilon)^{6d-1})$, and thus so is the running time. This completes the proof of the lemma. □

Notice that we do not explicitly constrain the velocities of the tracking trajectory Γ'. This allows us to apply the tracking lemma (Lemma A.1). However, it happens that $\|v_{\Gamma'}(t)\|$ falls into a small range of $[1-\varepsilon/4, 1+\varepsilon/8]$ by being able to track closely a trajectory Γ whose velocities are fixed to be 1 in L_2-norm. By upper bounding the accelerations and lower bounding the velocities of Γ', we are able to bound the curvature of Π'.

3.3. Approximating with obstacles. Let W be a d-dimensional configuration space of size L with a set Ω of obstacles. Let $\mathcal{B}$ be the box decomposition of W, as described in section 2.3.1. Let Π be an optimal $3l$-safe 1-constrained path from position X to position Y.

As we did in the proof of the safe loose-tracking theorem (Theorem 2.11), we can divide Π into segments of paths, $\Pi_{U_0U_1} \| \dots \| \Pi_{U_{m-1}U_m}$, such that $U_0 = X$, $U_m = Y$, and each U_i is the position where Π first exits $\xi(U_{i-1})$ for $1 \le i \le m$. Each segment $\Pi_{U_iU_{i+1}}$ is a 1-constrained path from U_i to U_{i+1} and lies inside $\xi(U_i)$. These are the paths we need to approximate.

We define the set of CS grid states, the canonical extended boxes, the canonical starting states, and the canonical ending states in the same way as we did in the previous sections, except that we let $\mathcal{V}$ be the set of *unit* vectors that are uniformly spaced with spacing δ. If $\delta = O(\epsilon)$ is chosen small enough, for any unit vector v, there is a unit vector $v' \in \mathcal{V}$ such that $\|v' - v\|_\infty \le \epsilon/2$. Thus $\mathcal{V}$ suffices for the curvature-constrained case, since if we consider U_i as a state then $\|\textsc{vec}(U_i)\| = 1$ for all U_i's along Π. The following corollary can be derived from the correcting lemma (Lemma 2.6).

Corollary 3.2. *Fix a constant $c > 1$ and let $\rho_c = 1/(c\kappa)$, where $\kappa > 0$ is arbitrary. Let Π be a κ-constrained path from position I to position F. Given any $\rho > 0$ and $\varepsilon > 0$ and given any two positions U and V, if $\rho > \rho_c$, $\|\textsc{loc}(F) - \textsc{loc}(I)\|_\infty \ge \rho$, $U \in ngb(I, \varepsilon\rho/2, \varepsilon/2)$, and $V \in ngb(F, \varepsilon\rho/2, \varepsilon/2)$, we can construct a path Π' from U to V by correcting Π such that Π' satisfies the following properties:*

P1. $L(\Pi') = (1 + e\varepsilon)L(\Pi)$,

P2. Π' *is* $((1+e\varepsilon)^2/(1-e\varepsilon)^2)\kappa$*-constrained,*

P3. $d(\Pi', \Pi) \le (17/16)\varepsilon\rho$,

where $e = 4c\sqrt{d}$.

The consequence of Lemma 3.1 and Corollary 3.2 is that we are able to precompute paths which approximate U_{ii+1}'s and store them in the connection tables. Similar to what we did for the kinodynamic case, we can prove a version of the loose-tracking lemma (Lemma 2.8) and the safe loose-tracking theorem (Theorem 2.11) for the curvature-constrained case. Since $|\mathcal{V}| = O((1/\epsilon)^{d-1})$ (as opposed to $O((1/\epsilon)^d)$ for the kinodynamic case), the number of edges in the CS graph is reduced to $O((1/\epsilon)^{4d-4})$.

Combining the above, and choosing small enough $\epsilon = O(\varepsilon)$, we can obtain the following result.

Theorem 3.3. *Fix an $l > 0$ and an $\varepsilon > 0$. After some precomputation, we can achieve the following.*

Let W be a d-dimensional space of size L and let Ω be a set of obstacles with a

total of n vertices. Given any two positions X and Y, we can compute a collision-free $(1+\varepsilon)$-constrained path from a position $X' \in ngb(X, \varepsilon l, \varepsilon)$ to a position $Y' \in ngb(Y, \varepsilon l, \varepsilon)$, such that the path length is at most $(1+\varepsilon)$ times the length of an optimal 1-constrained $3l$-safe path from X to Y.

The running time of our algorithm is $O(nN + N \log N (1/\varepsilon)^{4d-4})$, where $N = O((L/l)^d)$.

4. Conclusions. In this paper we have presented a faster approximation algorithm for the kinodynamic motion-planning problem. Contrary to the previous approximation algorithms which all use a uniform discretization in time, our method employs a nonuniform discretization in the configuration space. The discretization is nonuniform in that it is coarser in regions which are farther from all obstacles. The nonuniform discretization leads to a smaller search space and thus a faster algorithm. Moreover, our method is sensitive to the distribution of obstacles in the configuration space. It is expected to perform better (i.e., faster) than the given theoretical time bound in cases where the obstacles are sparsely or unevenly distributed. In developing the approximation algorithm, we utilize a hierarchical decomposition of the configuration space. We also developed the idea of using trajectories precomputed in the absence of obstacles as building blocks to construct trajectories in the presence of obstacles. Finally, we have applied this algorithm to give the first-known polynomial-time approximation algorithm for the curvature-constrained shortest-path problem in three and higher dimensions. The computed paths may have curvatures slightly larger (but can be arbitrarily close to) the given curvature bound.

It should be pointed out that, though improved, the time complexity of our approximation algorithm is still too high for the method to have real-world applications. At this point, the proposed method and results are of pure theoretical interest.

One future research direction is to give a more precise bound on N, the number of boxes in the box decomposition. Instead of describing N as a function of L, the size of the configuration space, we would prefer to describe N as a function of some parameters that describe the obstacle scene, for example, the aspect ratios of the obstacles.

In the kinodynamic motion-planning problem, we only considered *Cartesian robots*, i.e., robots whose inertia tensor is constant (see [15] for a more precise definition). We believe that our approach can also be applied to more general robots with *open kinematic chains*. In order to do this, we need to generalize our correcting lemma (Lemma 2.6) to open chains.

A key further problem is to determine cases of kinodynamic motion-planning problems that we can solve in closed forms, or for which we can give sufficient characterizations for developing fast algorithms. An example along this line is the case of planning for a point robot moving on a plane amid polygonal obstacles and with decoupled kinodynamic constraints. It is characterized in [9] that the minimum-time trajectory is a sequence of segments, where each segment is a "bang-bang" control between two obstacle boundary points. It is interesting to investigate whether such characterizations extend to coupled cases and higher dimensions.

The complexity of the kinodynamic motion-planning problem in two dimensions is still open.

Appendix. The *TC*-graph method. For the sake of completeness, we describe the gist of the tracking lemma (Lemma A.1) and the TC-graph method in this section.

A trajectory Γ' is said to *track* another trajectory Γ to a tolerance (η_x, η_v) if for

all $t \in [0, T]$,

$$\|p_\Gamma(t) - p_{\Gamma'}(t)\|_\infty \leq \eta_x \text{ and} \tag{A.1}$$

$$\|v_\Gamma(t) - v_{\Gamma'}(t)\|_\infty \leq \eta_v. \tag{A.2}$$

Notice that (A.1) implies that $d(\Gamma', \Gamma) \leq \eta_x$.

A *τ-bang* is a trajectory segment of time duration τ during which a *constant* acceleration is applied.[2] A *τ-bang trajectory* is a trajectory consisting of a sequence of τ-bangs. The set of bang trajectories is a restricted set of trajectories. But they suffice to track any nonrestricted trajectories if the acceleration set used by the bang trajectories has some *advantage* over the acceleration set used by the nonrestricted trajectories. Next we define the notion of advantage more formally.

Let $\mathcal{P}$ and $\mathcal{Q}$ be two sets of accelerations. For any acceleration $a \in \mathcal{P}$ and any direction given by a d-length vector σ of 1's and -1's, if there exists an acceleration $b \in \mathcal{Q}$ such that

$$\sigma^j(b^j - a^j) \geq \kappa_l \tag{A.3}$$

for $1 \leq j \leq d$, then $\mathcal{Q}$ is said to have a *uniform κ_l advantage* over $\mathcal{P}$ (α^j means the jth element of a vector α). The tracking lemma relates the parameter τ to the uniform advantage κ_l and the tracking tolerance (η_x, η_v).

LEMMA A.1 (tracking lemma [15]).[3] *Let $\mathcal{P}$ and $\mathcal{Q}$ be two sets of accelerations such that for each $a \in \mathcal{Q}$, $\|a\| \leq 1$, and $\mathcal{Q}$ has a uniform κ_l advantage over $\mathcal{P}$.*

Let Γ be a trajectory that uses $\mathcal{P}$. Let (η_x, η_v) be a tracking tolerance. There exists a time step τ, and a τ-bang trajectory Γ' that uses $\mathcal{Q}$ such that Γ' tracks Γ to tolerance (η_x, η_v).

Moreover, it is sufficient that

$$\tau = O(\min(\eta_v, \sqrt{\eta_x \kappa_l})). \tag{A.4}$$

Let $\mathcal{A}$ (resp., $\mathcal{A}_\epsilon$) be the set of accelerations whose L_2-norms are bounded by 1 (resp., $1/(1+\epsilon)^2$). Next we show how we can choose a finite set of accelerations such that this set has a uniform advantage over $\mathcal{A}_\epsilon$. Given a parameter μ, a set of grid-points of $\mathcal{A}$ with spacing μ is defined to be

$$\{a = (i_1\mu, \ldots, i_d\mu) \mid i_1, \ldots, i_d = 0, \pm 1, \pm 2, \ldots, a \in \mathcal{A}\}. \tag{A.5}$$

Let $\mathcal{A}_\mu$ be the set of *boundary* grid-points[4] (see Figure A.1; the dark dots represent the boundary grid-points in two dimensions). It is shown in [15] that if μ is chosen

[2] Here we slightly abuse the notion of a *bang*. A bang usually means that its acceleration should be *extremal* in some sense.

[3] This is a simplified version of the tracking lemma presented in [15]. In [15], $\mathcal{Q}$ is a set of *instantaneous accelerations* that depends on the current state, and may be a subset of the set of allowed accelerations. This is because applying certain accelerations may violate the velocity constraint. In the context where we apply this tracking lemma, we do not explicitly bound the velocities of the tracking trajectory. This means that the set of instantaneous accelerations equals the set of all allowed accelerations.

[4] In [15], two approximation algorithms, the true-extremal algorithm and the near-extremal algorithm, are described. They differ in the way of choosing a discretized acceleration set and in the way of defining bangs. The description here follows from the near-extremal algorithm.

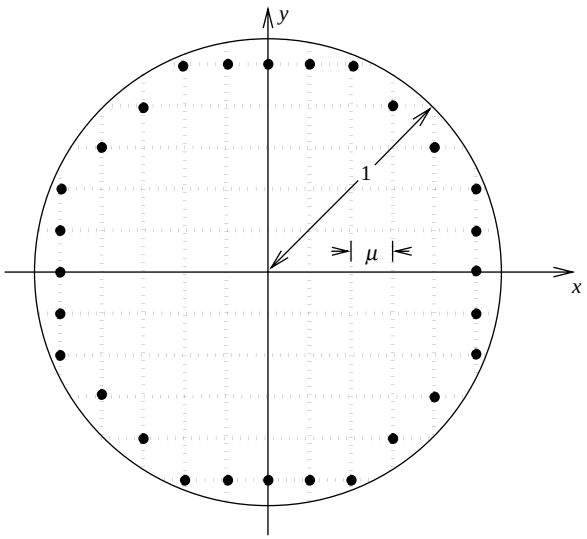


FIG. A.1. *The acceleration set $\mathcal{A}_\mu$ in two dimensions.*

such that

$$\mu \le \kappa_l \le \frac{\epsilon}{4\sqrt{d}}, \tag{A.6}$$

then $\mathcal{A}_\mu$ has a uniform advantage of κ_l over $\mathcal{A}_\epsilon$. Notice that $|\mathcal{A}_\mu| = O(d(1/\mu)^{d-1})$.

A τ-bang (resp., τ-bang trajectory) is a (μ,τ)-*bang* (resp., (μ,τ)-*bang trajectory*) if the acceleration set used is $\mathcal{A}_\mu$. Given a tracking tolerance (η_x,η_v) and any $\epsilon > 0$, there exist μ (satisfying (A.6)) and τ (satisfying (A.4)) such that for any $(1/(1+\epsilon)^2, 1/(1+\epsilon))$-trajectory Γ, there exists a (μ,τ)-bang trajectory Γ' (also a $(1,1)$-trajectory) that tracks Γ to a tolerance of (η_x,η_v). To approximate any given $(1,1)$-trajectory Γ to within a factor of $(1+\epsilon)$ in time length, we first time-rescale Γ to a $(1/(1+\epsilon)^2, 1/(1+\epsilon))$-trajectory Γ', with time length extended by a factor of $(1+\epsilon)$. Since there exists a (μ,τ)-bang trajectory Γ'' that tracks Γ' to a tolerance of (η_x,η_v), Γ'' approximates Γ in that $T(\Gamma'') \le (1+\epsilon)\,T(\Gamma)$. But Γ'' does not track Γ to the tolerance of (η_x,η_v). However, $d(\Gamma'',\Gamma) \le \eta_x$. This is because Γ' and Γ trace the same path and $d(\Gamma'',\Gamma') \le \eta_x$. Also the two end states of Γ'' are close to the two end states of Γ, respectively.

Having discovered the existence of a tracking bang trajectory, the next question is how to compute one. What makes it more difficult is that most of the time, the original $(1,1)$-trajectory Γ is not given except in its initial and final states. The *TC*-graph method transforms this problem to that of finding a shortest path in a directed graph. (In the following, we only describe the *TC*-graph method in the absence of obstacles, where we do not have to do collision-free checking.) The *TC* graph G is roughly as follows: (i) the *root node* of G approximates the initial state of Γ; (ii) a directed edge corresponds to a (μ,τ)-bang whose velocities are also bounded by 1 in L_2-norm; the weight of the edge is always τ; (iii) the graph is generated and explored from the root node in a breadth-first manner, and the search terminates when either a node approximating the final state is found or when no new nodes aregenerated.

Note the following:

- The size of G is finite (see the following analysis).
- The breadth-first search produces a trajectory whose time length is no more than $(1+\epsilon)\,T(\Gamma)$. The breadth-first search suffices for finding a shortest path since each edge has a uniform weight of τ.
- Suppose that Γ lies within a subset W of the configuration space. Since there exists a bang trajectory Γ'' satisfying $d(\Gamma'', \Gamma') \leq \eta_x$, then $d(\Gamma'', W) \leq \eta_x$. We can find such a trajectory by considering only those edges whose corresponding bangs do not diverge from W by more than η_x. Notice that it still holds that $T(\Gamma'') \leq (1+\epsilon)\,T(\Gamma)$.

Each node of G corresponds to a state. By carefully choosing the initial velocity, we can bound the number of possible velocities by $O((1/(\mu\tau))^d)$ and the number of possible locations by $O((L/(\mu\tau^2))^d)$, where L is the size of W. Thus the total number of nodes is bounded by

$$O\left(\left(\frac{L}{\mu^2\tau^3}\right)^d\right). \tag{A.7}$$

Each node can have at most $O(d(1/\mu)^{d-1})$ outgoing edges (because it can have at most this many choices of accelerations), thus the total number of graph edges is

$$O\left(\frac{dL^d}{\mu^{3d-1}\tau^{3d}}\right). \tag{A.8}$$

This also bounds the running time.

If we set

$$\eta_x = \epsilon\rho/2 \text{ and } \eta_v = \epsilon/2, \tag{A.9}$$

where $\rho = O(1)$, we obtain Corollary 2.2. Notice that since $\tau = O(\epsilon)$ and $\mu = O(\epsilon)$, the running time is $O(L^d(1/\epsilon)^{6d-1})$.

REFERENCES

[1] P. K. Agarwal, P. Raghavan, and H. Tamak, *Motion planning for a steering-constrained robot through moderate obstacles*, Proc. Annual Symposium Theory Comput., 27 (1995), pp. 343–352.

[2] J. Barraquand and J. C. Latombe, *Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles*, Algorithmica, 10 (1993), pp. 121–155.

[3] J. D. Boissonnat and X. N. Bui, *Accessibility Region for a Car that Only Moves Forwards along Optimal Paths*, Technical report, INRIA, Sophia Antipolis, France, 1994.

[4] J. D. Boissonnat, A. Cerezo, and J. Leblond, *Shortest paths of bounded curvature in the plane*, in Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 2315–2320.

[5] J. D. Boissonnat, A. Cerezo, and J. Leblond, *A Note on Shortest Paths in the Plane Subject to a Constraint on the Derivative of the Curvature*, Technical report, INRIA, Sophia Antipolis, France, 1994.

[6] X. N. Bui, J. D. Boissonnat, P. Soueres, and J. P. Laumond, *Shortest path synthesis for Dubins non-holonomic robot*, in Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, 1994, pp. 2–7.

[7] J. Canny, *Some algebraic and geometric configuration in* PSPACE, Proc. Annual ACM Sympos. Theory Comput., 20 (1988), pp. 460–467.

[8] J. Canny, B. Donald, J. Reif, and P. Xavier, *On the complexity of kinodynamic planning*, Proc. Sympos. Found. Comput. Sci., 29 (1988), pp. 306–316.

[9] J. Canny, A. Rege, and J. Reif, *An exact algorithm for kinodynamic planning in the plane*, Discrete Comput. Geom., 6 (1991), pp. 461–484.
[10] J. Canny and J. Reif, *New lower bound techniques for robot motion planning problems*, Proc. Sympos. Found. Comput. Sci., 28 (1987), pp. 49–60.
[11] B. Donald and P. Xavier, *A provably good approximation algorithm for optimal-time trajectory planning*, in Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, AZ, 1989, pp. 958–963.
[12] B. Donald and P. Xavier, *Near-optimal kinodynamic planning for robots with coupled dynamics bounds*, in Proceedings of the IEEE International Symposium on Intelligent Controls, Albany, NY, 1989, pp. 354–359.
[13] B. Donald and P. Xavier, *Provably good approximation algorithms for optimal kinodynamic planning for Cartesian robots and open chain manipulators*, Proc. Sympos. Comput. Geom., 6 (1990), pp. 290–300.
[14] B. Donald and P. Xavier, *Provably good approximation algorithms for optimal kinodynamic planning: Robots with decoupled dynamics bounds*, Algorithmica, 14 (1995), pp. 443–479.
[15] B. Donald and P. Xavier, *Provably good approximation algorithms for optimal kinodynamic planning for cartesian robots and open chain manipulators*, Algorithmica, 14 (1995), pp. 480–530.
[16] B. Donald, P. Xavier, J. Canny, and J. Reif, *Kinodynamic motion planning*, J. Assoc. Comput. Mach., 40 (1993), pp. 1048–1066.
[17] L. E. Dubins, *On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents*, Amer. J. Math., 79 (1957), pp. 497–516.
[18] S. Fortune and G. Wilfong, *Planning constrained motion*, Ann. Math. Artificial Intelligence, 3 (1991), pp. 21–82.
[19] T. Fraichard, *Smooth trajectory planning for a car in a structured world*, in Proceedings of the IEEE International Conference on Robotics and Automation, Sacramento, CA, 1991, pp. 2–7.
[20] G. Heinzinger, P. Jacobs, J. Canny, and B. Paden, *Time-optimal trajectories for a robot manipulator: a provably good approximation algorithm*, in Proceedings of the IEEE International Conference on Robotics and Automation, Cincinnati, OH, 1990, pp. 150–155.
[21] J. Hershberger and S. Suri, *Efficient computation of Euclidean shortest paths in the plane*, in Proceedings of the 34th Symposium on Foundations of Computer Science, Palo Alto, CA, 1993, pp. 508–517.
[22] J. M. Hollerbach, *Dynamic scaling of manipulator trajectories*, in Proceedings Automatic Control Council, 1983, pp. 752–756.
[23] P. Jacobs and J. Canny, *Planning smooth paths for mobile robots*, in Nonholonomic Motion Planning, Kluwer Academic Publishers, Norwell, MA, 1992.
[24] P. Jacobs, J. P. Laumond, and M. Taix, *Efficient motion planners for nonholonomic mobile robots*, in Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, Osaka, Japan, 1991, pp. 1229–1235.
[25] L. Kavraki and J. C. Latombe, *Randomized preprocessing of configuration space for fast path planning*, in Proceedings of the IEEE International Conference on Robotics and Automation, San Diego, CA, 1994, pp. 2138–2145.
[26] L. Kavraki, J. C. Latombe, R. Motwani, and P. Raghavan, *Randomized query processing in robot path planning*, in Proceedings of the 27th Sympos. Theory Computing, Las Vegas, NV, 1995, pp. 353–362.
[27] V. P. Kostov and E. V. Degtiariova-Kostova, *Suboptimal Paths in the Problem of a Planar Motion with Bounded Derivative of the Curvature*, Technical report, INRIA, Cedex, France, 1993.
[28] V. P. Kostov and E. V. Degtiariova-Kostova, *The Planar Motion with Bounded Derivative of the Curvature and Its Suboptimal Paths*, Technical report, INRIA, Cedex, France, 1994.
[29] J. C. Latombe, *A fast path-planner for a car-like indoor mobile robot*, in Proceedings of the Ninth National Conference on Artificial Intelligence, Anaheim, CA, 1991, pp. 659–665.
[30] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA, 1991.
[31] J. P. Laumond, *Finding collision free smooth trajectories for a nonholonomic mobile robot*, Proc. Internat. Joint Conf. Artificial Intelligence, 10 (1987), pp. 1120–1123.
[32] J. P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray, *A motion planner for nonholonomic mobile robots*, IEEE Trans. Robotics and Automation, 10 (1994), pp. 577–593.
[33] J. P. Laumond and T. Simeon, *Motion Planning for a Two Degrees of Freedom Mobile Robot with Towing*, Technical report, LAAS/CNRS, LAAS, Toulouse, France, 1989.
[34] J. P. Laumond, M. Taix, and P. Jacobs, *A motion planner for car-like robots based on a*

global/local approach, in Proc. IEEE/RSJ International Workshop on Intelligent Robots and Systems, Tsuchira, Japan, 1990, pp. 765–773.

[35] G. L. Miller, D. Talmor, S. Teng, and N. Walkington, *A Delaunay based numerical method for three dimensions: Generation, formulation, and partition*, Proc. Sympos. on Theory of Computing, 27 (1995), pp. 683–692.

[36] G. L. Miller, S. Teng, W. Thurston, and S. A. Vavasis, *Automatic mesh partitioning*, in Graph Theory and Sparse Matrix Computation, A. George, J. Gilbert, and J. Liu, eds., Springer-Verlag, New York, 1993, pp. 57–84.

[37] B. Mirtich and J. Canny, *Using skeletons for nonholonomic path planning among obstacles*, in Proceedings of the IEEE International Conference on Robotics and Automation, Nice, France, 1992, pp. 2533–2540.

[38] J. S. B. Mitchell, D. M. Mount, and S. Suri, *Query-sensitive ray shooting*, Proc. Sympos. Comput. Geom., 10 (1994), pp. 359–368.

[39] Y. Nakamura and R. Mukherjee, *Nonholonomic path planning and automation*, in Proceedings of the IEEE International Conference on Robotics and Automation, Scottsdale, AZ, 1989, pp. 1050–1055.

[40] C. Ó'Dúnlaing, *Motion planning with inertial constraints*, Algorithmica, 2 (1987), pp. 431–475.

[41] J. A. Reeds and L. A. Shepp, *Optimal paths for a car that goes both forwards and backwards*, Pacific J. Math., 145 (1990), pp. 367–393.

[42] J. Reif, *Complexity of the generalized movers problem*, in Planning, Geometry and Complexity of Robot Motion, J. Hopcroft, J. Schwartz, and M. Sharir, eds., Ablex, Norwood, NJ, 1987, pp. 267–281.

[43] J. H. Reif and S. R. Tate, *Approximate kinodynamic planning using L_2-norm dynamic bounds*, Comput. Math. Appl., 27 (1994), pp. 29–44.

[44] G. Sahar and J. M. Hollerbach, *Planning of Minimum-Time Trajectories for Robot Arms*, Technical report, MIT, Cambridge, MA, 1984.

[45] J. T. Schwartz and M. Sharir, *Algorithmic motion planning in robotics*, in Algorithms and Complexity, J. V. Leeuwen, ed., Handbook of Theoretical Computer Science, Vol. A, Elsevier, Amsterdam, 1990, pp. 391–430.

[46] H. Wang and P. K. Agarwal, *Approximation algorithms for curvature-constrained shortest paths*, in Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 409–418.

[47] G. Wilfong, *Motion planning for an autonomous vehicle*, in Proceedings of the IEEE International Conference on Robotics and Automation, 1988, pp. 529–533.

[48] G. Wilfong, *Shortest paths for autonomous vehicles*, in Proceedings of the IEEE International Conference on Robotics and Automation, 1989, pp. 15–20.

[49] P. Xavier, *private communication*, 1998.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 191–217

ALLOCATING BANDWIDTH FOR BURSTY CONNECTIONS*

JON KLEINBERG†, YUVAL RABANI‡, AND ÉVA TARDOS§

Abstract. In this paper, we undertake the first study of statistical multiplexing from the perspective of approximation algorithms. The basic issue underlying statistical multiplexing is the following: in high-speed networks, individual connections (i.e., communication sessions) are very *bursty*, with transmission rates that vary greatly over time. As such, the problem of packing multiple connections together on a link becomes more subtle than in the case when each connection is assumed to have a fixed demand.

We consider one of the most commonly studied models in this domain: that of two communicating nodes connected by a set of parallel edges, where the rate of each connection between them is a random variable. We consider three related problems: (1) stochastic load balancing, (2) stochastic bin-packing, and (3) stochastic knapsack. In the first problem the number of links is given and we want to minimize the expected value of the maximum load. In the other two problems the link capacity and an allowed *overflow probability* p are given, and the objective is to assign connections to links, so that the probability that the load of a link exceeds the link capacity is at most p. In bin-packing we need to assign each connection to a link using as few links as possible. In the knapsack problem each connection has a value, and we have only one link. The problem is to accept as many connections as possible.

For the stochastic load balancing problem we give an $O(1)$-approximation algorithm for arbitrary random variables. For the other two problems we have algorithms restricted to on-off sources (the most common special case studied in the statistical multiplexing literature), with a somewhat weaker range of performance guarantees.

A standard approach that has emerged for dealing with probabilistic resource requirements is the notion of *effective bandwidth*—this is a means of associating a fixed demand with a bursty connection that "represents" its distribution as closely as possible. Our approximation algorithms make use of the standard definition of effective bandwidth and also a new one that we introduce; the performance guarantees are based on new results showing that a combination of these measures can be used to provide bounds on the optimal solution.

Key words. combinatorial optimization, approximation algorithms, statistical multiplexing, effective bandwidth

AMS subject classifications. 05C85, 68R10, 68Q20

PII. S0097539797329142

1. Introduction.

Motivation and previous work. The issues of admission control and routing in high-speed networks have inspired recent analytical work on network routing and bandwidth allocation problems in several communities (e.g., [10, 1, 5]). One

*Received by the editors October 24, 1997; accepted for publication (in revised form) June 21, 1999; published electronically April 25, 2000. Research for this paper was supported in part by NSF through grant DMS 9505155. Portions of this work were performed while visiting the IBM Almaden Research Center.

http://www.siam.org/journals/sicomp/30-1/32914.html

†Cornell University, Department of Computer Science, Ithaca, NY 14853 (kleinberg@cs.cornell.edu). This author is supported in part by an Alfred P. Sloan Research Fellowship, an ONR Young Investigator Award, and NSF Faculty Early Career Development Award CCR-9701399.

‡Technion Institute, Computer Science Department, Haifa 32000, Israel (rabani@cs.technion.ac.il). Most of this work was done while visiting Cornell University. Support at Cornell was provided by ONR through grant N00014-96-1-0050. This work was also supported by grants from the fund for the promotion of sponsored research and the fund for the promotion of research at the Technion, and by a David and Ruth Moskowitz Academic Lecturship award.

§Cornell University, Department of Computer Science, Ithaca, NY 14853 (eva@cs.cornell.edu). The research of this author was supported in part by an NSF PYI award DMI-9157199, by NSF through grant DMS 9505155, and by ONR through grant N00014-96-1-0050.

line of work has been directed towards the development of *approximation algorithms* and *competitive on-line algorithms* for admission control and virtual circuit routing problems (see the survey by Plotkin [16]). The network model in this line of work represents the links of the network as edges of fixed *capacity* and connections as pairs of vertices with a fixed *bandwidth demand* between them. The algorithms and their analysis are motivated by this network flow perspective.

In fact, however, traffic in high-speed networks based on asynchronous transfer mode (ATM) and related technologies tends to be extremely *bursty*. The transmission rate of a single connection can vary greatly over time; there can be infrequent periods of very high *peak rate*, while the *average rate* is much lower.

One can try to avoid this issue by assigning each connection a demand equal to its maximum possible rate. The use of such a conservative approximation will ensure that edge capacities are never violated. But much of the strength of ATM comes from the advantage of *statistical multiplexing*—the packing of uncorrelated, bursty connections on the same link. In particular, suppose one is willing to tolerate a low rate of packet loss due to occasional violations of the link capacity. As the "peak" states of different connections coincide only very rarely, one can pack many more connections than is possible via the above worst-case approach and still maintain a very low rate of packet loss due to overflow.

Queueing theorists recently have devoted a great deal of study to the analysis of statistical multiplexing (see the book edited by Kelly, Zachary, and Zeidins [13]). Typically, this work models a single connection either as a discrete random variable X, with $\Pr[X = s]$ indicating the fraction of the time that the connection transmits at rate s, or as a finite-state Markov chain with a fixed transmission rate for each state. (A much-discussed case is when X is an *on-off source*. In our context, such a connection is equivalent to a weighted Bernoulli trial.) This line of work has concentrated primarily on the case of point-to-point transmission across a set of parallel links; this allows one to study the packing and load balancing issues that arise without the added complication of path selection in a large network.

One of the main concepts that has emerged from this work has been the development of a notion of *effective bandwidth* for bursty connections. This is based on the following natural idea. Suppose one is willing to tolerate a rate p of overflow on each link. One first assigns a number $\beta_p(X)$ to each connection (i.e., random variable) X, indicating the "effective" amount of bandwidth required by this connection. One then uses a standard packing or load balancing algorithm to assign connections to links, using the single number $\beta_p(X)$ as the *demand* of the connection X. This notion of effective bandwidth is indeed what underlies the modeling of routing problems as network flow questions.

Consensus has more or less been reached (see Kelly [12]) on a specific formula for β_p, first studied by Hui [10]: a scaled logarithm of the moments-generating function of X. One of its attractions is that packing according to $\beta_p(X)$ always provides a relatively *conservative estimate* in the following sense: If the sum of the effective bandwidths of a set of independent connections does not exceed the link capacity, then the probability that the sum of their transmission rates exceeds twice the capacity at any instant is at most p.

Problems studied in this paper. In this paper, we undertake the first study of the issues inherent in statistical multiplexing from the perspective of approximation algorithms. We are motivated primarily by the following fact: the queueing theoretical work discussed above does not attempt to prove that its methods, based on

effective bandwidth, provide solutions that are *near-optimal* on all (or even on typical) instances. Indeed, researchers have recognized that claims about the power of the effective bandwidth approach depend critically on a number of fundamental assumptions about the nature of the underlying traffic (e.g., de Veciana and Walrand [18]). Thus an analysis of statistical multiplexing problems in the framework of approximation algorithms can provide tools for understanding the performance guarantees that can be attained in this domain.

We mentioned above that the model studied in this area concentrates primarily on the case of two communicating nodes connected by a set of parallel edges. Thus, the problem of assigning bursty *connections* to *edges* is equivalent to that of assigning (bursty) *items* to *bins*. As a result, we have a direct connection between the standard questions addressed in statistical multiplexing and stochastic versions of some of the classical resource allocation problems in combinatorial optimization. We design and analyze approximation algorithms for the following fundamental problems:

Stochastic load balancing. An *item* is a discrete random variable. We are given items $X_1, \ldots, X_n$. We want to assign each item to one of the bins $1, \ldots, m$ so as to minimize the *expected maximum weight* in any bin. That is, we want to minimize

$$\mathrm{E}\left[\max_i \sum_{X_j \in B_i} X_j\right],$$

where B_i is the set of items assigned to bin i.

Stochastic bin-packing. We are given items as above, and we define the *overflow probability* of a subset of these items to be the probability that their sum exceeds 1. We are also given a number $p \geq 0$. We want to determine the minimum number of bins (of capacity 1) that we need in order to pack all the items, so that the overflow probability of the items in each bin is at most p.

Stochastic knapsack. We are given $p \geq 0$ and a set of items $X_1, \ldots, X_n$, with item X_i having a *value* v_i. We want to find a subset of the items of maximum value, subject to the constraint that its overflow probability is at most p.

Thus, the above problems provide us with a very concrete setting in which to try assessing the power of various approaches to the statistical multiplexing of bursty connections. These problems are also the natural stochastic analogues of some of the central problems in the area of approximation algorithms; and hence we feel that their approximability is of basic interest.

Of course, each of these problems is NP-hard, since the versions in which each item X_i is *deterministic* (i.e., takes a single value with probability 1) correspond to the *minimum makespan*, *bin-packing*, and *knapsack* problems, respectively. However, the stochastic versions introduce considerable additional complications. For example, we show that even *given* a set of items, determining its overflow probability is $\#P$-complete (see section 2).

Moreover, we also show that simple approaches such as (i) applying Hui's definition of effective bandwidth [10] to the items, and then (ii) running a standard algorithm for the case of deterministic weights (e.g., Graham's lowest-fit makespan algorithm or first-fit for bin packing) can lead to results that are very far from optimal. Indeed, we show in section 2 that in a certain precise sense there is no "direct" use of effective bandwidth that can provide approximation results as strong as those we obtain.

1.1. Our results. This paper provides the first approximation algorithms for these load balancing and packing problems with stochastic items. Our algorithms make use of effective bandwidth, and their analysis is based on new results showing, roughly, that it is possible to define a notion of *effective bandwidth* that can be used to obtain bounds on the value of the optimum.

However, the relationships between the effective bandwidth and the optimum are quite subtle. In particular, while Hui's definition is a useful ingredient in our algorithm for the case of load balancing, we show in the cases of bin-packing and knapsack that it is *necessary* to use a definition of effective bandwidth that is *different* from the standard one. Our new effective bandwidth function β' has a number of additional properties that make its analysis particularly tractable. In particular, it was through β' that we were able to establish our basic relations between the function β and the value of the optimum for the case of load balancing.

Load balancing. Perhaps our strongest result is for the load balancing problem: we provide a constant-factor approximation algorithm for the optimum load for arbitrary random variables. With a somewhat larger constant, we can modify our algorithm to work in an *on-line* setting, in which items arrive in sequence and must be assigned to bins immediately.

Let us give some indication of the techniques underlying this algorithm. First, we mentioned above that the standard effective bandwidth β_p comes with an *upper bound* guarantee: if the sum of the effective bandwidths of a set of items is bounded by 1, then the probability that the total load of these items exceeds 2 is at most p. (This fact is due originally to Hui [10] and has been extended and generalized by Kelly [11], Elwalid and Mitra [4], and others.)

Our proof of the constant approximation ratio uses a new *lower bound* guarantee for effective bandwidth. Suppose we have a set of random variables $X_1, \ldots, X_n$, so that each X_i is a weighted Bernoulli trial taking on the values 0 and 2^{-i} for an integer $0 \leq i \leq \log\log p^{-1}$. We show that there is an absolute constant $C \leq 7$ so that if the sum of the effective bandwidths of the X_i is at least C, then the probability that their sum exceeds 1 is *at least* p.

A number of issues must be resolved in order to use these bounds in the design and analysis of our algorithm. First, the upper bound guarantee holds only under some restricting assumptions on the item sizes, which are not necessarily valid for our input. Therefore, we have to handle *exceptional* items separately. Second, our lower bound concerns overflow probabilities, whereas our objective function is the expected maximum load in any bin. Finally, we have to use this lower bound in the setting of arbitrary random variables, despite the fact that the concrete result itself applies only to a restricted type of random variable.

Bin-packing and knapsack. In the case of the bin-packing and knapsack problems we consider primarily on-off sources. In our context, such a connection is equivalent to a weighted Bernoulli trial. Our emphasis on on-off sources is in keeping with the focus of much of the literature (see, e.g., the book [13]). With somewhat weaker performance guarantees, we can also handle the more general case of high-low sources: connections whose rates are always one of two positive values.

For the bin-packing problem with on-off items we give an algorithm that finds a solution with at most $O(\sqrt{\frac{\log p^{-1}}{\log\log p^{-1}}})B^* + O(\log p^{-1})$ bins, where B^* is the minimum possible number of bins. For the knapsack problem we provide an $O(\log p^{-1})$-approximation algorithm. We also provide constant-factor approximation algorithms

for both problems, in which case one is allowed to relax either the size of the bin or the overflow probability by an arbitrary constant $\varepsilon > 0$. Our algorithm for bin-packing can be modified to work in an *on-line* setting, in which items arrive in sequence and must be assigned to bins immediately.

Our algorithms are based on a notion of effective bandwidth, but not the standard one in the literature. In particular, the guarantee provided by the standard definition is not strong enough for the bin-packing and knapsack problems: it says that if the sum of the effective bandwidths of a set of items is bounded by 1, then the probability that the total load of these items exceeds 2 is at most p. While such a guarantee is strong enough for the load balancing problem—a load of 2 is within a constant factor of a load of 1—it is inadequate for the bin-packing and knapsack problems, which fix hard limits on the size of each bin. Stronger guarantees without exceeding the link capacity were provided by Hui [10], Kelly [11], and Elwalid and Mitra [4] using large overflow buffers. We provide such stronger guarantees without resorting to overflow buffers. In particular, for items of large peak rate (the most difficult case for the standard definition β), we make use of our new effective bandwidth β' to provide the desired performance guarantee.

1.2. Connections with stochastic scheduling. Although we have so far expressed things in the context of bursty traffic in a network, our result on load balancing also resolves a natural problem in the area of *stochastic scheduling*.

There is a large literature on scheduling with stochastic requirements; the recent book on scheduling theory by Pinedo [15] gives an overview of the important results known in this area. In a stochastic scheduling problem, the job processing times are represented by random variables; typical assumptions are that these processing times are independent and identically distributed, and that the distribution is Poisson or exponential. For some of these cases, algorithms have been developed that guarantee an asymptotically optimal schedule with high probability (e.g., Weiss [19, 20]).

We can naturally view our load balancing problem as a scheduling problem on m identical machines (the bins), with a set of n stochastic jobs (the items). Since the problem contains the NP-hard deterministic version as a special case, we cannot expect to find an optimal solution. What our load balancing result provides is a constant approximation for the minimum makespan problem on m identical machines, when the processing time of each job can have an *arbitrary* distribution.

One distinction that arises in these scheduling problems is the following: must all the jobs be loaded onto their assigned machines immediately, or can we perform an assignment adaptively, learning the processing times of earlier jobs as they finish? Our model, since it is motivated by a circuit-routing application, takes the first approach. This is also the approach taken by, e.g., Lehtonen [14], who considers the special case of exponentially distributed processing times; that work left the case of general distributions—which we handle here—as an open problem.

2. Preliminary results and examples. For much of the paper, we will be discussing random variables that are *Bernoulli trials*. We say that a random variable X is a *Bernoulli trial of type* (q, s) if X takes the value s with probability q and the value 0 with probability $1 - q$.

The load balancing, bin-packing, and knapsack problems are all NP-complete even when all items are *deterministic* (i.e., they assume a single value with probability 1). As mentioned above, the introduction of stochastic items leads to new sources of intractability.

THEOREM 2.1. *Given Bernoulli trials $X_1, \ldots, X_n$, where X_i is of type (q_i, s_i), it is #P-complete to compute* $\Pr[\sum_i X_i > 1]$.

Proof. Membership in $\#P$ is easy to verify. We prove $\#P$-hardness by a reduction from the problem of counting the number of feasible solutions to a knapsack problem. That is, given numbers $y_1, \ldots, y_n$ and a bound B, we want to know how many subsets of $\{y_1, \ldots, y_n\}$ add up to at most B. We make two modifications to this problem which do not affect its tractability:

(i) We assume that $B = 1$.

(ii) We consider the complementary problem of counting the number of subsets of $\{y_1, \ldots, y_n\}$ that sum to more than B.

Thus, given $y_1, \ldots, y_n$, we create Bernoulli trials $X_1, \ldots, X_n$ such that X_i is of type $(\frac{1}{2}, y_i)$. Let $p = \Pr[\sum_i X_i > 1]$. The theorem follows from the fact that the number of subsets of $\{y_1, \ldots, y_n\}$ that sum to more than 1 is equal to $p \cdot 2^n$. □

The use of effective bandwidth is a major component in the design of our approximation algorithms. We now give some examples to show that no "direct" use of effective bandwidth will suffice in order to obtain the approximation guarantees presented in later sections. These examples also provide intuition for some of the issues that arise in dealing with stochastic items.

First we consider the load balancing problem. A natural approximation method one might consider here is Graham's lowest-fit algorithm applied to the expected values of the items. However, this fails to achieve a constant-factor approximation. This is a consequence of the following much more general fact. Let γ be any function from random variables to the nonnegative real numbers. If $X_1, \ldots, X_n$ are random variables, and ϕ is an assignment of them to m bins, we say that ϕ is γ-optimal if it minimizes the maximum sum of the γ-values of the items in any one bin.

THEOREM 2.2. *For every function γ as above, there exist $X_1, \ldots, X_n$ and a γ-optimal assignment ϕ of $X_1, \ldots, X_n$ to m bins such that the load of ϕ is $\Omega(\log m / \log\log m)$ times the optimum load.*

Proof. For an arbitrary function γ, we consider just two kinds of distributions: a Bernoulli trial of type $(m^{-\frac{1}{2}}, 1)$ and a Bernoulli trial of type $(1, 1)$. (This latter distribution is simply a deterministic item of weight 1.) By rescaling, assume that γ takes the value 1 on Bernoulli trials of type $(1, 1)$ and the value $am^{-\frac{1}{2}}$ on Bernoulli trials of type $(m^{-\frac{1}{2}}, 1)$. We consider two cases.

Case 1. $a \leq \frac{\varepsilon \log m}{\log\log m}$ for some sufficiently small constant ε. In this case, we consider the following γ-optimal assignment: one item of type $(1, 1)$ in each of the $m - \sqrt{m}$ bins, and $\sqrt{m}/a$ items of type $(m^{-\frac{1}{2}}, 1)$ in each of the remaining $\sqrt{m}$ bins. With high probability, at least $\frac{\varepsilon \log m}{\log\log m}$ of the latter type of item will be on in the same bin, and hence the load of this assignment is $\Omega(\log m / \log\log m)$. By placing at most one item of each type in every bin, one can obtain a load of 2 for this problem.

Case 2. $a > \frac{\varepsilon \log m}{\log\log m}$. In this case, consider the following γ-optimal assignment ϕ: $C\sqrt{m}\log m$ items of type $(m^{-\frac{1}{2}}, 1)$ in each of $m - 1$ bins, for a sufficiently large constant C, and $aC \log m$ items of type $(1, 1)$ in the mth bin. Thus, the load of ϕ is at least $aC \log m$. However, with high probability, the maximum load in the first $m - 1$ bins will be $\Theta(\log m)$, and hence the assignment that evenly balances the items of both types has load $O((1 + \frac{a}{m}) \log m)$. This is better by a factor of $\Omega(\frac{am}{a+m})$. □

We now discuss a similar phenomenon in the case of bin-packing. Let us say that a packing of items into bins is *incompressible* if merging any two of its bins results in an infeasible packing. For the problem of packing deterministic items, a basic fact is that any incompressible packing is within a factor of 2 of optimal. In contrast, we can

show the existence of a set of stochastic items that can be packed in only two bins, but for which there is an *incompressible* packing using $\Omega(p^{-\frac{1}{2}})$ bins.

THEOREM 2.3. *Consider a bin-packing problem with overflow probability p. There exist sets of weighted Bernoulli trials S_1 and S_2 with the following properties.*

(i) $|S_1| = |S_2| = \Omega(p^{-\frac{1}{2}})$.

(ii) *All the items of S_1 can be packed in a single bin.*

(iii) *All the items of S_2 can be packed in a single bin.*

(iv) *One cannot pack one item from S_1 and two from S_2 together in one bin.*

Thus there is a packing of $S_1 \cup S_2$ in two bins, but the packing that uses $\Omega(p^{-\frac{1}{2}})$ and places one item from each set in each bin is incompressible.

Proof. Let p be the given overflow probability, q a real number slightly greater than p, and ε a small constant. One can verify that the above properties hold for the following two sets of weighted Bernoulli trials: S_1 consists of $\varepsilon p^{-\frac{1}{2}}$ items of type $(q, 1-\sqrt{p})$; S_2 consists of $\varepsilon p^{-\frac{1}{2}}$ items of type $(1, \sqrt{p})$. □

COROLLARY 2.4. *No algorithm which simply looks at a single "effective bandwidth" number for each item can provide an approximation ratio better than $\Omega(p^{-\frac{1}{2}})$.*

Proof. Note the behavior of any effective bandwidth function γ in the example of the above theorem. If $X \in S_1$ and $Y \in S_2$, then we have just argued that there exists a set of items whose effective bandwidths add up to $\gamma(X) + 2\gamma(Y)$ and which cannot be packed into one bin. But the entire set of items can be packed into two bins; and its total effective bandwidth is $\varepsilon p^{-\frac{1}{2}}[\gamma(X) + \gamma(Y)]$. This example also shows that the first-fit heuristic applied to a given item ordering can use a number of bins that is $\Omega(p^{-\frac{1}{2}})$ times optimal. □

The effective bandwidth we use. As discussed in the introduction, we will use both the standard definition of effective bandwidth β_p and a new *modified effective bandwidth* β'_p that turns out to be necessary in the case of bin-packing and is also used in proving our lower bounds on optimality for the load balancing problem. For a random variable X, one defines [10, 12]

$$\beta_p(X) = \frac{\log \mathrm{E}[p^{-X}]}{\log p^{-1}}. \tag{2.1}$$

For a Bernoulli trial X of type (q, s), we define its *modified effective bandwidth* by

$$\beta'_p(X) = \min\{s, sqp^{-s}\}. \tag{2.2}$$

For a set of random variables $\mathcal{R}$, we will use the notation $\beta_p(\mathcal{R}) = \sum_{X \in \mathcal{R}} \beta_p(X)$ and $\beta'_p(\mathcal{R}) = \sum_{X \in \mathcal{R}} \beta'_p(X)$.

We first give an inequality relating our modified effective bandwidth to the standard one. The proof follows from elementary calculus.

PROPOSITION 2.5. *For a Bernoulli trial X, $\beta_p(X) \le \beta'_p(X)$.*

Proof. First, we establish the following claim.

(A) *For $a \ge 1$, define $f(x) = a^x - 1$ and $g(x) = xa^x \ln a$. Then $f(x) \le g(x)$ for all $x \in [0, 1]$.*

We prove (A) by noting that

$$\lim_{x \to 0} \frac{f(x)}{g(x)} = 1,$$

and $f'(x) \le g'(x)$ for all $x \in [0, 1]$.

Now if X is of type (q, s), then we have

$$\beta_p(X) = \frac{\log\left(qp^{-s} + (1-q)\right)}{\log p^{-1}} = \frac{\log\left(1 + q(p^{-s}-1)\right)}{\log p^{-1}}.$$

To prove the proposition, it is sufficient to show that $\beta_p(X) \le s$ and $\beta_p(X) \le sqp^{-s}$. The first of these statements follows by taking logarithms base p^{-1} of the inequality $qp^{-s} + (1-q) \le p^{-s}$. To show the second, note that by Taylor's inequality

$$\beta_p(X) \le \frac{q(p^{-s}-1)}{\log p^{-1}},$$

and by fact (A)

$$\frac{q(p^{-s}-1)}{\log p^{-1}} \le qsp^{-s}. \qquad \square$$

3. Stochastic load balancing. Let $X_1, X_2, \ldots, X_n$ be mutually independent random variables taking nonnegative real values. We shall refer to them as *items*. Let $\phi : \{1, \ldots, n\} \to \{1, \ldots, m\}$ be a function assigning each item X_i to one of m *bins*. We define the *load* of the assignment ϕ, denoted $\mathcal{L}(\phi)$, to be the expected maximum load on any bin; that is, $\mathcal{L}(\phi) = \mathrm{E}[\max_i \sum_{j \in \phi^{-1}(i)} X_j]$. We are interested in designing approximation algorithms for the problem of minimizing $\mathcal{L}(\phi)$ over all possible assignments ϕ. Note that the maximum of the expectations would be easy to approximate by simply load balancing the expectations.

3.1. The algorithm for on-off items. In this subsection we present an $O(1)$-approximation algorithm for the case of weighted Bernoulli trials; we then extend this to handle arbitrary distributions in the following subsection. For a Bernoulli trial of type (q, s), we can further assume that s is a power of 2—by reducing all item sizes to the nearest power of 2 we lose only a factor of 2 in the approximation ratio.

Our load balancing algorithm is on-line. It proceeds through iterations; in each iteration it maintains a current estimate of the optimum load, which will always be correct to within a constant factor. An iteration can end in one of two ways: the input can come to an end, or the iteration can *fail*. In the latter case, the estimate of the optimum is doubled, and a new iteration begins.

For ease of notation, the algorithm rescales all modified sizes that it sees so that the estimate in the current iteration is always equal to 1. An item X_i of type (q_i, s_i) is said to be *exceptional* if $s_i > 1$, and *normal* otherwise. Throughout the algorithm, we define $p = m^{-1}$ (recall that m is the number of bins) and C to be an absolute constant. ($C = 18$ is sufficient.) One iteration proceeds as follows; suppose that item X_i has just been presented.

(1) For each bin j, let B_j denote the set of all nonexceptional items from this iteration that have been assigned to j.
(2) If X_i is normal, then we assign it to the bin j with the smallest value of $\beta_p(B_j)$. If this would cause $\beta_p(B_j)$ to exceed C, then the iteration *fails*.
(3) Suppose X_i is exceptional. If the total *expected* size of all exceptional items seen in this iteration (including X_i) exceeds 1, then the iteration *fails*. Otherwise, X_i is assigned to an arbitrary bin.

To prove that this algorithm provides a constant-factor approximation, we show that (i) if an iteration does not fail, then the load of the resulting assignment is within a constant factor of the estimate for that iteration; and (ii) if iteration fails, then

the load of any assignment must be at least a constant times the estimate for that iteration. We start with (ii).

Lower bounding the optimal solution. First we prove a lower bound on the optimal solution to the load balancing problem. This lower bound is the main new technical contribution of this part, and will be used also in analyzing the bin-packing and knapsack algorithm in the next two sections. In this subsection we state and prove the lower bound for the special case of weighted Bernoulli trials. (In section 3.2 we show how the general case follows from the special case.) Assume that $X_1, X_2, \ldots, X_n$ are independent Bernoulli trials such that X_i is of type (q_i, s_i). We will sometimes say that "item X_i is on" to refer to the event that $X_i = s_i$.

We use the following basic claim repeatedly.

CLAIM 3.1. *Let $\mathcal{E}_1, \ldots, \mathcal{E}_k$ be independent events, with $\Pr[\mathcal{E}_i] = q_i$. Let $\mathcal{E}'$ be the event that at least one of these events occurs. Let $q \leq 1$ be a number such that $\sum_i q_i \geq q$. Then $\Pr[\mathcal{E}'] \geq \frac{1}{2}q$.*

Proof. Let $\bar{q} = \frac{1}{k}\sum_i q_i$.

$$\Pr[\mathcal{E}'] = 1 - \prod_i (1 - q_i) \geq 1 - (1 - \bar{q})^{\left(\frac{1}{\bar{q}}\sum_i q_i\right)}$$

$$\geq 1 - e^{-\sum_i q_i} \geq 1 - e^{-q} \geq q - \frac{1}{2}q^2 \geq \frac{1}{2}q. \qquad \square$$

Our key technical lower bound is in the following lemma. Here $p \in [0,1]$ is a target probability (in this section we use $p = m^{-1}$).

LEMMA 3.2. *Let $X_1, \ldots, X_n$ be Bernoulli trials of types $(q_1, s_1), \ldots, (q_n, s_n)$, respectively, such that $\log^{-1} p^{-1} \leq s_i \leq 1$ for each i, and each s_i is an inverse power of 2. If $\sum_i \beta'_p(X_i) \geq 7$, then $\Pr[\sum_i X_i \geq 1] \geq p$.*

Proof. Our goal is to modify the given set of Bernoulli trials so as to obtain a new problem in which (i) the probability of the sum exceeding 1 is no greater than originally and (ii) the probability of the sum exceeding 1 is at least p.

If there is any X_i for which $\beta'_p(X_i) = s_i$, we lower q_i until $q_i = p^{s_i}$. This preserves the assumption that $\sum_i \beta'_p(X_i) \geq 7$.

Let s be an inverse power of two, and consider the set $W^{(s)}$ of items X_i for which $s_i = s$. We partition $W^{(s)}$ into sets $W_1^{(s)}, \ldots, W_{r_s}^{(s)}$ such that for all $j = 1, 2, \ldots, r_s - 1$, $2p^s \leq \sum_{i|X_i \in W_j^{(s)}} q_i \leq 3p^s$ and $\sum_{i|X_i \in W_{r_s}^{(s)}} q_i < 2p^s$. This can be done because $q_i \leq p^s$ for all $X_i \in W^{(s)}$. We define a set $V^{(s)}$ of Bernoulli trials $Y_1^{(s)}, \ldots, Y_{r_s-1}^{(s)}$, each of type (p^s, s). Intuitively, each $Y_j^{(s)}$ approximates well the behavior of $\sum_{X_i \in W_j^{(s)}} X_i$. In particular, we show that the former is stochastically dominated by the latter. We will prove the following:

(A) $\Pr[\sum_s \sum_j Y_j^{(s)} \geq 1] \leq \Pr[\sum_i X_i \geq 1]$;

(B) $\beta'_p(\cup_s V^{(s)}) \geq 1$;

(C) $\Pr[\sum_s \sum_j Y_j^{(s)} \geq 1] \geq p$.

The claim clearly follows from (A) and (C).

To prove (A), we show that $\Pr[\sum_{X_i \in W_j^{(s)}} X_i \geq s] \geq p^s = \Pr[Y_j^{(s)} \geq s]$. The expression on the left-hand side is simply the probability that any of the items in $W_j^{(s)}$ is on; by Claim 3.1, the fact that $\sum_{i|X_i \in W_j^{(s)}} q_i \geq 2p^s$, and the fact that $p^s \leq \frac{1}{2}$, this probability is at least p^s, and (A) follows.

To prove (B), notice that ${\beta'}_p(W_{r_s}^{(s)}) \le 2p^s sp^{-s} = 2s$, and for $1 \le j < r_s$, ${\beta'}_p(W_j^{(s)}) \le 3p^s sp^{-s} = 3s$. On the other hand, ${\beta'}_p(Y_j^{(s)}) = p^s sp^{-s} = s$. Thus ${\beta'}_p(V^{(s)}) \ge \frac{1}{3}({\beta'}_p(W^{(s)}) - 2s)$. Hence

$$\begin{aligned}{\beta'}_p(\cup_s V^{(s)}) &= \sum_s {\beta'}_p(V^{(s)}) \ge \sum_s \frac{{\beta'}_p(W^{(s)}) - 2s}{3} \\ &= \frac{1}{3}\sum_s {\beta'}_p(W^{(s)}) - \frac{2}{3}\sum_s s \ge 1,\end{aligned}$$

where the last inequality follows from the fact that $\sum_s {\beta'}_p(W^{(s)}) \ge 7$, and $\sum_s s \le 2$ because s only takes on the values of inverse powers of 2.

To prove (C), recall that for all j, s, ${\beta'}_p(Y_j^{(s)}) = p^s sp^{-s} = s$. Now, let V denote a subset of $\cup_s V^{(s)}$ consisting of items whose sizes sum to 1. That such a set exists follows from (B) and the fact that all sizes are inverse powers of 2. Let $\{Y_1', \ldots, Y_\ell'\}$ denote the items in V, and let $s_1', \ldots, s_\ell'$ denote their sizes, respectively. Note that the probability that Y_i' is on is equal to $p^{s_i'}$.

The probability of the event $\sum_s \sum_j Y_j^{(s)} \ge 1$ is at least as large as the probability that all items in V are on. But this latter probability is equal to $\prod_{i=1}^{\ell} p^{s_i'} = p$. ∎

The lower bound for exceptional items follows by an argument using Claim 3.1.

LEMMA 3.3. *Let $X_1, \ldots, X_n$ be such that $L \le s_1 \le \cdots \le s_n$ and $\sum_i q_i s_i \ge L$. Then for all ϕ, we have $\mathcal{L}(\phi) \ge \frac{1}{2}L$.*

Proof. Without loss of generality, we may assume $\sum_i q_i s_i = L$. Let $q_i' = \sum_{j \ge i} q_j$. Let $\mathcal{E}_i$ denote the event that at least one item among $\{X_j\}_{j \ge i}$ is on, and let $q_i'' = \Pr[\mathcal{E}_i]$. Note that because $\sum_i q_i s_i = L$ and $s_i \ge L$ for all i, we have $\sum_i q_i \le 1$ and hence $q_i' \le 1$ for all i. Thus, by Claim 3.1, $q_i'' \ge \frac{1}{2}q_i'$. Write $s_0 = 0$ and $q_{n+1}' = 0$.

Observe that $\sum_i q_i s_i = \sum_i q_i'(s_i - s_{i-1})$, because each s_i is counted with a multiplier of q_i on the right-hand side.

Since $\Pr[X_i \text{ is on } \textit{and} \text{ not } \mathcal{E}_{i+1}] = q_i'' - q_{i+1}''$, we have

$$\mathrm{E}[\max\{X_1, \ldots, X_n\}] \ge \sum_i s_i(q_i'' - q_{i+1}'') = \sum_i q_i''(s_i - s_{i-1}).$$

Thus for any assignment ϕ we have

$$\begin{aligned}\mathcal{L}(\phi) &\ge \mathrm{E}[\max\{X_1, \ldots, X_n\}] \ge \sum_i q_i''(s_i - s_{i-1}) \\ &\ge \frac{1}{2}\sum_i q_i'(s_i - s_{i-1}) = \frac{1}{2}\sum_i q_i s_i = \frac{1}{2}L.\end{aligned}$$ ∎

Our main lower bound for the load balancing problem is the following lemma.

LEMMA 3.4. *Suppose that for all i, s_i is an inverse nonnegative integral power of 2 (so $s_i \le 1$). Further suppose that $\sum_i {\beta'}_{m^{-1}}(X_i) \ge 17m$. Then, for all ϕ, $\mathcal{L}(\phi) = \Omega(1)$.*

Proof. Let ϕ be an arbitrary assignment of the items to bins. Let $B_1, \ldots, B_m$ denote the sets of items assigned to bins $1, \ldots, m$, respectively. Apply the following construction: as long as some set B_i' contains a subset$S$ with ${\beta'}_{m^{-1}}(S) \ge 8$, we

put aside a minimal subset S with this property. Note that ${\beta'}_{m^{-1}}(S) \leq 9$ as the bandwidth of a single item of size at most 1 never exceeds 1. When we can no longer find such a subset, then the set of remaining items R has ${\beta'}_{m^{-1}}(R) \leq 8m$. Thus, this construction produces at least m subsets, such that each is assigned to a single bin by ϕ. We denote the first m of these subsets by $W_1, \ldots, W_m$.

Call a Bernoulli trial X of type (q, s) *small* if $s < 1/\log p^{-1}$. Using the fact that small items have $p^{-s} \leq 2$, we can see that the effective bandwidth ${\beta'}_p(X)$ of a small item is at most twice its expectation $\mathrm{E}[X] = qs$. Call a set W_i *dense* if the set of small items $S_i \subseteq W_i$ has ${\beta'}_{m^{-1}}(S_i) \geq 1$. If there exists a dense set W_i, then the expected size of W_i is at least $\frac{1}{2}$. Since $\mathcal{L}(\phi)$ is at least as large as the expected size of W_i, $\mathcal{L}(\phi) \geq \frac{1}{2}$ and the lemma follows.

Thus, we consider the case in which no W_i is dense. Let $W_i' \subseteq W_i$ denote the set of items in W_i which are not small. Since W_i is not dense, ${\beta'}_{m^{-1}}(W_i') \geq 7$. By Lemma 3.2, the probability that size of W_i' exceeds 1 is at least m^{-1}. Hence the probability that *some* W_i' exceeds 1 is at least $1 - (1 - m^{-1})^m \geq 1 - e^{-1}$. Since $\mathcal{L}(\phi) \geq \mathrm{E}[\max\{W_1', \ldots, W_m'\}]$, the lemma follows. □

Recall that the algorithm maintains a current estimate. The iteration fails if the total effective bandwidth of the small and normal items in a bin would exceed a constant C (we use $C = 18$) or if the total expected size of all exceptional items seen in this iteration exceeds 1.

THEOREM 3.5. *Let W denote the set of items presented to the algorithm in an iteration that fails. For any assignment ϕ of W to a set of m bins we have $\mathcal{L}(\phi) = \Omega(1)$, where 1 is the estimate for the iteration.*

Proof. Let ϕ be an arbitrary assignment of items in W to bins. An iteration can fail in one of two ways: either because the expected total size of exceptional items exceeds 1, or because the assignment of the new item to any bin j would cause $\beta_p(B_j)$ to exceed C.

In the first case, Lemma 3.3 implies that $\mathcal{L}(\phi) \geq \frac{1}{2}$. Concerning the second case, consider the moment at which the iteration fails. We have $\sum_j \beta_p(B_j) \geq m(C - 1)$ (because the new item's size, and therefore its effective bandwidth, cannot exceed 1). Recalling that $C \geq 18$, Lemma 3.4 asserts that $\mathcal{L}(\phi) = \Omega(1)$. □

Upper bounding the solution obtained. The following proposition is essentially due to Hui [10], who stated it with $a = 2$ and $b = 1$. We give a short proof for the sake of completeness.

PROPOSITION 3.6 (see [10]). *Let $X_1, \ldots, X_n$ be independent random variables, and $X = \sum_i X_i$. Let $a > b$. If $\sum_i \beta_p(X_i) \leq b$, then $\Pr[X \geq a] \leq p^{a-b}$.*

Proof. First, if $\sum_i \beta_p(X_i) \leq b$, then $\sum_i \log \mathrm{E}[p^{-X_i}] \leq \log p^{-b}$ and hence $\prod_i \mathrm{E}[p^{-X_i}] \leq p^{-b}$.

Thus we have $\Pr[X \geq a] = \Pr[p^{-X} \geq p^{-a}] \leq p^a \mathrm{E}[p^{-X}] = p^a \prod_i \mathrm{E}[p^{-X_i}] \leq p^{a-b}$, where the first inequality follows from Markov's inequality, the equation from the independence of the X_i, and the last inequality from inequality above. □

LEMMA 3.7. *Consider the assignment produced by any iteration of the algorithm. The load of this assignment is $O(1)$. (Recall that sizes are scaled so that 1 is the estimate for that iteration.)*

Proof. The expected size of the sum of exceptional items placed in this iteration is at most 1, so they only add at most 1 to the expected maximum load.

Let $S_j = \sum_{X_i \in B_j} X_i$. Let $x \geq 0$. As $\beta_p(B_j) \leq C$, $\Pr[S_j > x + C] \leq m^{-x}$ by Proposition 3.6. Let $S^* = \max\{S_1, \ldots, S_m\}$. We have$\Pr[S^* \geq y] \leq \sum_j \Pr[S_j \geq y]$.

Hence

$$\begin{aligned}\mathrm{E}[S^*] &= \int_0^\infty \Pr[S^* \geq x]dx \leq C + 1 + \int_{C+1}^\infty \Pr[S^* \geq x]dx \\ &= C + 1 + \int_1^\infty \Pr[S^* \geq x + C]dx \\ &\leq C + 1 + \int_1^\infty m \cdot m^{-x}dx \\ &= C + 1 + m\frac{1}{m \ln m} = C + O(1),\end{aligned}$$

from which the lemma follows. □

Since the estimates increase geometrically, a consequence of Lemma 3.7 is the following theorem.

THEOREM 3.8. *Let ϕ_A be the assignment produced by the algorithm. Then $\mathcal{L}(\phi_A) = O(1)$, where item sizes are scaled so that 1 is the estimate for the final iteration.*

Combining Theorems 3.8 and 3.5, we get our main result.

THEOREM 3.9. *The algorithm provides a constant-factor approximation to the minimum load.*

3.2. Extension to arbitrary distributions. We may assume that the only values taken on by our random variables are powers of 2. If not, other values are rounded down to a power of 2. As in the previous section, this increases our approximation guarantee by a factor of 2 at most. Call a random variable that only takes values that are powers of 2 *geometric*. By the following claim we can reduce the problem for geometric items to the problem for Bernoulli trial items, which we have already solved.

LEMMA 3.10. *Let X be a geometric random variable. Then there exists a set of independent Bernoulli trials $Y_1, \ldots, Y_k$, with $Y = \sum_i Y_i$, such that $\Pr[X = s] = \Pr[s \leq Y < 2s]$.*

Proof. Suppose that X takes the value s_i with probability q_i for $i = 1, \ldots, k$. Suppose that $s_1 > s_2 > \cdots > s_k$. We define Y_i to be of type (q_i', s_i), where

$$q_i' = \frac{q_i}{(1 - q_1 - \cdots - q_{i-1})}.$$

Notice that the events $X = s_i$, $i = 1, \ldots, k$, are mutually exclusive, and therefore q_i' is simply $\Pr[X = s_i \mid X \leq s_i]$. The set $\{q_i'\}$ is the solution to

$$\begin{aligned}q_1 &= q_1', \\ q_2 &= (1 - q_1')q_2', \\ q_3 &= (1 - q_1')(1 - q_2')q_3', \\ &\vdots \\ q_k &= \left(\prod_{i=1}^{k-1}(1 - q_i')\right) q_k',\end{aligned}$$

and hence

$$\Pr[\max_j Y_j = s_i] = q_i' \prod_{j=1}^{i-1}(1 - q_j') = q_i = \Pr[X = s_i].$$

As $s_i > \sum_{j>i} s_j$, the claim follows. □

The algorithm is essentially the same as before. It uses the standard definition of effective bandwidth (Equation (2.1)), which applies to any distribution. The only change arises from the fact that we must define what we mean by "exceptional" in this case. Each item X_i is now divided into an *exceptional part* $X_i \cdot \mathbf{1}_{\{X_i>1\}}$ and a *nonexceptional part* $X_i \cdot \mathbf{1}_{\{X_i\leq 1\}}$. When the expected total value of all exceptional parts exceeds 1, the iteration fails; before this, exceptional parts are (necessarily) just packed together with their nonexceptional parts.

THEOREM 3.11. *The algorithm provides a constant-factor approximation to the minimum load.*

Proof. Recall that in the case of Bernoulli trials exceptional items could be packed in any bin. The upper bound argument follows as before, using Proposition 3.6 for the nonexceptional parts of the items.

The lower bound argument requires the approximation of each item by a sum of Bernoulli trials using Lemma 3.10. We replace each item X_i of a geometric random variable by the corresponding independent Bernoulli trials and apply the lower bound of the previous subsection to the resulting set of Bernoulli trials. □

4. Bin packing with stochastic on-off items. In this section we consider the *bin packing problem* with independent weighted Bernoulli trials, which we will refer to as "items." In addition we are given an allowed probability of overflow p. The problem is to pack the items into as few bins of size 1 each as possible, so that in each bin the probability that the total size of the items in the bin exceeds 1 is at most p. We assume throughout that $p \leq \frac{1}{8}$; this is consistent with routing applications, where p is much smaller than this [4].

We develop approximation algorithms parameterized by a number ε, $0 < \varepsilon < \frac{1}{2}$. Our results show that a solution whose value is within a factor of $O(\varepsilon^{-1})$ to optimal can be obtained if we relax either the bin size or the overflow probability. That is, we compare the performance of our algorithm to the optimum for a slightly smaller bin size or overflow probability. Using these results we then give an approximation algorithm without relaxing either the bin size or the overflow probability. Our algorithms will be on-line, as before.

The basic outline of the method is as follows. As in the load balancing algorithm, we will classify items according to their sizes. For the case with relaxed sizes and/or probabilities, an item will be *small* if $s_i \leq 1/\log_2 p^{-1}$, *large* if $s_i \geq \frac{1}{2}\varepsilon$ for the parameter ε, and *normal* otherwise. We pack using the expectation for small items, using the effective bandwidth $\beta_p(X)$ for normal items, and we develop techniques for packing large items based on our version of the effective bandwidth $\beta'_p(X)$. It can in fact be shown that the standard definition of effective bandwidth is not adequate for obtaining a strong enough approximation ratio.

For a large item of type (q_i, s_i), we effectively discretize its size, and work with its *effective size* $\bar{s}_i$; this is the reciprocal of the minimum number of copies of weight s_i that will overflow a bin of size 1: $\bar{s}_i^{-1} = \min\{j : js_i > 1\}$. Notice that $\bar{s}_i < s_i$ for all i.

An algorithm with relaxed bin size and probability. We start by describing a simpler version of the algorithm in which we relax both the bin size and the overflow probability. Each bin will contain items only of the same type (small, normal, or large). Each item is assigned a *weight*, according to which it is packed. Bins of each type can be packed according to any on-line bin-packing heuristic, applied to the weights; to be concrete, we will assume that the first-fit heuristic is being used.

Small items are given a weight equal to their expectation. A bin with small items will be packed so that its total weight does not exceed $\frac{1}{6}$. Each normal item X is assigned a weight of $\beta_p(X)$. A bin of normal items will be packed so that its total weight does not exceed ε.

The set of large items can have at most $\lceil 2\varepsilon^{-1} \rceil$ different effective sizes. They are classified into groups by the following two criteria.

(i) Each bin will only contain items of the same effective size.

(ii) We say that a large item X_i of type (q_i, s_i) and effective size $\bar{s}$ has *large probability* if $q_i \geq p^{\bar{s}}$ and *normal probability* otherwise. No bin will contain items of both large and normal probabilities.

We pack large probability items in bins so that fewer than $\frac{1}{\bar{s}}$ are in any bin. We pack normal probability items so that the sum of the probabilities of items in a bin does not exceed $p^{\bar{s}}/\bar{s}e$ where $e \approx 2.7..$ is the base of the natural logarithm. We now argue that the algorithm yields a feasible packing in bins of size $1 + \varepsilon$.

First we consider large items. If a bin contains items of effective size $\bar{s} = \frac{1}{k}$, then it will overflow if and only if at least k items are on. This implies that bins with large probability items do not overflow even if all items are on. Large items with normal probability are handled by the following lemma, which involves an analysis of our modified effective bandwidth.

LEMMA 4.1. *Let $X_1, \ldots, X_n$ be independent Bernoulli trials of types $\{(q_i, s_i)\}$, and assume that the effective size $\bar{s}_i = \bar{s}$ and $q_i \leq p^{\bar{s}}$ for all i. Let $X = \sum_i X_i$, and assume that $\sum_i q_i \leq p^{\bar{s}}/\bar{s}e$. Then $\Pr[X \geq 1] \leq p$.*

Proof. We get overflow in a bin if and only if at least k items are on, where $k = \frac{1}{\bar{s}}$. Let $\mathcal{I}$ denote the set of all items. For a set of items $S \subseteq \mathcal{I}$ of size k, the probability that all items in S are on is $\prod_{i \in S} q_i$. Thus the probability of overflow is at most

$$\sum_{S \subseteq I, |S|=k} \prod_{i \in S} q_i. \tag{4.1}$$

We claim that this formula is maximized for a given sum of probabilities $\sum_i q_i$ if all probabilities q_i are all the same. To see this, suppose that we have two items X_i, X_j with different probabilities, and consider modified items with probabilities $q_i' = q_j' = \frac{1}{2}(q_i + q_j)$. We now observe that the sum of probabilities has remained the same, but the probability of overflow is larger: the terms of (4.1) that contain 0 or 1 of the values q_i, q_j contribute in total the same as before, and terms containing both are each increased.

Assume now that all items have the same probability q. The sum of the probabilities of items is at most $p^{\bar{s}}/\bar{s}e$; hence, the number of these items is at most $p^{\bar{s}}/\bar{s}qe$. Now the probability that k items are on is bounded by

$$\binom{p^{\bar{s}}/q\bar{s}e}{1/\bar{s}} q^{\frac{1}{\bar{s}}} \leq \left(\frac{p^{\bar{s}}}{q}\right)^{\frac{1}{\bar{s}}} q^{\frac{1}{\bar{s}}} = p;$$

the inequality follows from the estimate $\binom{n}{k} \leq (\frac{en}{k})^k$. □

The feasibility for small items follows easily from Chernoff bounds.

LEMMA 4.2. *If $X_1, \ldots, X_k$ be independent Bernoulli trials of types $(q_1, s_1), \ldots, (q_k, s_k)$, such that $s_i \leq \frac{1}{\log_2 p^{-1}}$, and $\sum_i \mathrm{E}[X_i] \leq \frac{1}{6}$, then $\Pr[\sum_i X_i \geq 1] < p$.*

Proof. We use Chernoff bounds to bound the probability that the sum exceeds 1. With $\mu = \frac{1}{6} \log p^{-1}$, we have $\Pr[\sum_i X_i > 1] < (e^{5/6}/6)^{6\mu} < 2^{-6\mu} = p$. □

For the normal items, we apply Proposition 3.6 with $a = 1 + \varepsilon$ and $b = \varepsilon$.

We state this special case here for easy reference.

LEMMA 4.3. *Let $X_1, \ldots, X_n$ be independent random variables, and $X = \sum_i X_i$. Let $\varepsilon > 0$. If $\sum_i \beta_p(X_i) \le \varepsilon$, then $\Pr[X \ge 1 + \varepsilon] \le p$.*

THEOREM 4.4. *The on-line algorithm finds a packing of items in bins with the property that for each bin, the probability that the total size of the items in that bin exceeds $1 + \varepsilon$ is at most p.*

Note that large and small items are also feasible with bin size 1; it is only the normal items that require the relaxed bin size.

To prove the approximation ratio, we need to lower-bound the optimum. For small items, Chernoff bounds are sufficient; for normal items and large items of a given effective size we make use of a more careful analogue of Lemma 3.2.

Lemmas 4.7 and 4.8 will show that on large items of a given effective size the number of bins used by our algorithm is at most a constant factor away from the minimum possible. Since there are only $\lceil 2\varepsilon^{-1} \rceil$ different large effective sizes, this implies a bound of $O(\varepsilon^{-1})$ on large items. Lemma 4.6 shows that normal items with large total effective size (more than $5(1 + 2\varepsilon)$) have overflow probability more than $p^{1+3\varepsilon}$. This will imply that the number of bins used for normal items is at most an $O(\varepsilon^{-1})$ factor away from optimal. Finally, small items are again handled directly with Chernoff bounds.

LEMMA 4.5. *Let $p < \frac{1}{2}$ and $X_1, \ldots, X_k$ be independent Bernoulli trials of types (q_1, s_1), ..., (q_k, s_k), such that $s_i \le \log_2 p^{-1}$. If $\sum_i \mathrm{E}[X_i] \ge 4$, then $\Pr[\sum_i X_i > 1] > p$.*

Proof. We use Chernoff bounds to bound the probability that the sum exceeds 1. With $\mu = 4 \log p^{-1}$, we have

$$\Pr\left[\sum_i X_i \le 1\right] \le e^{-\frac{1}{2}\left(\frac{3}{4}\right)^2 \mu} < p,$$

and hence $\Pr[\sum_i X_i > 1] > 1 - p \ge p$. □

Next we consider normal items. In the load-balancing algorithm we proved a lower bound for effective bandwidth in Lemma 3.2; here we require a stronger version of this lemma. For later use we state the lemma with a parameter δ. Here we will use it with $\delta = 1$.

LEMMA 4.6. *Let $X_1, \ldots, X_k$ be independent Bernoulli trials of types $(q_1, s_1), \ldots, (q_k, s_k)$, such that $s_i \ge \frac{1}{\log_2 p^{-1}}$, and $\sum_i \beta_p(X_i) \ge (3\delta + 2)(1 + 2\varepsilon)$; then*

$$\Pr\left[\sum_i X_i > \delta\right] > p^{\delta(1+2\varepsilon)+\varepsilon}.$$

Proof. Recall that $\beta_p(X) \le {\beta'}_p(X)$ for all Bernoulli trials; hence we have that $\sum_i {\beta'}_p(X_i) \ge (3\delta + 2)(1 + 2\varepsilon)$. Further, we will round up the size of each Bernoulli trial X_i to an integer power of $1 + \varepsilon$. Let X_i' denote the resulting rounded item, and let (q_i, s_i') denote its type. Rounding up cannot decrease the effective bandwidth, so we have that $\sum_i {\beta'}_p(X_i') \ge (3\delta + 2)(1 + 2\varepsilon)$.

Next we prove an analogue of Lemma 3.2 for the rounded items. We claim that with probability more than $p^{\delta(1+2\varepsilon)+\varepsilon}$, the total size of the rounded items exceeds $\delta(1 + \varepsilon)$. Notice that this implies that the total size of the original items exceeds δ with probability more than $p^{\delta(1+2\varepsilon)+\varepsilon}$. The proof is analogous to the proof of Lemma 3.2.

We may assume without loss of generality that $q_i \le p^{s_i}$ for all i. Now we have that

$$\sum_i q_i s_i p^{-s_i} \ge (3\delta + 2)(1 + 2\varepsilon). \tag{4.2}$$

We define the sets of items $W^{(s)}$ for each size s; partition $W^{(s)}$ into sets $W_1^{(s)}, \ldots, W_{r_s}^{(s)}$ such that $2p^s \le \sum_{X_i \in W_j^{(s)}} q_i < 3p^s$ for $j = 1, \ldots, r_s - 1$; and define the set $V^{(s)}$ of Bernoulli trials $Y_1^{(s)}, \ldots, Y_{r_s-1}^{(s)}$, each of type (p^s, s), as in the proof of Lemma 3.2.

Next we want to argue that (i) the probability of the sum $\sum_{j,s} Y_j^{(s)}$ exceeding $\delta(1+\varepsilon)$ is no greater than the probability of $\sum_i X_i'$ exceeding $\delta(1+\varepsilon)$, and (ii) the probability of the sum $\sum_{j,s} Y_j^{(s)}$ exceeding $\delta(1+\varepsilon)$ is at least $p^{\delta(1+2\varepsilon)+\varepsilon}$.

To argue part (i) we show as before, using Claim 3.1, that $\Pr[\sum_{X_i' \in W_j^{(s)}} X_i' \ge s] \ge p^s = \Pr[Y_j^{(s)} \ge s]$. The fact that $p^s \le 1/2$ follows from the assumption that $s_i \ge 1/\log_2 p^{-1}$ for all i.

To show part (ii) we claim, using the notation from the proof of Lemma 3.2, that $\beta'_p(\cup_s V^{(s)}) > \delta(1+\varepsilon)$. To prove this, we note that as before we have $\beta'_p(V^{(s)}) > \frac{\beta'_p(W^{(s)}) - 2s}{3}$. Hence

$$\begin{aligned}\beta'_p(\cup_s V^{(s)}) &= \sum_s \beta'_p(V^{(s)}) > \sum_s \frac{\beta'_p(W^{(s)}) - 2s}{3} = \frac{1}{3}\left(\sum_s \beta'_p(W^{(s)}) - 2\sum_s s\right) \\ &= \frac{1}{3}\left(\sum_s \beta'_p(W^{(s)}) - 2(1+\varepsilon)\right) \ge \delta(1+2\varepsilon),\end{aligned}$$

since $\sum_s \beta'_p(W^{(s)}) \ge (3\delta+2)(1+2\varepsilon)$ by (4.2), and $\sum_s s \le (1+\varepsilon)$ since s only takes on values that are integer powers of $(1+\varepsilon)$ and at most ε.

We complete the proof of the lemma by showing $\Pr[\sum_s \sum_j Y_j^{(s)} > \delta(1+\varepsilon)] \ge p^{\delta(1+2\varepsilon)+\varepsilon}$. Note that for all j, s, $\beta'_p(Y_j^{(s)}) = p^s s p^{-s} = s$. Now, let V denote a subset of $\cup_s V^{(s)}$ consisting of items whose sizes sum to a number in $(\delta(1+2\varepsilon), \delta(1+2\varepsilon)+\varepsilon)$; such a set can be chosen as we have shown above that the sum of all sizes in $\cup_s V^{(s)}$ is at least $\delta(1+2\varepsilon)$, and all sizes are at most ε. Let $\{Y_1', \ldots, Y_\ell'\}$ denote the items in V, with $s_1', \ldots, s_\ell'$ denoting their sizes. Note that the probability that Y_i' is on is equal to $p^{s_i'}$.

The probability of the event

$$\sum_s \sum_j Y_j^{(s)} \ge \delta(1+2\varepsilon) > \delta(1+\varepsilon)$$

is at least as large as the probability that all items in V are on. But this latter probability is equal to

$$\prod_{i=1}^{\ell} p^{s_i'} = p^{\sum_{i=1}^{\ell} s_i'} > p^{\delta(1+2\varepsilon)+\varepsilon}. \qquad \square$$

Next we consider a group of large items of effective size s. The packing created by the algorithm is clearly optimal for items of large probability.

LEMMA 4.7. *If $\frac{1}{s}$ large probability items of effective size s are in the same bin, then the probability of overflow is more than p.*

Proof. Let $X_1, \ldots, X_s$ denote $\frac{1}{s}$ large probability items of effective size s. Note that if all $\frac{1}{s}$ items are on, then the total size exceeds the bin size 1. The probability of item i is $q_i > p^s$ for all i. The probability that all s items are on is therefore at least $\prod_i q_i > (p^s)^{\frac{1}{s}} = p$. □

Finally, consider large items of a given effective size and normal probability.

LEMMA 4.8. *Let $X_1, \ldots, X_k$ be independent Bernoulli trials of effective size s and probability $q_1, \ldots, q_k$, such that $s \geq \frac{1}{\log_2 p^{-1}}$; $q_i \leq p^s$ for all i, and $\sum_i q_i \geq 3p^s/s$. Then $\Pr[\sum_i X_i > 1] > p$.*

Proof. We need to argue that the probability that at least $\frac{1}{s}$ of the items are on exceeds p. We partition the set of items into sets $W_1, \ldots, W_{r+1}$ such that

$$2p^s < \sum_{X_i \in W_j} q_i \leq 3p^s$$

for $j = 1, \ldots, r$, and $r \geq \frac{1}{s}$. This is possible as $\sum_i q_i \geq 3p^s/s$ and $q_i \leq p^s$ for each i.

By the assumption that $p^s \leq \frac{1}{2}$, Claim 3.1 implies that in any set W_j for $j = 1, \ldots, r$ the probability that at least one of the items is on the set is more than p^s. Now the probability that at least one item is on in each of the first $\frac{1}{s}$ groups is more than $(p^s)^{\frac{1}{s}} = p$. This implies the lemma. □

Now we are ready to prove the general bound.

THEOREM 4.9. *For a parameter $\varepsilon \geq \frac{1}{\log_2 p^{-1}}$, the above on-line algorithm finds a packing of items in bins of size $1 + \varepsilon$ such that the number of bins used is at most $O(\varepsilon^{-1})$ times the minimum possible number of bins in any packing with bin size 1 and overflow probability at most $p^{1+3\varepsilon}$.*

Proof. We show that the number of bins used by our algorithm for small, normal, and large items is within $O(\varepsilon^{-1})$ of optimal.

First, suppose we use B bins for small items. Each bin is packed up to an expected value of at least $\frac{1}{6} - \frac{1}{\log p^{-1}}$ since packing an extra small item in the bin would exceed the expected value of $\frac{1}{6}$. It follows that the total expected value of all small items is at least $\frac{B(\log p^{-1} - 6)}{6 \log p^{-1}}$. Hence, if fewer than $\frac{B(\log p^{-1} - 6)}{24 \log p^{-1}}$ bins are used, some bin will overflow with probability exceeding p, by Lemma 4.5.

Next, suppose we use B bins for normal items. Each bin is packed up to a β_p-value of at least $\frac{1}{2}\varepsilon$ since adding a new normal item to a bin would exceed the total β_p value of ε, and each normal item has β_p value at most $\frac{1}{2}\varepsilon$. Therefore, the total β_p-value of normal items is at least $\frac{1}{2}\varepsilon B$. Hence, if fewer than $\frac{\varepsilon B}{10(1+2\varepsilon)}$ bins of size 1 are used for normal items, then Lemma 4.6 implies that some bin will overflow with probability exceeding p.

Finally, we consider large items of a given effective size. We show that we are within a constant factor of optimal on this set of items, where the constant does not depend on ε; thus, since there are only $\lceil 2\varepsilon^{-1} \rceil$ different effective sizes, our packing of large items will be within $O(\varepsilon^{-1})$ of optimal. First, Lemma 4.7 implies that for each effective size, the number of bins used for large items of large probability is optimal. Now suppose that we use B bins for large items of normal probability and a given effective size s. Then the total probability of this set of items is at least $Bp^s/2se$. Therefore, if fewer than $B/6e$ bins were used for this set of items, the items in at least one bin would have total probability more than $3p^s/s$, and by Lemma 4.8 the probability of overflow would exceed p. □

Algorithms with either relaxed bin size or probability. In fact, we can obtain the same approximation ratios (up to a constant factor) by only relaxing *either* the bin size or the overflow probability, but not both. Since the relaxed guarantees were only needed for normal items, the idea is to slightly "inflate" or "deflate" the size of the normal items that we present to the above algorithm and argue that we still do not lose too much in comparison to the optimum.

THEOREM 4.10. *There is a constant c such that for any parameter $\varepsilon \geq \frac{c}{\log p^{-1}}$ the following holds. There is an on-line polynomial time algorithm that finds a packing of items in bins of size* 1 *with overflow probability p, such that the number of bins used is at most $O(\varepsilon^{-1})$ times the minimum number of bins in any packing with bin sizes* 1 *and overflow probability at most $p^{1+\varepsilon}$.*

Proof. As just noted, the analysis for large and small items follows as before. The trouble with applying the previous analysis for normal items, of course, is that the packing created by the algorithm above might overflow bins of size exactly 1.

Here, we continue to use the effective bandwidth to pack normal items; however, for each normal item of type (q_i, s_i), we present the algorithm with an *inflated item* of type $(q_i, s_i(1+\varepsilon))$. We also set the threshold for large items at $\frac{1}{2}\varepsilon(1+\varepsilon)$, so that inflated items remain normal. Lemma 4.3 implies that the probability that the total sizes of the inflated items exceeds $1+\varepsilon$ is at most p; hence, the probability that the total size of the original items exceed 1 is at most p. For the lower bound on the optimum, we apply Lemma 4.6 to the inflated items, with $\delta = 1+\varepsilon$ to conclude that if the total effective bandwidth of the inflated items is sufficiently large then the probability that these items overflow a bin of size $1+\varepsilon$ is at least $p^{(1+\varepsilon)(1+2\varepsilon)+\varepsilon} \geq p^{1+5\varepsilon}$. Finally, we observe that a set of inflated items overflows a bin of size $1+\varepsilon$ if and only if the original items overflow a bin of size 1.

To get the bound claimed in the theorem, we must run the above algorithm with a parameter $\varepsilon' = \frac{1}{5}\varepsilon$. □

THEOREM 4.11. *For a parameter $\varepsilon \geq \frac{1}{\log_2 p^{-1}}$, there is a polynomial time algorithm that finds a packing of items in bins of size $1+\varepsilon$ with overflow probability p such that the number of bins used is at most $O(\varepsilon^{-1})$ times the minimum possible number of bins in any packing with bin sizes* 1 *and overflow probability at most p.*

Proof. We use an algorithm similar to that of Theorem 4.10, except that now we decrease the size of each normal item by a factor of $1-\varepsilon$. Lemma 4.3 implies that the decreased sized items do not overflow a bin of size $1+\varepsilon$, and hence the original items do not overflow a bin of size $\frac{1+\varepsilon}{1-\varepsilon} \leq 1+4\varepsilon$.

To obtain the lower bound, we want to prove that if the total effective bandwidth of the decreased sized items is sufficiently large, then the probability that the total size of these items exceeds $1-\varepsilon$ is at least p. This will imply that the total size of the original items is at least 1 with probability at least p. The proof follows from Lemma 4.6 applied to the decreased item sizes and $\delta = \frac{1-\varepsilon}{1+\varepsilon}$.

To get the bound claimed in the theorem, we must run the above algorithm with a parameter $\varepsilon' = \frac{1}{2}\varepsilon$. □

An algorithm without relaxing bin size or probability. In this section we use the results above to obtain an approximation algorithm without relaxing either the bin size or the capacity. In fact, our algorithm will simply be the on-line algorithm from the previous section, with $\varepsilon = \frac{1}{\log_2 p^{-1}}$. Thus, there will be no items classified as *normal*—only small and large. One can give a weak analysis of this algorithm as follows: since the relaxed probabilities and sizes were only required for normal items, this algorithm produces a packing that is with $O(\varepsilon^{-1}) = O(\log p^{-1})$ times the

optimum with bin size 1 and overflow probability p.

Our goal in this section is to give a more involved analysis of the same algorithm, showing that its performance is actually much better than this: it produces a packing with $O(\sqrt{\frac{\log p^{-1}}{\log\log p^{-1}}})B^* + O(\log p^{-1})$ bins, where B^* is the optimum number of bins required (with size 1 and overflow probability p).

The main step of the analysis is the following extension of Lemma 3.2.

LEMMA 4.12. *Let* $\varepsilon = \frac{1}{6}\sqrt{\frac{\log\log p^{-1}}{\log p^{-1}}}$. *If* $X_1, \ldots, X_k$ *are independent Bernoulli trials of types* $(q_1, s_1), \ldots, (q_k, s_k)$, $\varepsilon \geq s_i \geq \frac{1}{\log_2 p^{-1}}$, *and* $\sum_i {\beta'}_p(X_i) \geq 7\varepsilon^{-1}$, *then* $\Pr[\sum_i X_i > 1] > p$.

The proof relies heavily on our modified effective bandwidth, with a grouping scheme as in the proof of Lemma 3.2. However, we cannot afford to analyze the groups in each effective size separately; thus we require a combinatorial argument which analyzes the antichain of minimal collections of groups that would cause the bin to overflow.

Before proving this lemma, we require a simple combinatorial fact. Let S be a set of size n, let $k \leq \ell \leq n$, and let $\mathcal{F}_{k\ell}$ denote the collection of all subsets of S whose size is at least k and at most ℓ. We say that $\mathcal{I} \subseteq \mathcal{F}_{k\ell}$ is an *antichain* if no set in $\mathcal{I}$ contains any other set, and a *maximal antichain* if it is maximal with this property.

CLAIM 4.13. *Assume* $k \leq \ell \leq \frac{n}{2}$. *Then the number of elements in a maximal antichain* $\mathcal{I} \subset \mathcal{F}_{k\ell}$ *is at least* $\binom{n}{k}/\binom{\ell}{k}$.

Proof. Consider a maximal antichain $\mathcal{A}$. Each k-element set S must be contained in one of the sets T of the maximal antichain $\mathcal{A}$. An antichain element T can contain up to $\binom{\ell}{k}$ k-element subsets, and there are altogether $\binom{n}{k}$ k-element sets. This implies the claim. □

Proof of Lemma 4.12. The proof starts out analogous to the proof of Lemma 4.6. We round item sizes up to a power of $(1+\varepsilon)$ and assume without loss of generality that all items have normal probability, i.e., $q_i \leq p^{s_i}$. Then we have that $\sum_i q_i s_i p^{-s_i} \geq 7\varepsilon^{-1}$.

We partition the set of items into sets $W_j^{(s)}$ for each size s such that $2p^s \leq \sum_{X_i \in W_{(s)}^j} q_i \leq 3p^s$ for $j = 1, \ldots, r_s$; and define the set $V^{(s)}$ of Bernoulli trials $Y_1^{(s)}, \ldots, Y_{r_s-1}^{(s)}$, each of type (p^s, s), as in the proof of Lemma 3.2. As before we have that ${\beta'}_p(V^{(s)}) \geq \frac{{\beta'}_p(W^{(s)}) - 2s}{3}$. Further $\sum_s s \leq 1+\varepsilon$, hence we get that ${\beta'}_p(\cup_s V^{(s)}) \geq 2\varepsilon^{-1}$. Note also that ${\beta'}_p(\cup_s V^{(s)}) \leq \frac{1}{2}{\beta'}_p(\cup_s W^{(s)}) \leq \frac{7}{2}\varepsilon^{-1}$.

Next we form groups $G_1, \ldots, G_k$ from the items in $\cup_s V^{(s)}$ so that in each group G_j for $j = 1, \ldots, k$ the sum of the sizes is in the range $[\varepsilon, 2\varepsilon)$ and $k \geq \frac{7}{2}\varepsilon^{-2}$. This is possible as each item has size at most ε, so we can form groups of the right size, and the number of groups that can be formed is at least ε^{-2}, since ${\beta'}_p(\cup_s V^{(s)}) \geq \frac{7}{2}\varepsilon^{-1}$, and the effective bandwidth of each item in $\cup_s V^{(s)}$ is equal to its size by definition (since $p^s s p^{-s} = s$). Moreover, the number of groups formed is at most $\varepsilon^{-1} \cdot {\beta'}_p(\cup_s V^{(s)}) \leq \frac{7}{2}\varepsilon^{-2}$.

A subset $\mathcal{I} \subseteq \{1, \ldots, k\}$ is called *critical* if it is minimal subset to the property that the total size of the items in $\cup_{j\in\mathcal{I}} G_i$ exceeds 1. We note the following facts:

- The number of groups in a critical set is at least $1 + \frac{\varepsilon^{-1}}{2}$ and at most $1 + \varepsilon^{-1}$.
- The probability that all items are on in the groups of a critical set is at least $p^{1+2\varepsilon}$. This follows from the facts that the total size of items in the groups of a critical set is at most $1 + 2\varepsilon$, and each item in $\cup_j G_j$ of size s has

probability p^s.

- The number of critical sets is at least $\frac{1}{2}(\frac{\varepsilon^{-1}}{2})^{-\frac{\varepsilon^{-1}}{2}}$. To see this consider the set of critical sets. The critical sets form a maximal antichain. We want to use Claim 4.13 for this antichain. From the first fact we see that the claim should be applied to $1+\frac{\varepsilon^{-1}}{2} \le 1+\varepsilon^{-1} \le \frac{k}{2}$. We have that $k \ge \varepsilon^{-2}$; therefore, the number of critical sets is at least

$$\binom{\varepsilon^{-2}}{1+\frac{\varepsilon^{-1}}{2}} \Big/ \binom{1+\varepsilon^{-1}}{1+\frac{\varepsilon^{-1}}{2}}.$$

  To get the claimed bound above, we bound $\binom{1+\varepsilon^{-1}}{1+\frac{\varepsilon^{-1}}{2}} \le 2^{1+\varepsilon^{-1}}$, and $\binom{\varepsilon^{-2}}{1+\frac{\varepsilon^{-1}}{2}} \ge (2\varepsilon-1)^{\frac{\varepsilon^{-1}}{2}}$.

We say that a group is *on* if all elements of the group are on, and the group is *off* if at least one element in the group is not on. The probability that a group is on is at least p^ε and at most $p^{2\varepsilon}$. Consider a critical set $\mathcal{I}$. The probability that all groups not in $\mathcal{I}$ are off is at least

$$(1-p^\varepsilon)^k = (1-p^\varepsilon)^{\frac{7}{2}\varepsilon^{-2}} \ge e^{-\frac{\frac{7}{2}\varepsilon^{-2}}{p^{-\varepsilon}}}.$$

By the choice of ε we have that

$$\varepsilon^{-2} = 36 \cdot \frac{\log p^{-1}}{\log\log p^{-1}}$$

and $p^{-\varepsilon} = e^{\Theta(\sqrt{\log p^{-1} \log\log p^{-1}})}$, and so $\frac{7}{2}\varepsilon^{-2} \le p^{-\varepsilon}$ and the probability that all groups not in $\mathcal{I}$ are off is at least e^{-1}.

Now, the probability of overflow is at least the sum, over all critical sets $\mathcal{I}$, of the probability that the groups which are on are precisely those in $\mathcal{I}$. Thus, by the above bounds, we have all

$$\Pr\left[\sum_i X_i \ge 1\right] \ge \frac{1}{2}\left(\frac{\varepsilon^{-1}}{2}\right)^{-\frac{\varepsilon^{-1}}{2}} p^{1+2\varepsilon}\frac{1}{e}.$$

Finally, since $p \ge (18\varepsilon^2)^{(\varepsilon^{-2}/18)}$, it is straightforward to see that

$$\frac{1}{2}\left(\frac{\varepsilon^{-1}}{2}\right)^{-\frac{\varepsilon^{-1}}{2}} p^{2\varepsilon}\frac{1}{e} > 1. \qquad \square$$

This lemma allows us to give a stronger analysis of our algorithm: although the algorithm only recognizes small and large items, our analysis further partitions the large items depending on whether their sizes are smaller or larger than $\varepsilon = \frac{1}{6}\sqrt{\frac{\log\log p^{-1}}{\log p^{-1}}}$. For large items with sizes below ε, we apply Lemma 4.12.

THEOREM 4.14. *The above algorithm finds a packing of items in bins of size* 1 *with overflow probability* p *such that the number of bins used is at most* $O(\sqrt{\frac{\log p^{-1}}{\log\log p^{-1}}})\, B^* + O(\log p^{-1})$*, where* B^* *is the minimum possible number of bins.*

Proof. Although the algorithm only recognizes small and large items, our analysis makes use of three types of items. Let $\varepsilon = \frac{1}{6}\sqrt{\frac{\log\log p^{-1}}{\log p^{-1}}}$, and say that an item of type

(q, s) is *large* if $s \geq \frac{1}{2}\varepsilon^{-1}$, *small* if $s \leq \log p^{-1}$, and *normal* otherwise. By earlier arguments, the algorithm is within a constant factor of optimal on small items and within an $O(\varepsilon^{-1})$ factor on large items. The problem is that the algorithm treats normal items as though they were large and packs them according to ${\beta'}_p$ applied to their effective sizes. Since there are $\Theta(\log p^{-1})$ different effective sizes of normal items, our analysis cannot consider each effective size separately. Thus, we use Lemma 4.12.

As before, we distinguish (normal) items as having *large* or *normal* probability. By giving up a factor of 2, we can still afford to analyze bins with items of normal probability separately from those with items of large probability. We say that a bin with large probability items of effective size s is *filled* if there are $\frac{1}{s} - 1$ items in the bin. For each effective size there is at most one bin of large probability items that is not filled. For a bin with normal probability items we say that the bin is *filled* if the total probability of the items in the bin is at least $p^s/2se$ (half of the maximum possible). Again, for each effective size there is at most one bin of normal probability items that is not filled. So the number of nonfilled bins of normal items is $O(\log p^{-1})$.

Next we consider filled bins. We claim that the total ${\beta'}_p$-value in a filled bin is at least $\frac{1}{2e}$. To show this, we first recall that the effective size is smaller than the real size, and that effective bandwidth is monotone in the size. This implies that a large probability item X of effective size s has effective bandwidth ${\beta'}_p(X) \geq s$. Therefore, the effective bandwidth of a bin filled with large probability items of effective size s is at least $s(\frac{1}{s} - 1) = 1 - s$. A small probability item X of probability q and effective size s has effective bandwidth at least ${\beta'}_p(X) \geq sqp^{-s}$, and so the effective bandwidth of a bin filled with normal probability items of effective size s is at least $\frac{p^s}{2es} p^{-s} s = \frac{1}{2e}$.

If our algorithm produces B filled bins, then the total effective bandwidth over all items in filled bins is at least $\frac{B}{2e}$. Lemma 4.12 implies that any packing of these items with fewer than $\frac{\varepsilon B}{14e}$ bins would result in at least one bin with too high an overflow probability. Thus the number of bins used for normal sized items is at most $O(\varepsilon^{-1})B^* + O(\log p^{-1})$, and we are finished. □

It is natural to ask whether this analysis can be further tightened to show that the same algorithm is in fact producing a packing with $O(B^*) + O(\log p^{-1})$ bins. In fact this is not possible; this is contained in the following theorem.

THEOREM 4.15. *There exist instances in which B^* is arbitrarily large, and the above algorithm uses more than $B^* \cdot \Omega(\log\log\log\log p^{-1})$ bins.*

Proof. Let b be an arbitrarily large constant, let $k = \frac{\ln\ln p^{-1}}{\ln\ln\ln p^{-1}}$, and let J be the set of all prime numbers less than or equal to k. For $r \in J$ and $1 \leq i \leq b$, let X_r^i be a Bernoulli trial of type $(p^{1/r}, 1/r)$.

First, we claim that the set of items $S^i = \{X_r^i : r \in J\}$ can be packed in a single bin. To prove this, consider any set $S' \subset S$ whose sizes sum to a number strictly greater than 1; call such a set *large*. Since the sizes of the items in S' have denominators that are pairwise relatively prime, the sum of these sizes must be at least $1 + 1/k!$. Thus we have

$$\Pr[S \text{ overflows}] \leq \sum_{large\, S' \subset S} \Pr[S' \text{ is on}] \leq 2^k p^{1+1/k!} \leq p,$$

with the last inequality following from the fact that

$$k < \frac{1}{k!} \ln p^{-1}.$$

Thus, the set of all items can be packed in b bins.

Now consider the packing produced by our algorithm. Of the items of size r, it will pack at most r in each bin. Thus, the total number of bins it produces will be at least

$$\sum_{r \in J} \frac{b}{r} = \Theta(b \log\log k) = \Theta(b \log\log\log\log p^{-1}). \qquad \square$$

The form of the final bound suggests that it is possible that our analysis could be tightened further, albeit not to provide a constant ratio.

5. The knapsack problem. Finally, we consider the knapsack problem. First we consider a simple version of the knapsack problem with items $X_1, X_2, \ldots, X_n$ that are independent Bernoulli trials. Each item has a value v_i, and we are given a knapsack size, say 1, and an allowed probability of overflow p. The problem is to find a set of items of maximum value such that the probability that the total size of the set exceeds 1 is at most p.

The lower bounds and techniques developed in the previous section yield similar results for the knapsack problem. We distinguish items by their sizes (small, normal, and large), we group large items by their effective size, and we distinguish large and small probability items just as in the previous section. The solution we construct for the knapsack problem only contains one type of item (either small, normal, or large with a given effective bandwidth). We will look for a near-optimal solution in each of these groups and select the best alternative. Thus, we can show the following.

THEOREM 5.1. *Let $X_1, \ldots, X_n$ be independent Bernoulli trials.*

- *There is a polynomial time algorithm that finds a solution to the knapsack problem with items $X_1, \ldots, X_n$ of value at least an $O(\log p^{-1})$ fraction of the optimum.*
- *For any $\varepsilon > 0$, there is a polynomial time algorithm that finds a solution to the knapsack problem, using knapsack size $1 + \varepsilon$ and overflow probability p, of value at least an $O(\varepsilon^{-1})$ fraction of the maximum possible with a knapsack of size 1 and overflow probability p.*
- *For any $\varepsilon > 0$, there is a polynomial time algorithm that finds a solution to the knapsack problem, using knapsack size 1 and overflow probability p, of value at least an $O(\varepsilon^{-1})$ fraction of the maximum possible with a knapsack of size 1 and overflow probability $p^{1+\varepsilon}$.*

Proof. For small items we use a knapsack approximation algorithm to find a set of items of approximately maximum value with at most a total of $\frac{1}{2}$ expected value.

For normal items we use a knapsack approximation algorithm to find a set of items of approximately maximum value with at most a total of ε total effective bandwidth. As in the previous section, we need to increase or decrease the item sizes by a factor of $(1 + \varepsilon)$ before computing the effective bandwidth depending on the type of result desired.

We group large items according to their effective size. For large items of effective size s we either (i) take the (at most) $\frac{1-s}{s}$ items of largest value; or (ii) use a knapsack approximation algorithm on the large items of normal probability to find a set of approximately maximum value with a total probability of at most $\frac{p^s}{es}$.

To prove the lower bound we note that the optimal value of a deterministic knapsack problem grows only linearly with the knapsack size, as long as new items of larger size are not considered. Our approximation algorithm is simply the greedy algorithm, which either (i) takes the single most valuable item, or (ii) orders the items

in decreasing order of value divided by weight (i.e., expectation, effective bandwidth, or probability) and greedily fills the knapsack in this order.

FACT 5.2. *Consider a deterministic knapsack problem where the knapsack has size* 1*, and all items have size at most* 1*.*

- *The value of the solution obtained by the above greedy method is at least a* $\frac{1}{2}$ *of the optimal.*
- *For any* $c > 1$*, the value of the optimal solution with knapsacks of size* 1 *is at least a fraction of* $\frac{1}{2c}$ *of the optimal value with knapsack size* c*.*

For small items Lemma 4.2 implies that the resulting knapsack solution is feasible, and Lemmas 5.2 and 4.2 imply that its value is within a constant factor of the optimal packing of small items.

For normal items we use Lemma 4.3 to show either that (1) the packing obtained is feasible with a knapsack of size $1 + \varepsilon$ or (2) if we increased sizes before computing the effective bandwidth, then it is feasible with a knapsack of size 1. Lemmas 5.2 and 4.6 imply that the packing obtained is within an $O(\varepsilon^{-1})$ factor of any packing of normal items with either (1) a knapsack of size 1 and overflow probability at most p or (2) a knapsack of size 1 and overflow probability at most $p^{1+\varepsilon}$.

For large items of effective size s, Lemma 4.1 shows that the solution is feasible, and Lemmas 4.7 and 4.8 and Fact 5.2 imply that the solution is within a constant factor to any packing using items of effective size s.

In total we get an $O(1)$ approximation algorithm for small items, and items of a given effective size. There are $O(\varepsilon^{-1})$ different effective sizes. For normal items, we get an $O(\varepsilon^{-1})$ approximation. The approximation ratio of the best solution among these options is the sum of the approximation ratios of the special cases. Thus, the theorem follows. □

Extension to other distributions. Next we extend the solution to a distribution that is somewhat more general than the on-off distribution we have been considering so far. We assume that each item is parameterized by numbers lo, hi, and q, where the item is of size hi with probability q and of size $lo \leq hi$ otherwise. This kind of item is a simple model of a bursty communication, with lo being the normal rate of transmission and hi being a burst that occurs with probability q.

Consider the optimal knapsack packing. Assume we used items $X_1, \ldots, X_k$ in the packing, and let $g = \sum_i lo_i$. The idea is that we guess the value of g in the optimal solution, and for every guess we look for packings that are feasible and in which the total of the lo values is at most g.

We define small items depending on the size $s_i = hi_i - lo_i$ of the probabilistic part, limiting the value s_i to be at most half of what it was for the size in the case of Bernoulli trials. We pack small items using expectation, subject to the the fact that the total expectation is at most $\frac{1}{2}$, and we use Lemmas 4.2 and 4.5 and Fact 5.2 to see that the value of the packing is within a constant factor of the maximum possible using small items.

We define normal items also using $s_i = hi_i - lo_i$ and pack normal items using the effective bandwidth. However, to compute the right effective bandwidth for normal items we need to know $1 - g$, the amount of space left for the probabilistic part of the items, since the effective bandwidth formula assumed that the bin size is 1, so we will have to rescale the bin size to apply the formula. Notice, however, that it suffices to know $1 - g$ roughly up to a factor of $1 + \varepsilon$.

Notice that it is essentially no loss of generality to assume that $g \leq \frac{1}{2}$. Among items with lo values above $\frac{1}{2}$ at most one can be in the knapsack, so we can either

pack a single one of these items in the knapsack by itself or assume that we are not using any of them. For items with $lo \leq \frac{1}{2}$ the restriction that $g \leq \frac{1}{2}$ will not change the optimum value by more than a small constant factor. Therefore, we need only to consider $O(\varepsilon^{-1})$ different values for g to obtain an $O(\varepsilon^{-1})$ approximate solution the optimal value of a solution using normal items.

In the case of Bernoulli trials we used the greedy method to pack items in each category into a knapsack using expectation, effective size, or the probability depending on whether we considered items that are small, normal, or large. Here we cannot use the greedy method to find a solution to the resulting deterministic problem, as we have to consider an extra parameter, the sum of the lo values. We use the following method instead.

A deterministic *two-dimensional knapsack problem* is defined by a set of items $X_1, \ldots, X_n$, each with a value v_i, size s_i, and weight w_i. In addition, we are given a knapsack size S and a weight limit W. The problem is to find a subset $\mathcal{I}$ of items of maximum total value so that the total size $\sum_{i \in \mathcal{I}} s_i$ is at most S and the total weight $\sum_{i \in \mathcal{I}} w_i$ is at most W.

LEMMA 5.3. *A simple greedy algorithm yields a constant factor approximation for the two-dimensional knapsack problem. Using dynamic programming we can obtain a* $1 + \delta$*-approximation for any fixed value* $\delta > 0$.

Proof. First notice that it is no loss of generality to assume that the size S and the weight limit W are both 1.

Consider items that have both size and weight at most $\frac{1}{2}$. We claim that the following greedy method provides an approximation: Find a greedy solution to maximizing using the sum $s_i + w_i$ as size, i.e., this greedy algorithm approximates the maximum possible value subject to the limit that the total size *plus* the total weight is at most 1. The optimal solution has size at most 1 and weight at most 1, so the sum of the total size and weight is at most 2. Hence, by Fact 5.2, the value obtained is at least $\frac{1}{4}$th of the optimal.

For items that have either size above $\frac{1}{2}$ or weight above $\frac{1}{2}$ at most 2 can fit in a knapsack, so we can get the optimal solution by trying all pairs.

The better of the two solutions obtained has a value of at least $\frac{1}{5}$th of the optimum. □

Next we consider large items. The definition of effective size is also related to $1 - g$: The effective size of an item of type (hi, lo, q) is defined as $\bar{s}$, where $\frac{1}{\bar{s}} = \min\{j : j(hi - lo) > 1 - g\}$, the number of copies the probabilistic part of this item can fit in a knapsack of size $1 - g$. Given an estimate for the value g, we group large items according to their effective size and pack items of one effective size using the two-dimensional knapsack problem above. As before, we separate items of identical effective size, depending on whether they have large or normal probability. For large probability items of effective size s, we need at most $\frac{1}{s} - 1$ items of total lo value at most g and maximum total value. For normal probability items one dimension is the lo value, where the total lo value is limited to g, and the other dimension is the probability, bounded by p^s/se. In both cases we can use Lemma 5.3 to obtain a knapsack solution.

Next we consider the issue of how many different estimates we have to consider in order to get a near-optimal solution using large items. Depending on the value of g the effective size of an item can change. Each item can have at most ε^{-1} different effective sizes, and hence it creates at most ε^{-1} different "cut-off" values for g. Hence, the grouping of large items changes for at most $n\varepsilon^{-1}$ discrete values of g, where n

is the number of large items. This implies that it suffices to try $O(n\varepsilon^{-1})$ different g values in order to get an approximately optimal solution.

The above discussion proves the following theorem.

THEOREM 5.4. *Let $X_1, \ldots, X_n$ be independent trials of the type defined above.*

- *There is a polynomial time algorithm that finds a solution to the knapsack problem with items $X_1, \ldots, X_n$ of value at least an $O(\log p^{-1})$ fraction of the optimum.*
- *For any $\varepsilon > 0$, there is a polynomial time algorithm that finds a solution to this knapsack problem using a knapsack size $1 + \varepsilon$ and overflow probability at most p of value at least an $O(\varepsilon^{-1})$ fraction of the maximum possible with a knapsack of size 1 and overflow probability p.*
- *For any $\varepsilon > 0$, there is a polynomial time algorithm that finds a solution to this knapsack problem using a knapsack size 1 and overflow probability p of value at least an $O(\varepsilon^{-1})$ fraction of the maximum possible with a knapsack of size 1 and overflow probability $p^{1+\varepsilon}$.*

Bin-packing with other distributions. Using the knapsack result and set-cover we get a bin-packing algorithm for independent items of type (lo, hi, q). To get a solution to the bin-packing problem we repeatedly take the maximum number of items possible to include in a single bin. The result is an $O(\log n)$ extra factor in the approximation ratio.

COROLLARY 5.5. *Let $X_1, \ldots, X_n$ be independent trials of the type defined above.*

- *There is a polynomial time algorithm that finds a solution to the bin-packing problem with items $X_1, \ldots, X_n$ using at most $O(\log p^{-1} \log n)$ times the minimum possible number of bins.*
- *For any $\varepsilon > 0$, there is a polynomial time algorithm that finds a solution to this bin-packing problem using bins of size $1 + \varepsilon$ and overflow probability at most p with a number of bins that is at most $O(\varepsilon^{-1} \log n)$ times the minimum possible with bins of size 1 and overflow probability p.*
- *For any $\varepsilon > 0$, there is a polynomial time algorithm that finds a solution to this bin-packing problem using bins size 1 and overflow probability p with at most $O(\varepsilon^{-1} \log n)$ times as many bins as the minimum possible with bins of size 1 and overflow probability $p^{1+\varepsilon}$.*

6. Extensions to general networks. The model we have been considering—two nodes communicating over a set of parallel links—is a common one in the study of bursty traffic. However, it is interesting to consider the extent to which one can carry over the results developed here to the problem of routing bursty connections in a general network. The model for a general network follows directly from the discussion in the introduction: we are given a graph $G = (V, E)$ with capacities $\{c_e\}$ on the edges, and source-sink pairs (s_i, t_i) indicating connection requests in the network. For each source-sink pair, we are given a random variable X_i corresponding to the demand of that connection; a *routing* is a choice of a path P_i in G for each connection (s_i, t_i).

There are several options for how one might want to model the *capacity constraints* for a problem of this type; we define two main possibilities here. Suppose we are given an allowed overflow probability p.

(i) The *link-based overflow constraint* requires that for each edge e, we have $\Pr[\sum_{i:e \in P_i} X_i > c_e] \leq p$.

(ii) The *connection-based overflow constraint* requires that for each connection

(s_i, t_i), we have

$$\Pr\left[\exists e \in P_i : \sum_{i:e\in P_i} X_i > c_e\right] \leq p.$$

One can argue that from the perspective of providing guaranteed quality of service to users in a network, the *connection-based overflow constraint* is more natural. In this section we use this model.

Now suppose we are in a "high-capacity" setting in which the capacity of every edge exceeds the *peak* bandwidth rate of every connection X_i by a factor of $c\log(p^{-1}|E|)$ for an appropriate constant c. Let us define the *value* of a set of connections to be the sum of their expectations; we consider the problem of accepting a set of connections of maximum value. We run the on-line algorithm of Awerbuch, Azar, and Plotkin [2], using $\mathrm{E}[X_i]$ as the demand for connection (s_i, t_i) and $\frac{1}{4}c_e$ as the capacity of edge e. The analysis of [2] can then be used to show the following.

LEMMA 6.1. *For any constant γ there is a constant C such that, if k denotes the total value of connections accepted by the algorithm, then in any routing of a set of connections of value at least $C(\log|E|)k$, there is some edge e carrying a total expected value greater than γc_e.*

THEOREM 6.2. *The set of connections accepted by the above algorithm satisfies the connection-based overflow constraints, and the total value of the connections accepted is within an $O(\log|E|)$ factor of the off-line optimum on the graph G.*

Proof. Without loss of generality, we may assume that the minimum edge capacity in the network is 1. Recall our assumption that the peak rate of any connection X is at most $1/(c\log(p^{-1}|E|))$; thus, for each connection X, the effective bandwidth $\beta_{p|E|^{-1}}(X)$, with respect to probability $\frac{p}{|E|}$, is at most $2\mathrm{E}[X]$. Now Proposition 3.6 implies that our routing satisfies the link-based overflow constraint with probability $\frac{p}{|E|}$ and hence the connection-based overflow constraints with probability p.

To compare our performance to that of the optimum, we use Lemma 6.1 with $\gamma = 8$. Further, we give up a constant factor in the approximation ratio and use Lemma 3.10 to model each connection. Using the notation of the lemma we model a connection X as a sum of of independent Bernoulli trials $\frac{1}{2}Y = \sum_i \frac{1}{2}Y_i$ whose peak rates are inverse powers of two, such that $Y \leq X$ and $\mathrm{E}(Y) \leq \frac{1}{2}\mathrm{E}(X)$.

Lemma 4.5 shows that such a routing violates the link-based overflow constraint on edge e, and hence any path through the edge e violates the connection-based overflow constraint. It follows that our routing is within $O(\log|E|)$ of optimal. ∎

We note that the analysis of [2] also allows us to provide performance guarantees in terms of more general notions of "value."

REFERENCES

[1] J. ASPNES, Y. AZAR, A. FIAT, S. PLOTKIN, AND O. WAARTS, *On-line load balancing with applications to machine scheduling and virtual circuit routing*, in Proceedings of the 25th ACM Symposium on the Theory of Computing, San Diego, CA, 1993, pp. 623–631.

[2] B. AWERBUCH, Y. AZAR, AND S. PLOTKIN, *Throughput-competitive online routing*, in Proceedings of the 34th IEEE Foundations in Computer Science, Palo Alto, CA, 1993, pp. 32–40.

[3] Y. BARTAL, A. FIAT, H. KARLOFF, AND R. VORHA, *New algorithms for an ancient scheduling problem*, in Proceedings of the 24th ACM Symposium on the Theory of Computing, Vancouver, British Columbia, Canada, 1992, pp. 51–58.

[4] A. ELWALID AND D. MITRA, *Effective bandwidth of general Markovian traffic sources and admission control of high speed networks*, IEEE Trans. Networking, 1 (1993), pp. 329–343.

[5] R. Gawlick, C. Kalmanek, and K. G. Ramakrishnan, *On–line routing for permanent virtual circuits*, in Proceedings of INFOCOM, Boston, MA, 1995, pp. 278–288.
[6] R. Gibbens and P. Hunt, *Effective bandwidths for the multi-type UAS channel*, Queueing Systems Theory Appl., 9 (1991), pp. 17–27.
[7] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 563–1581.
[8] R. L. Graham, *Bounds on multiprocessing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.
[9] D. S. Hochbaum and D. B. Shmoys, *Using dual approximation algorithms for scheduling problems: Theoretical and practical results*, J. ACM, 34 (1987), pp. 144–162.
[10] J. Y. Hui, *Resource allocation for broadband networks*, IEEE J. Selected Areas in Comm., 6 (1988), pp. 1598–1608.
[11] F. P. Kelly, *Effective bandwidths at multi-class queues*, Queueing Systems Theory Appl., 9 (1991), pp. 5–15.
[12] F. P. Kelly, *Notes on effective bandwidths*, in F. P. Kelly, S. Zachary, I. B. Zeidins, eds., Stochastic Networks: Theory and Applications, Oxford University Press, Oxford, 1996, pp. 141–168.
[13] F. P. Kelly, S. Zachary, and I. B. Zeidins, eds., *Stochastic Networks: Theory and Applications*, Oxford University Press, Oxford, 1996.
[14] T. Lehtonen, *Scheduling jobs with exponential processing times on parallel machines.* J. Appl. Probab., 25 (1988), pp. 752–762.
[15] M. Pinedo, *Scheduling: Theory, Algorithms and Systems*, Prentice-Hall, Upper Saddle River, NJ, 1995.
[16] S. Plotkin, *Competitive routing in ATM networks*, IEEE J. Selected Areas in Communications, 13 (1995), pp. 1128–1136.
[17] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.
[18] G. de Veciana and J. Walrand, *Effective bandwidths: Call admission, traffic policing and filtering for ATM networks*, Queueing Systems Theory Appl., 20 (1995), pp. 37–39.
[19] G. Weiss, *Approximation results in parallel machines stochastic scheduling*, Ann. of Oper. Res., 26 (1990), pp. 195–242.
[20] G. Weiss, *A tutorial in stochastic scheduling*, in Scheduling Theory and Its Applications, P. Chretienne, E. G. Coffman, J. K. Lenstra, and Z. Liu, eds., Wiley, New York, 1995, pp. 33–64.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 218–246

MOTION PLANNING OF LEGGED ROBOTS*

JEAN-DANIEL BOISSONNAT†, OLIVIER DEVILLERS†, AND SYLVAIN LAZARD‡

Abstract. We study the problem of computing the free space $\mathcal{F}$ of a simple legged robot called the spider robot. The body of this robot is a single point and the legs are attached to the body. The robot is subject to two constraints: each leg has a maximal extension R (accessibility constraint) and the body of the robot must lie above the convex hull of its feet (stability constraint). Moreover, the robot can only put its feet on some regions, called the foothold regions. The free space $\mathcal{F}$ is the set of positions of the body of the robot such that there exists a set of accessible footholds for which the robot is stable. We present an efficient algorithm that computes $\mathcal{F}$ in $O(n^2 \log n)$ time using $O(n^2\alpha(n))$ space for n discrete point footholds where $\alpha(n)$ is an extremely slowly growing function ($\alpha(n) \leqslant 3$ for any practical value of n). We also present an algorithm for computing $\mathcal{F}$ when the foothold regions are pairwise disjoint polygons with n edges in total. This algorithm computes $\mathcal{F}$ in $O(n^2\alpha_8(n)\log n)$ time using $O(n^2\alpha_8(n))$ space. ($\alpha_8(n)$ is also an extremely slowly growing function.) These results are close to optimal since $\Omega(n^2)$ is a lower bound for the size of $\mathcal{F}$.

Key words. legged robots, computational geometry, motion planning

AMS subject classification. 68U05

PII. S0097539797326289

1. Introduction.

Although legged robots have already been studied in robotics [13, 14], only a very few papers consider the motion planning problem amidst obstacles [9, 8, 2]. In [9, 8], some heuristic approaches are described, while, in [2], efficient and provably correct geometric algorithms are described for a restricted type of legged robot, the so-called spider robot to be defined precisely below, and for finite sets of point footholds.

A *legged robot* consists of a body with legs. Each leg has one end attached to the body and the other end (called the foot) that can lie on the ground (or move in space between two positions on the ground). Compared to the classic piano movers problem, legged robots introduce new types of constraints. We assume that the environment consists of regions in the plane, called *foothold regions*, where the robot can safely put its feet. A *foothold* is a point in a foothold region. The legged robot must satisfy two different constraints: the accessibility and the stability constraints. A foothold is said to be *accessible* from a *placement* (position of the body of the robot) if it can be reached by a leg of the robot. A placement is called *stable* if there exist accessible footholds and if the center of mass of the robot lies above the convex hull of these accessible footholds. The set of stable placements is clearly relevant for planning the motion of a legged robot: we call this set the *free space* of the legged robot. Note that a legged robot has at least four legs; three legs ensure the stability of a placement and a fourth leg permits the motion of the robot.

A first simple instance of a legged robot is the *spider robot* (see Figure 1.1). The spider robot was inspired by Ambler, developed at Carnegie Mellon University [1].

*Received by the editors August 20, 1997; accepted for publication (in revised form) August 19, 1999; published electronically May 2, 2000. Part of these results have been presented in the Proceedings of the 11th IEEE Internat. Conf. Robot. Autom., San Diego, CA, 1994, and the First Workshop on the Algorithmic Foundation of Robotics, Boston, MA 1994.

http://www.siam.org/journals/sicomp/30-1/32628.html

†INRIA Sophia-Antipolis, BP 93, 06902 Sophia Antipolis Cedex, France (jean-daniel.boissonnat@sophia.inria.fr, olivier.devillers@sophia.inria.fr).

‡INRIA Lorraine, 615 rue du jardin botanique, B.P. 101, 54602 Villers-les-Nancy Cedex, France (lazard@loria.fr). Most of this work was completed while this author was at INRIA Sophia-Antipolis.

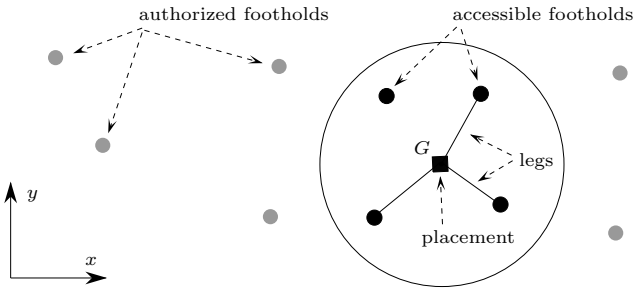


FIG. 1.1. *The spider robot.*

The body of the spider robot is a single point in the Euclidean plane, and all its legs are attached to the body. The legs are retractable and their lengths may vary between 0 and a constant R. We also assume that the center of mass of the robot is its body. It follows that a placement is stable if the body of the robot lies above the convex hull of the accessible footholds.

The constraint that the body of the spider robot lies in the plane (instead of in three dimensions) is not really restrictive. Indeed, consider a legged robot for which that constraint is relaxed. Then, if a placement (x, y, z) of such a legged robot is stable, then any placement (x, y, z'), $0 \leqslant z' \leqslant z$, is also stable. Reciprocally, it can be shown that if (x, y) is in the interior of the free space of the spider robot, then there exists $z > 0$ such that (x, y, z) is a stable placement of the corresponding legged robot.

The problem of planning the motion of a spider robot has already been studied by Boissonnat et al. [2]. However, their method assumes that the set of footholds is a finite set of points and cannot be generalized to more complex environments. This paper proposes a new method for computing the free space of a spider robot in the presence of polygonal foothold regions. This method is based on a transformation between this problem and the problem of moving a half-disk amidst obstacles. Our method requires the computation of some parts of the free space of the half-disk. These computations are rather technical and complicated. Consequently, for the sake of clarity, we first present our algorithm for the simple case of discrete footholds, and then we show how it can be generalized to the case of polygonal foothold regions.

Once the free space of the spider robot has been computed, it can be used to find trajectories and sequences of legs assignments allowing the robot to move from one point to another. Indeed, once the free space is known, a trajectory of the body can be found in the free space. Then, a sequence of legs assignments can be computed as follows (see [2] for details). Given an initial legs assignment, the body of the robot moves along its trajectory until it crosses the convex hull of its (three) feet that are on the ground, or one leg reaches its maximal extension. Then, a suitable foothold is found for the fourth leg and one leg leaves its foothold.

The paper is organized as follows: some notations and results of [2] are recalled in the next section. Section 3 shows the transformation between the spider robot problem and the half-disk problem. We present in section 4 our algorithm for computing the free space of a spider robot for a discrete set of footholds. Section 5 shows how to extend the algorithm to polygonal foothold regions.

2. Notations and previous results. In sections 2, 3, and 4, $\mathcal{S}$ denotes a discrete set of distinct footholds $\{s_1, \ldots, s_n\}$ in the Euclidean plane. ($\mathcal{S}$ will denote

in section 5 a set of disjoint polygonal regions.) Point G denotes the body of the robot (in the same plane) and $[0, R]$ is the length range of each leg. The free space $\mathcal{F}$ is the set of all stable placements of G. A placement is said to be at the *limit of stability* if it lies on the boundary of the convex hull of its accessible footholds. Notice that $\mathcal{F}$ is a closed set and contains the placements at the limit of stability.

Let C_i denote the circle of radius R centered at s_i. $\mathcal{A}$ is the arrangement of the circles C_i for $1 \leqslant i \leqslant n$, i.e., the subdivision of the plane induced by the circles. This arrangement plays an important role in our problem and we will express the complexity results in term of $|\mathcal{A}|$, the size of $\mathcal{A}$. In the worst case, $|\mathcal{A}| = \Theta(n^2)$, but if k denotes the maximum number of disks that can cover a point of the plane, among the disks of radius R centered at the s_i, it can be shown that $|\mathcal{A}| = O(kn)$ [15]. Clearly k is not larger than n and in case of sparse footholds, $|\mathcal{A}|$ may be linearly related to the number of footholds.

For any set $\mathcal{E}$, let $\partial(\mathcal{E})$ denote its boundary, $\mathrm{CH}(\mathcal{E})$ its convex hull, $int(\mathcal{E})$ its relative interior,[1] $clos(\mathcal{E})$ its closure, and $compl(\mathcal{E})$ its complementary set. Let S^1 denote the set of angles $\mathbb{R}/2\pi\mathbb{Z}$. We denote by $x = y[p]$ the equality of x and y modulo p. We say in the following that two objects *properly intersect* if and only if their relative interiors intersect.

The algorithm described in [2] is based on the following observation: for G in a cell Γ of $\mathcal{A}$, the set of footholds that can be reached by the robot is fixed; the portion of Γ that belongs to $\mathcal{F}$ is exactly the intersection of Γ with the convex hull of the footholds that can be reached from Γ. Therefore, the edges of $\partial(\mathcal{F})$ are either circular arcs belonging to $\mathcal{A}$ or portions of line segments joining two footholds. Moreover, a vertex of $\partial(\mathcal{F})$ incident to two straight edges is a foothold (see Figure 2.1). The complexity of $\mathcal{F}$ has been proved to be $|\mathcal{F}| = \Theta(|\mathcal{A}|)$ [2].

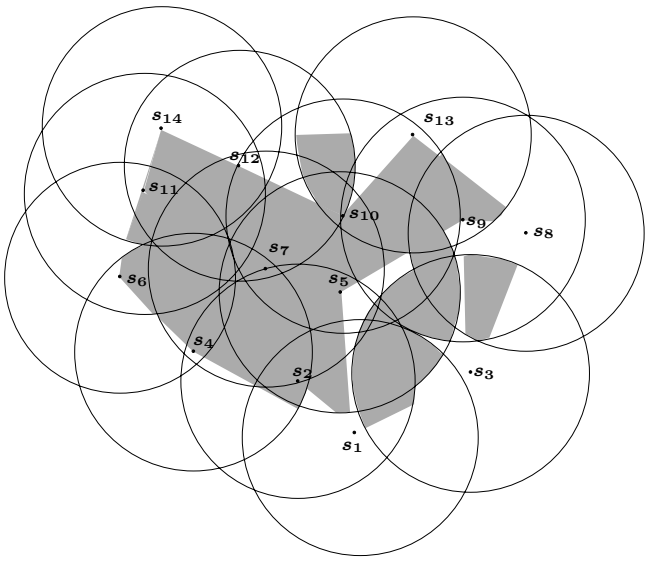


FIG. 2.1. *An example of the free space of a spider robot.*

[1] The relative interior of a set $\mathcal{E}$ in a space E is the interior of $\mathcal{E}$ in the space $\mathcal{E}$ for the topology induced by E. For example, the relative interior of a closed line segment in $\mathbb{R}^3$ is the line segment without its endpoints, though its interior in $\mathbb{R}^3$ is empty.

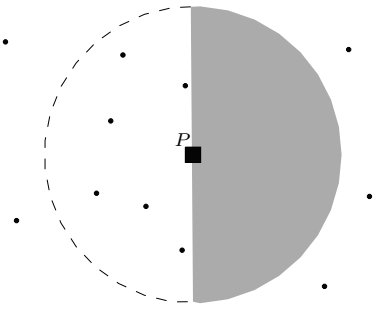


FIG. 3.1. *A placement which is not stable.*

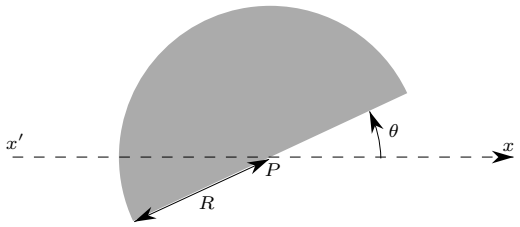


FIG. 3.2. $HD(P, \theta)$.

The algorithm presented in [2] computes the free space $\mathcal{F}$ in $O(|\mathcal{A}| \log n)$ time. It uses sophisticated data structures allowing the off-line maintenance of convex hulls.

The algorithm described in this paper has the same time complexity, uses simple data structures, and can be extended to the case where the set $\mathcal{S}$ of footholds is a set of polygonal regions and not simply a set of points. For simplicity, we consider first the case of point footholds and postpone the discussion on polygonal foothold regions to section 5.

General position assumption. To simplify the presentation of this paper, we make the following general position assumptions. All these hypotheses can be removed by a careful analysis. Recall that we consider here that the set of footholds is discrete.

No two footholds lie at distance exactly R or $2R$. Among the circles $C_1, \ldots, C_n$ and the line segments joining two footholds, the intersection between three circles, or two circles and a line segment, or one circle and two line segments is empty.

3. From spider robots to half-disk robots. In this section, we establish the connection between the free space of the spider robot and the free space of a half-disk robot moving by translation and rotation amidst n point obstacles.

THEOREM 3.1. *The spider robot does not admit a stable placement at point P if and only if there exists a half-disk (of radius R) centered at P that does not contain any foothold of $\mathcal{S}$ (see Figure* 3.1*).*

Proof. Let $\mathcal{R}$ be the set of all the footholds that are reachable from placement P. By definition, P is not stable if and only if the convex hull of $\mathcal{R}$ does not contain P (see Figure 3.1). That is equivalent to saying that there exists an open half-plane through P containing $\mathcal{R}$ or that there exists a closed half-disk of radius R centered at P which does not contain any foothold. □

DEFINITION 3.2. *Let $HD(P,\theta)$ be the half-disk of radius R centered at P (see Figure 3.2) defined by*

$$\begin{cases} (x-x_P)^2+(y-y_P)^2 \leqslant R^2, \\ (x-x_P)\sin\theta-(y-y_P)\cos\theta \leqslant 0. \end{cases}$$

DEFINITION 3.3. *For all $s_i \in \mathcal{S}$ $(1 \leqslant i \leqslant n)$, let*

$$\mathcal{H}_i = \{(P,\theta) \in \mathbb{R}^2 \times S^1 \mid P \in HD(s_i,\theta)\},$$
$$\mathcal{H} = \bigcup_{i=1}^{n} \mathcal{H}_i,$$
$$\mathcal{C}_i = C_i \times S^1.$$

$\mathcal{H}_i$ will be called the helicoidal volume centered at s_i (see Figure 3.3).

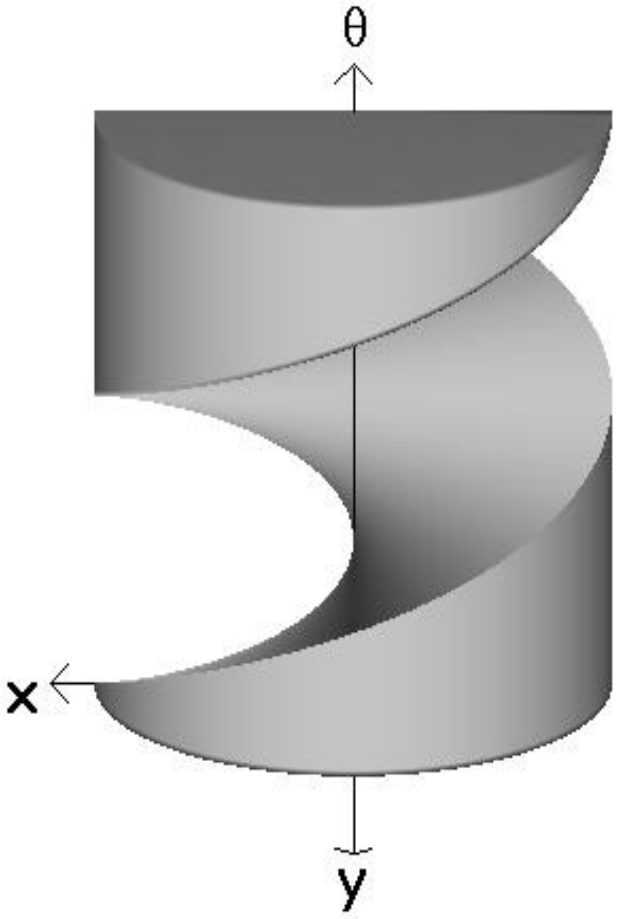


FIG. 3.3. *Helicoidal volume $\mathcal{H}_i$.*

Notice the typographical distinction between the circle C_i defined in $\mathbb{R}^2$ and the torus $\mathcal{C}_i$ defined in $\mathbb{R}^2 \times S^1$. For convenience, we will often identify S^1 and the interval $[0, 2\pi]$ of $\mathbb{R}$. This allows us to draw objects of $\mathbb{R}^2 \times S^1$ in $\mathbb{R}^3$ and to speak of the θ-axis. Π_{θ_0} denotes the "plane" $\{(P,\theta) \in \mathbb{R}^2 \times S^1 \mid \theta = \theta_0\}$.

DEFINITION 3.4. *The free space $\mathcal{L}$ of a half-disk robot moving by translation and rotation amidst the set of obstacles $\mathcal{S}$ is the set of $(P,\theta) \in \mathbb{R}^2 \times S^1$ such that the half-disk $HD(P,\theta+\pi)$ does not intersect $\mathcal{S}$.*

PROPOSITION 3.5. $\mathcal{L} = compl(\mathcal{H})$.

Proof. For all $\theta \in S^1$, the set $\mathcal{L} \cap \Pi_\theta$ is the free space of the half-disk $HD(P,\theta+\pi)$ moving by translation only amidst the obstacle $s_1,\ldots,s_n$. Since the set of points P such that $HD(P,\theta+\pi)$ contains an s_i is $HD(s_i,\theta)$, $\mathcal{L} \cap \Pi_\theta$ is the complementary set of the union of the $HD(s_i,\theta)$. Thus, $\mathcal{L}$ is the complementary set of the union of the $\mathcal{H}_i$, that is $\mathcal{H}$. □

Let $p_{//\theta}$ denote the mapping (called "orthogonal projection"): $\mathbb{R}^2 \times S^1 \longrightarrow \mathbb{R}^2, (P,\theta) \mapsto P$.

THEOREM 3.6. $\mathcal{F} = compl(p_{//\theta}(compl(\mathcal{H})))$.

Proof. By definition of $\mathcal{L}$, $p_{//\theta}(\mathcal{L})$ is the set of points $P \in \mathbb{R}^2$ such that there exists an angle $\theta \in S^1$ such that the half-disk $HD(P,\theta)$ does not intersect $\mathcal{S}$. By Theorem 3.1, it is equivalent to say that there exists $\theta \in S^1$ such that $HD(P,\theta)$ does not intersect $\mathcal{S}$, or that P is not a stable placement of the spider robot. Thus, $p_{//\theta}(\mathcal{L})$ is the set of points P, where the robot does not admit a stable placement, i.e., $\mathcal{F} = compl(p_{//\theta}(\mathcal{L}))$. The result then follows from Proposition 3.5. □

Remark 3.7. $compl(p_{//\theta}(compl(\mathcal{H}))) \times S^1$ is the largest "cylinder" included in $\mathcal{H}$, whose axis is parallel to the θ-axis (in grey in Figure 3.4). The basis of this cylinder is $\mathcal{F}$.

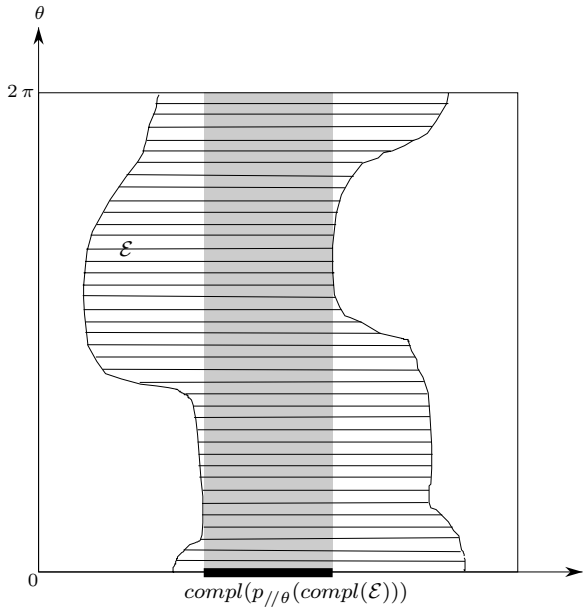


FIG. 3.4. $compl(p_{//\theta}(compl(\mathcal{E})))$.

Remark 3.8. The results of this section do not depend on the fact that the footholds are discrete points. For more general foothold regions, we simply need to replace the helicoidal volumes by their analogue. This will be done in section 5.

4. Computation of $\mathcal{F}$. In this section, we propose an algorithm for computing $\mathcal{F}$ based on Theorem 3.6.

A first attempt to use Theorem 3.6 may consist of computing $\mathcal{L} = compl(\mathcal{H})$ and projecting it onto the horizontal plane. The motion planning of a convex polygonal robot in a polygonal environment has been extensively studied (see, for example, [10, 11]). Such algorithms can be generalized to plan the motion of a half-disk. It should lead to an algorithm of complexity $O(n\lambda_s(n)\log n)$, where $\lambda_s(n)$ is an almost linear function of n. The projection can be done using classical techniques, such as projecting all the faces of $\mathcal{L}$ and computing their union. Since the complexity of the three-dimensional object $\mathcal{L}$ is not directly related to the complexity of its projection, this approach does not provide a combinatorial bound on $\mathcal{F}$. However, if we assume $|\mathcal{F}| = O(\lambda_s(|\mathcal{A}|))$ (which will be proved in this paper), the time complexity of the algorithm of Kedem Sharir, and Toledo is $O(n\lambda_s(n)\log n + \lambda_s(|\mathcal{A}|)\log^2 n)$.

In this paper, we present a direct computation of $\mathcal{F}$. This approach provides an upper bound on the size of $\mathcal{F}$, namely $|\mathcal{F}| = O(\lambda_s(|\mathcal{A}|))$. It also provides an algorithm for computing $\mathcal{F}$ in $O(\lambda_s(|\mathcal{A}|)\log n)$ time. As in [16] and contrary to [11], the algorithm proposed here is sensitive to $|\mathcal{A}|$ which is usually less than quadratic. Another advantage of our direct computation is to avoid the explicit construction of the three-dimensional object $\mathcal{L}$ which is useless for our application. Our algorithm manipulates only two-dimensional arrangements or lower envelopes, and we provide a detailed description of the curves involved in the construction.

Let us now detail the computation of $\mathcal{F}$ in the case of point footholds. We know that each arc of the boundary $\partial(\mathcal{F})$ of $\mathcal{F}$ is either a straight line segment belonging to a line joining two footholds or an arc of a circle C_i (see section 2). The circular arcs $\partial(\mathcal{F}) \cap C_i$ are computed first (sections 4.1, 4.2, and 4.3) and are linked together with the line segments in a second step (sections 4.4 and 4.5).

4.1. Computation of $\partial(\mathcal{F}) \cap \mathcal{A}$. In what follows, the *contribution* of an object X to another object Y is $X \cap Y$. We compute the contribution of each circle C_{i_0}, $i_0 = 1, \ldots, n$, to $\partial(\mathcal{F})$ in turn. Recall that $\mathcal{C}_{i_0}$ denotes the torus $C_{i_0} \times S^1$. The contribution of each circle C_{i_0} to $\partial(\mathcal{F})$ will be obtained by computing the intersection of all the $\mathcal{H}_i$, $i = 1, \ldots, n$, with the torus $\mathcal{C}_{i_0}$. Let $\mathcal{Z}_i$, $i = 1, \ldots, n$, denote these intersections:

$$\mathcal{Z}_i = \mathcal{H}_i \cap \mathcal{C}_{i_0}.$$

We first show how to compute the contribution of C_{i_0} to $\partial(\mathcal{F})$ in term of the $\mathcal{Z}_i$ and leave the studies of the shape and properties of $\mathcal{Z}_i$ to section 4.2. Figures 4.1 and 4.2 show some (hatched) $\mathcal{Z}_i \subset \mathcal{C}_{i_0}$ ($i \neq i_0$), where $\mathcal{C}_{i_0}$ is parameterized by (u, θ) (u and θ parameterize C_{i_0} and S^1, respectively); the dark grey region shows $\mathcal{Z}_{i_0}$.

PROPOSITION 4.1. *The contribution of C_{i_0} to $\partial(\mathcal{F})$ is*

$$C_{i_0} \cap \partial(\mathcal{F}) = compl(p_{//\theta}(compl(\cup_i \mathcal{Z}_i))) \setminus int(compl(p_{//\theta}(compl(\cup_{i \neq i_0} \mathcal{Z}_i)))).$$

Proof. Since $\mathcal{F}$ is a closed set, $C_{i_0} \cap \partial(\mathcal{F}) = [C_{i_0} \cap \mathcal{F}] \setminus [C_{i_0} \cap int(\mathcal{F})]$. According to Theorem 3.6, $\mathcal{F} = compl(p_{//\theta}(compl(\mathcal{H})))$. One can easily prove that for any set $\mathcal{E} \in \mathbb{R}^2 \times S^1$, $int(compl(\mathcal{E})) = compl(clos(\mathcal{E}))$, $clos(compl(\mathcal{E})) = compl(int(\mathcal{E}))$, and $clos(p_{//\theta}(\mathcal{E})) = p_{//\theta}(clos(\mathcal{E}))$. It then follows from the expression of $\mathcal{F}$ that $int(\mathcal{F}) = compl(p_{//\theta}(compl(int(\mathcal{H}))))$.

Recall that for any sets $X, Y \in \mathbb{R}^2 \times S^1$, $compl(X \cap Y) = compl(X) \cup compl(Y)$, $p_{//\theta}(X \cup Y) = p_{//\theta}(X) \cup p_{//\theta}(Y)$, and $compl(X \cup Y) = compl(X) \cap compl(Y)$. That implies

$$compl(p_{//\theta}(compl(X \cap Y))) = compl(p_{//\theta}(compl(X))) \cap compl(p_{//\theta}(compl(Y))).$$

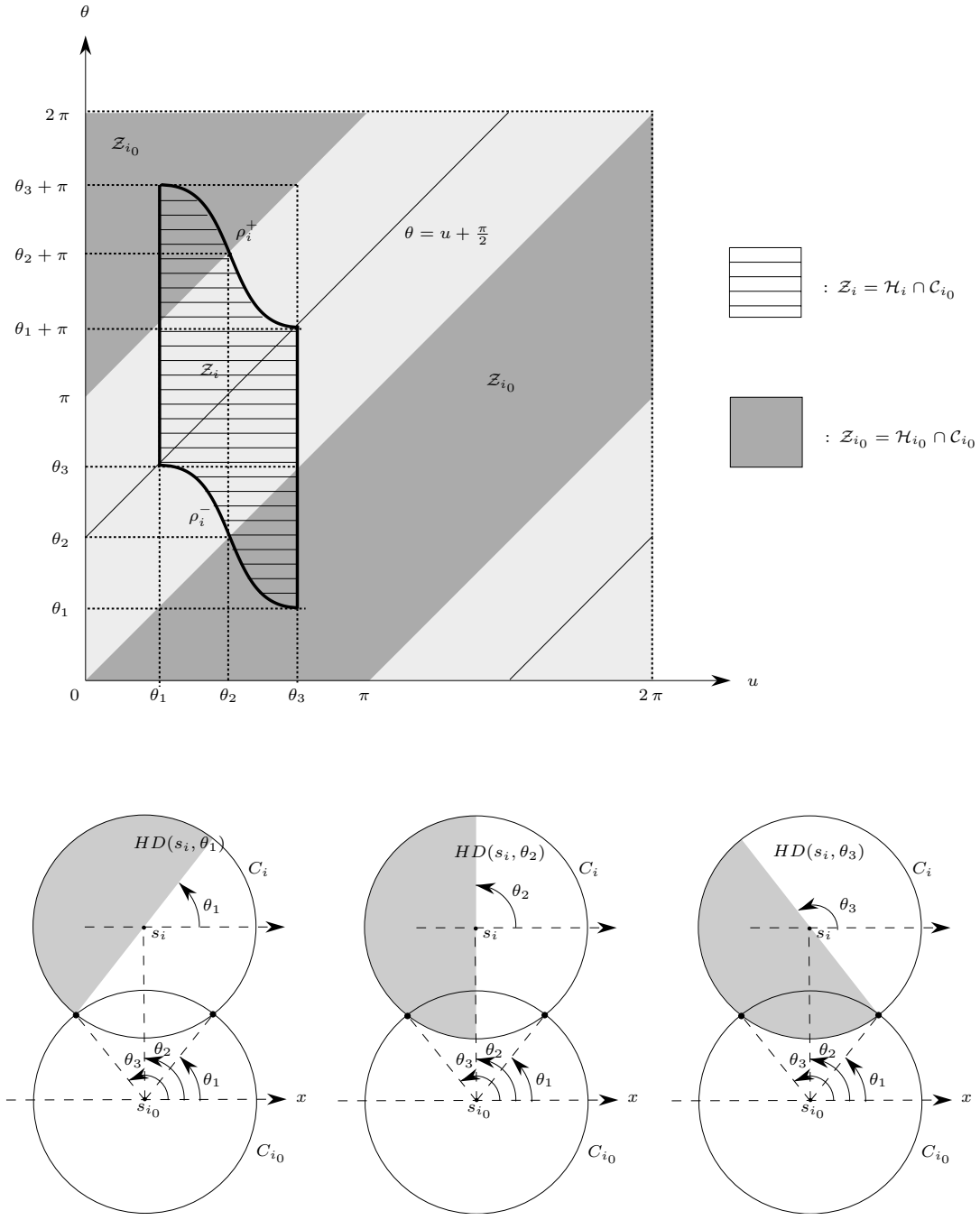


FIG. 4.1. *Example of* $\mathcal{Z}_i$ *for* $\|s_{i_0}s_i\| = \sqrt{2}\,R$ *and some corresponding critical positions of* $HD(s_i, \theta)$.

We now consider that equation with X equal to $\mathcal{H}$ or $int(\mathcal{H})$, and Y equal to the torus $\mathcal{C}_{i_0}$. Since $compl(p_{//\theta}(compl(\mathcal{C}_{i_0})))$ is the circle C_{i_0}, we get

$$compl(p_{//\theta}(compl(\mathcal{H} \cap \mathcal{C}_{i_0}))) = \mathcal{F} \cap C_{i_0} \quad \text{and}$$
$$compl(p_{//\theta}(compl(int(\mathcal{H}) \cap \mathcal{C}_{i_0}))) = int(\mathcal{F}) \cap C_{i_0}.$$

Since $\mathcal{H} = \cup_{i=1}^{n} \mathcal{H}_i$ and $\mathcal{Z}_i = \mathcal{H}_i \cap \mathcal{C}_{i_0}$ by definition, $\mathcal{H} \cap \mathcal{C}_{i_0} = \cup_{i=1}^{n} \mathcal{Z}_i$ and $int(\mathcal{H}) \cap \mathcal{C}_{i_0} = \cup_{i=1}^{n} (int(\mathcal{H}_i) \cap \mathcal{C}_{i_0})$. By the general position assumption, no two footholds lie at distance $2R$, thus for $i \neq i_0$, $int(\mathcal{H}_i) \cap \mathcal{C}_{i_0} = int(\mathcal{Z}_i)$.[2] As $int(\mathcal{H}_{i_0}) \cap \mathcal{C}_{i_0} = \emptyset$, we get $int(\mathcal{H}) \cap \mathcal{C}_{i_0} = \cup_{i \neq i_0} int(\mathcal{Z}_i)$. The study of the shape of $\mathcal{Z}_i$ will yield (see Lemma 4.8) that $\cup_{i \neq i_0} int(\mathcal{Z}_i) = int(\cup_{i \neq i_0} \mathcal{Z}_i)$. Therefore, $int(\mathcal{F}) \cap C_{i_0} = compl(p_{//\theta}(compl(int(\cup_{i \neq i_0} \mathcal{Z}_i)))) = int(compl(p_{//\theta}(compl(\cup_{i \neq i_0} \mathcal{Z}_i))))$ and $\mathcal{F} \cap C_{i_0} = compl(p_{//\theta}(compl(\cup_i \mathcal{Z}_i)))$. Using $C_{i_0} \cap \partial(\mathcal{F}) = [C_{i_0} \cap \mathcal{F}] \setminus [C_{i_0} \cap int(\mathcal{F})]$, we get the result. □

Thus, the contribution of C_{i_0} to $\partial(\mathcal{F})$ comes from the computation of $\cup_i \mathcal{Z}_i$ and $\cup_{i \neq i_0} \mathcal{Z}_i$.

Geometrically, $compl(p_{//\theta}(compl(\cup_i \mathcal{Z}_i)))$ is the vertical projection (along the θ-axis) of the largest vertical strip Σ_{i_0} included in $\cup_i \mathcal{Z}_i$ (see Figure 4.2). Similarly, $compl(p_{//\theta}(compl(\cup_{i \neq i_0} \mathcal{Z}_i)))$ is the projection of the largest vertical strip Σ'_{i_0} included in $\cup_{i \neq i_0} \mathcal{Z}_i$. Thus, $\partial(\mathcal{F}) \cap C_{i_0}$ is the vertical projection onto C_{i_0} of the vertical strip $\Sigma_{i_0} \setminus int(\Sigma'_{i_0})$.

In order to compute $\mathcal{F}$ efficiently, we need to compute the union of the regions $\mathcal{Z}_i$ efficiently. More precisely, we will show that the union of the regions $\mathcal{Z}_i$ can be computed in $O(k_{i_0} \log k_{i_0})$ time, where k_{i_0} is the number of helicoidal volumes $\mathcal{H}_i$ intersecting $\mathcal{C}_{i_0}$.

This is possible because the $\mathcal{Z}_i$ have special shapes that allow us to reduce the computation of their union to the computation of a small number of lower envelopes of curves drawn on $\mathcal{C}_{i_0}$, with the property that two of them intersect at most once. The geometric properties of the $\mathcal{Z}_i$ are discussed in section 4.2, and in section 4.3 we present and analyze the algorithm for constructing $\partial(\mathcal{F}) \cap C_{i_0}$.

4.2. Properties of the $\mathcal{Z}_i$. Here we study the regions $\mathcal{Z}_i = \mathcal{H}_i \cap \mathcal{C}_{i_0}$. Recall that we parameterize $\mathcal{C}_{i_0} = C_{i_0} \times S^1$ by (u, θ), where u and θ parameterize C_{i_0} and S^1, respectively. ($u = 0$ corresponds to the point of C_{i_0} with maximum x-coordinate.) Figures 4.1 and 4.2 show examples of such regions $\mathcal{Z}_i$. For convenience, we will use the vocabulary of the plane when describing objects on the torus $\mathcal{C}_{i_0}$. For instance, the curve drawn on the torus $\mathcal{C}_{i_0}$ with equation $a\,\theta + b\,u + c = 0$ will be called a line. The line $u = u_0$ will be called vertical and oriented according to increasing θ. Lower and upper will refer to this orientation. The discussion below considers only nonempty regions $\mathcal{Z}_i$ (such that $\|s_{i_0} s_i\| < 2R$).

First we introduce some notations. Let $HC_i(\theta)$ be the half-circle of the boundary of $HD(s_i, \theta)$, i.e., $HC_i(\theta) = C_i \cap HD(s_i, \theta)$. Let $r_i(\theta)$ be the spoke of C_i that makes an angle θ with the x-axis, i.e., $r_i(\theta) = \{s_i + \lambda \vec{u}_\theta \mid \lambda \in [0, R]\}$, where $\vec{u}_\theta$ is the unit

[2] Recall that *int* denotes the relative interior, thus $int(\mathcal{H}_i)$ is the interior of $\mathcal{H}_i$ in $\mathbb{R}^2 \times S^1$, but $int(\mathcal{Z}_i)$ denotes the interior of $\mathcal{Z}_i$ in $\mathcal{C}_{i_0}$.

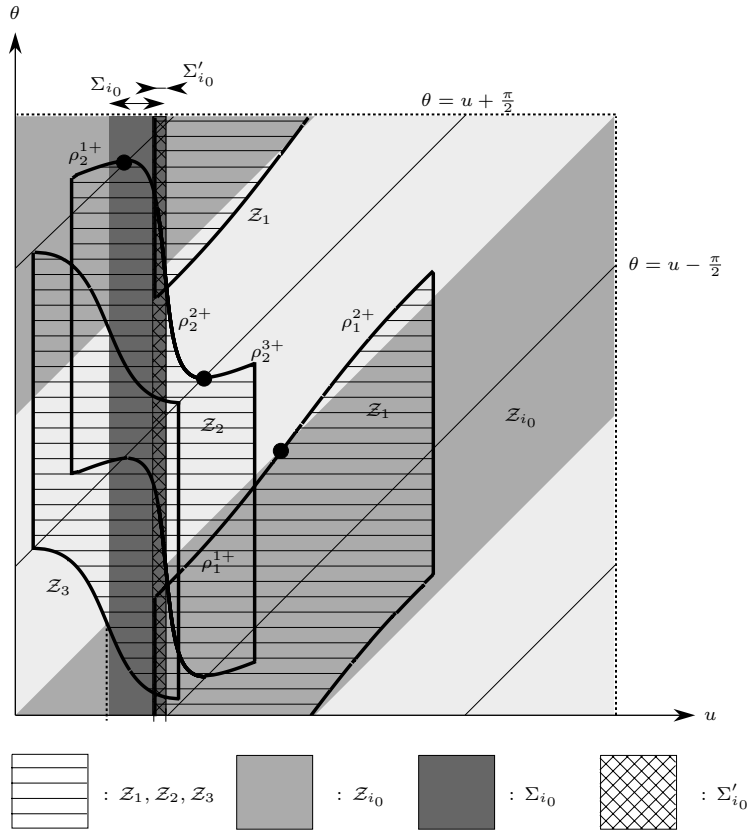


FIG. 4.2. *Contribution of C_{i_0} to $\partial(\mathcal{F})$ ($0 < \|s_1 s_{i_0}\| < R$, $R \leqslant \|s_2 s_{i_0}\| < \sqrt{2}\,R$, $\sqrt{2}\,R \leqslant \|s_3 s_{i_0}\| < 2R$).*

vector whose polar angle is θ. The boundary of $\mathcal{H}_i$ is composed of the following three patches:

$$\begin{aligned}\mathcal{T}_i &= \{(HC_i(\theta), \theta) \in \mathbb{R}^2 \times S^1\},\\ \mathcal{R}_i^+ &= \{(r_i(\theta), \theta) \in \mathbb{R}^2 \times S^1\},\\ \mathcal{R}_i^- &= \{(r_i(\theta + \pi), \theta) \in \mathbb{R}^2 \times S^1\}.\end{aligned}$$

Let ρ_i^- and ρ_i^+ denote the curves $\mathcal{R}_i^- \cap \mathcal{C}_{i_0}$ and $\mathcal{R}_i^+ \cap \mathcal{C}_{i_0}$, respectively. Since $\mathcal{R}_i^-$ and $\mathcal{R}_i^+$ are translated copies of one another, i.e., $\mathcal{R}_i^- = \mathcal{R}_i^+ \pm (0, 0, \pi)$, we have the following lemmas.

LEMMA 4.2. *ρ_i^- and ρ_i^+ are translated copies of one another, i.e.,*

$$\rho_i^+ = \{(u, \theta) \in S^1 \times S^1 \mid (u, \theta - \pi) \in \rho_i^-\} = \{(u, \theta) \in S^1 \times S^1 \mid (u, \theta + \pi) \in \rho_i^-\}.$$

LEMMA 4.3. *The curves $\rho_i^\pm$ are monotone in u.*

Proof. Assume for a contradiction that a curve $\rho_i^\pm$ is not monotone in u. Then there exists u and $\theta \neq \theta'$ in S^1 such that (u, θ) and (u, θ') parameterize points of $\rho_i^\pm$. By the definition of $\mathcal{R}_i^\pm$, it then follows that the point $U \in C_{i_0}$ parameterized by u belongs to the two spokes $r_i(\theta)$ (or $r_i(\theta+\pi)$) and $r_i(\theta')$ (or $r_i(\theta'+\pi)$). The intersection between any two of these spokes is exactly s_i. Thus, $U = s_i$, which contradicts (since

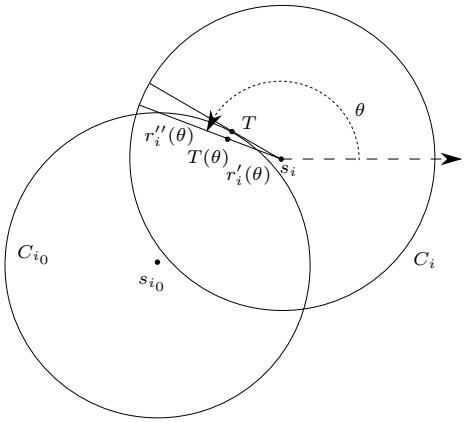


FIG. 4.3. *For the definition of $r_i'(\theta)$ and $r_i''(\theta)$.*

$U \in C_{i_0}$) the general position assumption saying that the distance between s_i and s_{i_0} is not R. □

LEMMA 4.4. *The region $\mathcal{Z}_{i_0}$ is the subset of $\mathcal{C}_{i_0}$ parameterized by $\{(u,\theta) \in S^1 \times S^1 \mid \theta \leqslant u \leqslant \theta + \pi\}$ (shown in grey in Figures* 4.1 *and* 4.2*).*

Proof. For any $\theta \in S^1$, the intersection between $\mathcal{H}_{i_0}$ and the "horizontal plane" Π_θ is the half-disk $HD(s_{i_0}, \theta)$. Similarly, the intersection between $\mathcal{C}_{i_0}$ and that plane is C_{i_0}. Thus, the intersection between $\mathcal{Z}_{i_0}$ and Π_θ is $HC_{i_0}(\theta)$, which is parameterized on C_{i_0} by $\{u \in S^1 \mid \theta \leqslant u \leqslant \theta + \pi\}$. That intersection is actually on the plane Π_θ and is therefore parameterized on $\mathcal{C}_{i_0}$ by $\{(u,\theta) \in S^1 \times S^1 \mid \theta \leqslant u \leqslant \theta + \pi\}$. □

PROPOSITION 4.5. *$\mathcal{Z}_i$ is a connected region bounded from below by ρ_i^- and from above by ρ_i^+, i.e., $\mathcal{Z}_i = \{(u,\theta) \in S^1 \times S^1 \mid \exists x \in [0,\pi], (u, \theta - x) \in \rho_i^-, (u, \theta - x + \pi) \in \rho_i^+\}$ (see Figures* 4.1 and 4.2*).*

Proof. By cutting $\mathcal{C}_{i_0}$ and $\mathcal{H}_i$ by the "horizontal plane" Π_θ, we get that a point parameterized by (u,θ) on $\mathcal{C}_{i_0}$ belongs to $\mathcal{H}_i$ if and only if the point U parameterized by u on C_{i_0} belongs to $HD(s_i,\theta)$. Since $HD(s_i,\theta)$ can be seen as the union of the spokes $\{r_i(\theta+\gamma) \mid \gamma \in [0,\pi]\}$, $(u,\theta) \in \mathcal{Z}_i$ if and only if there exists $\gamma \in [0,\pi]$ such that $U \in r_i(\theta+\gamma)$, or, equivalently, $U \in r_i(\theta - x + \pi)$ with $x = \pi - \gamma \in [0,\pi]$. Since $\mathcal{R}_i^- = \{(r_i(\theta - x + \pi), \theta - x) \mid \theta - x \in S^1\}$, it follows from $U \in r_i(\theta - x + \pi)$ that the point of $\mathcal{C}_{i_0}$ parameterized by $(u, \theta - x)$ belongs to $\mathcal{R}_i^-$ and thus to $\rho_i^- = \mathcal{R}_i^- \cap \mathcal{C}_{i_0}$. From Lemma 4.2, we get that the point parameterized by $(u, \theta - x + \pi)$ belongs to ρ_i^+. Therefore, $\mathcal{Z}_i$ is a connected region bounded from below by ρ_i^- and from above by ρ_i^+. □

We want to compute the union of the $\mathcal{Z}_i$ by computing the "lower envelope"[3] of the lower edges ρ_i^-, and the "upper envelope" of the upper edges ρ_i^+. Unfortunately it is impossible to do so because some upper edges ρ_i^+ may be "below" or intersect some lower edges ρ_j^-. However, we can subdivide the regions $\mathcal{Z}_i$ into blocks $\mathcal{Z}_i^k$, $k \in \mathcal{K}$, and separate these blocks into two sets Ω_1 and Ω_2 such that the union of the $\mathcal{Z}_i^k$ in Ω_1 (respectively, Ω_2) is the region bounded from above by the upper envelope of the upper edges of the $\mathcal{Z}_i^k \in \Omega_1$ and bounded from below by the lower envelope of the lower edges of the $\mathcal{Z}_i^k \in \Omega_1$ (respectively, Ω_2). Such property can be realized by showing that all the upper edges of the $\mathcal{Z}_i^k \in \Omega_1$ belong to the strip $\{(u,\theta) \in S^1 \times [u + \frac{\pi}{2}, u + \frac{3\pi}{2}]\}$ and

[3]Note that the lower and upper envelopes of curves in $S^1 \times S^1$ are not actually defined.

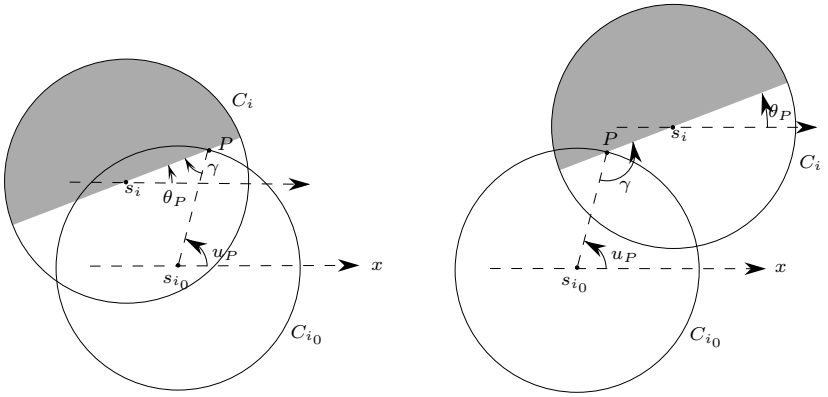


FIG. 4.4. *For the proof of Proposition* 4.6.

all the lower edges of the $\mathcal{Z}_i^k \in \Omega_1$ belong to the strip $\{(u,\theta) \in S^1 \times [u - \frac{\pi}{2}, u + \frac{\pi}{2}]\}$. (A similar property is shown for Ω_2.) Note that the upper and lower envelopes are then defined since they are considered in $S^1 \times \mathbb{R}$.

We subdivide $\mathcal{Z}_i$ into blocks $\mathcal{Z}_i^k$ when $R < \|s_{i_0}s_i\| < \sqrt{2}\,R$. That subdivision is performed such that the upper and lower edges of the $\mathcal{Z}_i^k$ are θ-monotone. Recall that the upper edge ρ_i^+ of $\mathcal{Z}_i$ is the intersection of $\mathcal{R}_i^+ = \{(r_i(\theta),\theta) \mid \theta \in S^1\}$ and $\mathcal{C}_{i_0}$. The spoke $r_i(\theta)$ intersects C_{i_0} twice (for some θ) when $R < \|s_{i_0}s_i\| < \sqrt{2}\,R$, which implies that ρ_i^+ is not θ-monotone. We cut the spoke $r_i(\theta)$ into two pieces such that each piece intersects C_{i_0} at most once. Let T be the intersection point between C_{i_0} and on one of the two lines passing through s_i and tangent to C_{i_0} (see Figure 4.3). Let $T(\theta)$ be the point on $r_i(\theta)$ at distance $\|s_iT\|$ from s_i. Cutting $r_i(\theta)$ at $T(\theta)$ defines two subspokes $r_i'(\theta)$ and $r_i''(\theta)$ that intersect C_{i_0} in at most one point each; without loss of generality, let $r_i'(\theta)$ denote the subspoke joining s_i to $T(\theta)$. The set of $\theta \in S^1$ for which $r_i'(\theta)$ intersects C_{i_0} is clearly connected but the set of $\theta \in S^1$ for which $r_i''(\theta)$ intersects C_{i_0} consists of two connected components. We denote by ρ_i^{2+} the intersection $\{(r_i'(\theta),\theta) \mid \theta \in S^1\} \cap \mathcal{C}_{i_0}$ and by ρ_i^{1+} and ρ_i^{3+} the two connected components of the intersection $\{(r_i''(\theta),\theta) \mid \theta \in S^1\} \cap \mathcal{C}_{i_0}$ (see Figure 4.2). Since $r_i'(\theta)$ and $r_i''(\theta)$ intersect C_{i_0} at most once for any $\theta \in S^1$, the curves ρ_i^{1+}, ρ_i^{2+}, and ρ_i^{3+} are θ-monotone. The lower edges ρ_i^{k-}, $k = 1,2,3$, can be defined similarly or in a simpler way as the translated copies of ρ_i^{k+}, $k = 1,2,3$, i.e., $\rho_i^{k-} = \{(u,\theta) \in S^1 \times S^1 \mid (u,\theta+\pi) \in \rho_i^{+k}\}$. We denote by $\mathcal{Z}_i^k$, $k = 1,2,3$, the subset of $\mathcal{Z}_i$ bounded from above by ρ_i^{k+} and from below by ρ_i^{k-}.

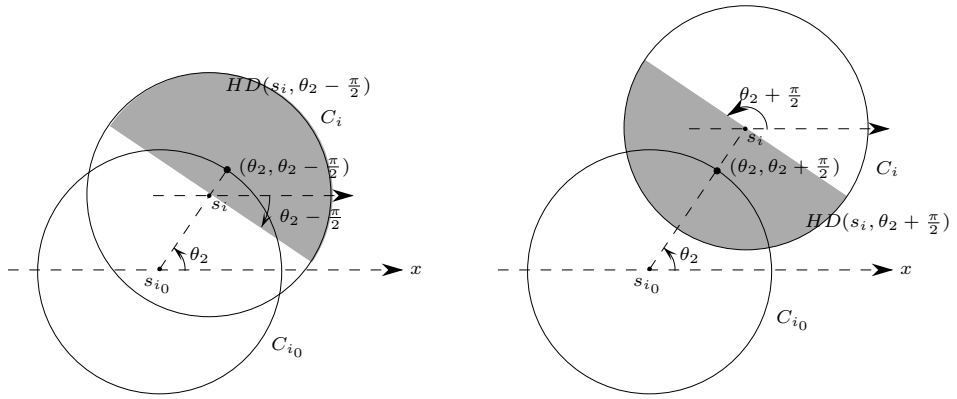
We can now prove the following proposition that will allow us to compute the union of the $\mathcal{Z}_i$ by computing the upper and lower envelopes of their upper and lower edges.

PROPOSITION 4.6. *If $0 \leqslant \|s_{i_0}s_i\| < R$, the line $\theta = u - \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i$, and the lines $\theta = u \pm \frac{\pi}{2}$ properly intersect neither ρ_i^+ nor ρ_i^-.*

If $R < \|s_{i_0}s_i\| < \sqrt{2}\,R$, the line $\theta = u + \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i^2$, and the line $\theta = u - \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i^1$ and $\mathcal{Z}_i^3$. Furthermore, the lines $\theta = u \pm \frac{\pi}{2}$ properly intersect none of the edges ρ_i^{1+}, ρ_i^{1-}, ρ_i^{2+}, ρ_i^{2-}, ρ_i^{3+}, and ρ_i^{3-}.

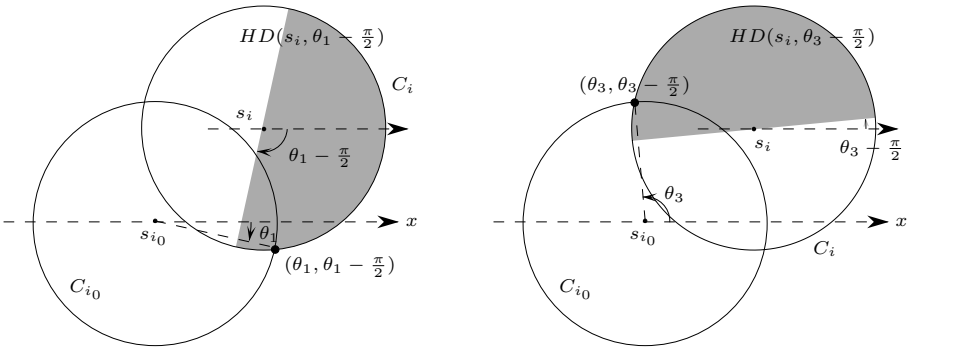
If $\sqrt{2}\,R \leqslant \|s_{i_0}s_i\| < 2R$, the line $\theta = u + \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i$, and the lines $\theta = u \pm \frac{\pi}{2}$ properly intersect neither ρ_i^+ nor ρ_i^-.

Proof. Let (u_P,θ_P) parameterize a point of a curve ρ_i. Let P denote the point



(a): $0 < \|s_{i_0} s_i\| < R$, $(\theta_2, \theta_2 - \frac{\pi}{2}) \in \mathcal{Z}_i$

(b): $R < \|s_{i_0} s_i\| < 2R$, $(\theta_2, \theta_2 + \frac{\pi}{2}) \in \mathcal{Z}_i$



(c): $R < \|s_{i_0} s_i\| < \sqrt{2}\, R$, $(\theta_1, \theta_1 - \frac{\pi}{2}) \in \mathcal{Z}_i^1$

(d): $R < \|s_{i_0} s_i\| < \sqrt{2}\, R$, $(\theta_3, \theta_3 - \frac{\pi}{2}) \in \mathcal{Z}_i^3$

FIG. 4.5. *For the proof of Proposition* 4.6: *section of* $\mathcal{H}_i$ *and* $\mathcal{C}_{i_0}$ *by the "planes"* $\Pi_{\theta_2 - \frac{\pi}{2}}$, $\Pi_{\theta_2 + \frac{\pi}{2}}$, $\Pi_{\theta_1 - \frac{\pi}{2}}$, *and* $\Pi_{\theta_3 - \frac{\pi}{2}}$, *respectively.*

of C_{i_0} with parameter u_P and $\gamma = \angle(\overrightarrow{Ps_{i_0}}, \overrightarrow{Ps_i})$ $[2\pi]$ (see Figure 4.4). One can easily show that $\gamma = \theta_P - u_P[\pi]$. We prove that $\gamma \neq \frac{\pi}{2}$ $[\pi]$, except possibly when (u_P, θ_P) is an endpoint of ρ_i (or ρ_i^k when $R < \|s_{i_0} s_i\| < \sqrt{2}\, R$), which implies, since $\gamma = \theta_P - u_P[\pi]$, that the lines $\theta = u \pm \frac{\pi}{2}$ intersect neither ρ_i^+ nor ρ_i^- (respectively, neither ρ_i^{k+} nor ρ_i^{k-}), except possibly at their endpoints.

Case 1. $0 \leqslant \|s_{i_0} s_i\| < R$. Since s_i belongs to the disk of radius R centered at s_{i_0}, $\gamma \in (-\frac{\pi}{2}, \frac{\pi}{2})$ for any $P \in C_{i_0}$ (see Figure 4.4). Thus, the lines $\theta = u \pm \frac{\pi}{2}$ properly intersect neither ρ_i^+ nor ρ_i^-. Finally, the point of $\mathcal{C}_{i_0}$ $(\theta_2, \theta_2 - \frac{\pi}{2})$, where $\theta_2 = \angle(\vec{x}, \overrightarrow{s_{i_0} s_i})$ $[2\pi]$, belongs to the line $\theta = u - \frac{\pi}{2}$ and also to the relative interior of $\mathcal{Z}_i$ since it belongs to the interior of $\mathcal{H}_i$ (see Figure 4.5a). Therefore, the line $\theta = u - \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i$.

Case 2. $R < \|s_{i_0} s_i\| < \sqrt{2}\, R$. Let (u_{P_1}, θ_{P_1}) parameterize the point connecting ρ_i^{1+} and ρ_i^{2+} and (u_{P_2}, θ_{P_2}) parameterize the point connecting ρ_i^{2+} and ρ_i^{3+}. Let P_1 and P_2 denote the points of C_{i_0} parameterized by u_{P_1} and u_{P_2}, respectively. According to the construction of ρ_i^{1+}, ρ_i^{2+}, and ρ_i^{3+}, the tangent lines to C_{i_0} at P_1 and P_2 pass through s_i. At most two tangent lines to C_{i_0} pass through s_i, thus P_1 and P_2 are the only points of C_{i_0} where $\gamma = \frac{\pi}{2}$ $[\pi]$. Since ρ_i^+ is u-monotone by Lemma 4.3,

(u_{P_1}, θ_{P_1}) and (u_{P_2}, θ_{P_2}) are the only points of ρ_i^+ where $\gamma = \frac{\pi}{2}$ $[\pi]$. Therefore, the lines $\theta = u \pm \frac{\pi}{2}$ do not properly intersect ρ_i^{k+}, $k = 1, 2, 3$. Similarly, the lines $\theta = u \pm \frac{\pi}{2}$ do not properly intersect ρ_i^{k-}, $k = 1, 2, 3$.

Let θ_1 and θ_3 be the parameters on C_{i_0} of the intersection points between C_{i_0} and C_i (see Figures 4.5c and d); to differentiate θ_1 from θ_3, assume without loss of generality that, for any $\varepsilon > 0$ small enough, the points of C_{i_0} parameterized by $\theta_1 + \varepsilon$ and $\theta_3 - \varepsilon$ are in the disk of radius R centered at s_i. Then, the points $(\theta_1, \theta_1 - \frac{\pi}{2})$ and $(\theta_3, \theta_3 - \frac{\pi}{2})$ of $\mathcal{C}_{i_0}$ belong to $\mathcal{Z}_i^1$ and $\mathcal{Z}_i^3$ (or to $\mathcal{Z}_i^3$ and $\mathcal{Z}_i^1$), respectively (see Figures 4.5c and d). However, these points do not belong to the relative interior of $\mathcal{Z}_i^1$ and $\mathcal{Z}_i^3$ (because they lie on the border of $HD(s_i, \theta_1 - \frac{\pi}{2})$ and $HD(s_i, \theta_3 - \frac{\pi}{2})$). Nevertheless, there clearly exists $\varepsilon > 0$ small enough such that the point parameterized by $\theta_1 + \varepsilon$ (respectively, $\theta_3 - \varepsilon$) on C_{i_0} belongs to the interior of the half-disk $HD(s_i, \theta_1 - \frac{\pi}{2} + \varepsilon)$ (respectively, $HD(s_i, \theta_3 - \frac{\pi}{2} - \varepsilon)$). Thus, the points $(\theta_1 + \varepsilon, \theta_1 + \varepsilon - \frac{\pi}{2})$ and $(\theta_3 - \varepsilon, \theta_3 - \varepsilon - \frac{\pi}{2})$ of $\mathcal{C}_{i_0}$ belong to the relative interior of $\mathcal{Z}_i^1$ and $\mathcal{Z}_i^3$, respectively. Therefore, the line $\theta = u - \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i^1$ and $\mathcal{Z}_i^3$.

On the other hand, $(\theta_2, \theta_2 + \frac{\pi}{2})$ (where $\theta_2 = \angle(\vec{x}, \overrightarrow{s_{i_0} s_i})$ $[2\pi]$) belongs to relative interior of $\mathcal{Z}_i^2$ because the point of C_{i_0} parameterized by θ_2 belongs to the relative interior of the subspoke $r_i'(\theta_2 + \pi)$ (see Figure 4.5b) which belongs to interior of $HD(s_i, \theta_2 + \frac{\pi}{2})$. Therefore, the line $\theta = u + \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i^2$.

Case 3. $\sqrt{2}\, R \leqslant \|s_{i_0} s_i\| < 2R$. Since $r_i(\theta)$ intersects C_{i_0} at most once, $\gamma \in [\frac{\pi}{2}, \frac{3\pi}{2}]$ (see Figure 4.4). Moreover, $\gamma = \frac{\pi}{2}$ $[\pi]$ only when $\|s_{i_0} s_i\| = \sqrt{2}\, R$, but then P is at distance R from s_i which implies that (u_P, θ_P) is an endpoint of ρ_i. Thus, the lines $\theta = u \pm \frac{\pi}{2}$ intersect neither ρ_i^+ nor ρ_i^-, except possibly at their endpoints. Finally, the point $(\theta_2, \theta_2 + \frac{\pi}{2})$ of $\mathcal{C}_{i_0}$ (where $\theta_2 = \angle(\vec{x}, \overrightarrow{s_{i_0} s_i})$ $[2\pi]$) belongs to the line $\theta = u + \frac{\pi}{2}$ and also to the relative interior of $\mathcal{Z}_i$ (see Figures 4.5b and 4.1). Therefore, the line $\theta = u + \frac{\pi}{2}$ properly intersects $\mathcal{Z}_i$. □

By Proposition 4.6, we can compute the union $\cup_{i \neq i_0} \mathcal{Z}_i$ by separating the $\mathcal{Z}_i$, $\mathcal{Z}_i^k$ into two sets Ω_1 and Ω_2 (where $\mathcal{Z}_i$, $\mathcal{Z}_i^k$ belongs to Ω_1 if and only if ρ_i^+, ρ_i^{k+} belongs to the strip $\{(u, \theta) \in S^1 \times [u + \frac{\pi}{2}, u + \frac{3\pi}{2}]\}$ and ρ_i^-, ρ_i^{k-} belongs to the strip $\{(u, \theta) \in S^1 \times [u - \frac{\pi}{2}, u + \frac{\pi}{2}]\}$) and by computing the union of the $\mathcal{Z}_i$, $\mathcal{Z}_i^k$ in Ω_1 (respectively, Ω_2) by computing the upper envelope of their upper edges and the lower envelope of their lower edges. In order to compute efficiently these upper and lower envelopes, we show that the curves ρ_i^+, ρ_i^-, ρ_i^{k+}, and ρ_i^{k-} intersect each other at most once. However, we need for that purpose to split the regions $\mathcal{Z}_i$ when $0 < \|s_{i_0} s_i\| < R$ into two blocks $\mathcal{Z}_i^1$ and $\mathcal{Z}_i^2$ separated by the vertical line $u = \theta_2 = \angle(\vec{x}, \overrightarrow{s_{i_0} s_i})$; it also remains to split the θ-interval (or the u-interval) over which ρ_i is defined into two intervals of equal length over which $\rho_i^{1\pm}$ and $\rho_i^{2\pm}$ are defined (see Figure 4.2). Note that Proposition 4.6 still holds if we replace (when $0 < \|s_{i_0} s_i\| < R$) $\mathcal{Z}_i$ by $\mathcal{Z}_i^k$ and $\rho_i^\pm$ by $\rho_i^{k\pm}$, $k = 1, 2$.

For consistency, we split $\mathcal{Z}_{i_0}$ into two blocks $\mathcal{Z}_{i_0}^1$ and $\mathcal{Z}_{i_0}^2$ separated by a vertical line (chosen arbitrarily, say $u = \pi$). Also for consistency, the curves $\rho_i^\pm$ when $\sqrt{2}\, R \leqslant \|s_{i_0} s_i\| < 2R$ are occasionally denoted in what follows by $\rho_i^{1\pm}$.

LEMMA 4.7. *Let ρ_i' and ρ_j' be some connected portions of $\rho_i^\pm$ and $\rho_j^\pm$, respectively $(i \neq j)$. If ρ_i' or ρ_j' is monotone in θ and defined over a θ-interval smaller than π, then ρ_i' and ρ_j' intersect at most once.*

Proof. Let (u_I, θ_I) be a point of intersection between ρ_i' and ρ_j' and I be the point of the circle C_{i_0} with parameter u_I. Since ρ_i' is a portion of the intersection between $\mathcal{C}_{i_0}$ and $\mathcal{R}_i^\pm$, I is a point of intersection between C_{i_0} and the diameter of $HD(s_i, \theta_I)$. Therefore, the line passing through s_i and I has slope θ_I.

By applying the same argument to ρ'_j, we obtain that s_i and s_j belong to the same straight line of slope θ_I. Therefore, if ρ'_i and ρ'_j intersect twice, at (u_I, θ_I) and (u_J, θ_J), then $\theta_I = \theta_J[\pi]$. It follows, if ρ'_i or ρ'_j is defined over a θ-interval smaller than π, that $\theta_I = \theta_J[2\pi]$. Furthermore, if ρ'_i or ρ'_j is monotone in θ, then (u_I, θ_I) and (u_J, θ_I) are equal. □

LEMMA 4.8. *For all i, j, $int(\mathcal{Z}_i) \cup int(\mathcal{Z}_j) = int(\mathcal{Z}_i \cup \mathcal{Z}_j)$.*

Proof. We assume that $i \neq j$ because otherwise the result is trivial. One can easily show that $int(\mathcal{Z}_i) \cup int(\mathcal{Z}_j) \neq int(\mathcal{Z}_i \cup \mathcal{Z}_j)$ only if the boundaries of $\mathcal{Z}_i$ and $\mathcal{Z}_j$ partially coincide, i.e., the dimension of $\partial(\mathcal{Z}_i) \cap \partial(\mathcal{Z}_j)$ is 1.

By Proposition 4.5, $\partial(\mathcal{Z}_i)$ consists of the edges ρ_i^+ and ρ_i^- and of two vertical line segments joining the endpoints of ρ_i^+ and ρ_i^- when these endpoints exist (which is the case when $i \neq i_0$). Moreover, these vertical line segments are clearly supported by the vertical lines $u = \theta_1$ and $u = \theta_3$, where θ_1 and θ_3 parameterize on C_{i_0} the points of intersection between C_{i_0} and C_i (see Figure 4.1).

By Lemma 4.7, the edges $\rho_i^\pm$ and $\rho_j^\pm$ do not partially coincide. By the general position assumption, no three distinct circles C_{i_0}, C_i, and C_j have a common intersection point. Thus, for any $i \neq j$, $C_{i_0} \cap C_i$ and $C_{i_0} \cap C_j$ are disjoint. Therefore, the vertical lines $\partial(\mathcal{Z}_i) \setminus \{\rho_i^+, \rho_i^-\}$ and $\partial(\mathcal{Z}_j) \setminus \{\rho_j^+, \rho_j^-\}$ do not partially coincide. Finally, since $\rho_i^\pm$ is nowhere partially supported by a vertical line by Lemma 4.3, $\rho_i^\pm$ and the vertical lines $\partial(\mathcal{Z}_j) \setminus \{\rho_j^+, \rho_j^-\}$ do not partially coincide. □

PROPOSITION 4.9. *Any two curves among the curves $\rho_i^{k\pm}$ intersect at most once (where $k \in \{1, 2\}$ if $0 \leqslant \|s_{i_0} s_i\| < R$, $k \in \{1, 2, 3\}$ if $R < \|s_{i_0} s_i\| < \sqrt{2}\, R$, and $k = 1$ if $\sqrt{2}\, R \leqslant \|s_{i_0} s_i\| < 2R$).*

Proof. By Lemma 4.7, it is sufficient to prove that all the curves $\rho_i^{k\pm}$, $i \neq i_0$ are monotone in θ and defined over θ-intervals smaller than π. Indeed, the curves $\rho_{i_0}^{1+}$, $\rho_{i_0}^{1-}$, $\rho_{i_0}^{2+}$, and $\rho_{i_0}^{2-}$ clearly do not pairwise intersect more than once, by Lemma 4.4.

If $0 < \|s_{i_0} s_i\| < R$, any spoke of C_i intersects C_{i_0} at most once. Hence, $\rho_i^\pm$ is monotone in θ. $\rho_i^\pm$ is defined over a θ-interval greater than π but smaller than 2π. Since we have split that interval in two equal parts, $\rho_i^{1\pm}$ and $\rho_i^{2\pm}$ are defined over a θ-interval smaller than π (see $\mathcal{Z}_1$ in Figure 4.2).

If $R < \|s_{i_0} s_i\| < \sqrt{2}\, R$, the θ-interval where $r_i(\theta)$ (or $r_i(\theta + \pi)$) intersects C_{i_0} is smaller than π, which implies that ρ_i is defined over a θ-interval smaller than π. The curves ρ_i^{k+}, $k = 1, 2, 3$, are defined as the connected components of $\{(r'_i(\theta), \theta) \mid \theta \in S^1\} \cap \mathcal{C}_{i_0}$ and $\{(r''_i(\theta), \theta) \mid \theta \in S^1\} \cap \mathcal{C}_{i_0}$. Since the subspokes $r'_i(\theta)$ and $r''_i(\theta)$ intersect C_{i_0} at most once for any $\theta \in S^1$, the curves ρ_i^{k+}, $k = 1, 2, 3$, are θ-monotone.

If $\sqrt{2}\, R \leqslant \|s_{i_0} s_i\| < 2R$, $r_i(\theta)$ (and also $r_i(\theta + \pi)$) intersects C_{i_0} in at most one point, which proves that ρ_i is monotone in θ. Furthermore, the θ-interval where ρ_i is defined is smaller than π because the θ-interval where $r_i(\theta)$ (or $r_i(\theta + \pi)$) intersects C_{i_0} is smaller than π. □

4.3. Construction of $\partial(\mathcal{F}) \cap C_{i_0}$. We first show how to compute $\cup_i \mathcal{Z}_i$. Let Ω_1 and Ω_2 be the following sets of $\mathcal{Z}_i^k$:

$$\Omega_1 = \{\mathcal{Z}_i \mid \sqrt{2}\, R \leqslant \|s_{i_0} s_i\| < 2R\} \cup \{\mathcal{Z}_i^2 \mid R < \|s_{i_0} s_i\| < \sqrt{2}\, R\},$$
$$\Omega_2 = \{\mathcal{Z}_i^1, \mathcal{Z}_i^2 \mid 0 \leqslant \|s_{i_0} s_i\| < R\} \cup \{\mathcal{Z}_i^1, \mathcal{Z}_i^3 \mid R < \|s_{i_0} s_i\| < \sqrt{2}\, R\}.$$

By Proposition 4.6, the line $\theta = u + \frac{\pi}{2}$ properly intersects all the $\mathcal{Z}_i^k \in \Omega_1$ but the lines $\theta = u \pm \frac{\pi}{2}$ properly intersect none of their upper and lower edges ρ_i^{k+} and ρ_i^{k-}. Thus, the regions $\mathcal{Z}_i^k \in \Omega_1$ can be seen as regions of $\{(u, \theta) \in S^1 \times [u - \frac{\pi}{2}, u + \frac{3\pi}{2}]\}$

such that all their upper edges ρ_i^{k+} lie in $\{(u,\theta) \in S^1 \times [u+\frac{\pi}{2}, u+\frac{3\pi}{2}]\}$ and all their lower edges ρ_i^{k-} lie in $\{(u,\theta) \in S^1 \times [u-\frac{\pi}{2}, u+\frac{\pi}{2}]\}$. Therefore, the union of the $\mathcal{Z}_i^k \in \Omega_1$ is the region of $\{(u,\theta) \in S^1 \times [u-\frac{\pi}{2}, u+\frac{3\pi}{2}]\}$ bounded from above by the upper envelope of their ρ_i^{k+} and bounded from below by the lower envelope of their ρ_i^{k-}. Similarly, the union of the $\mathcal{Z}_i^k \in \Omega_2$ is the region of $\{(u,\theta) \in S^1 \times [u-\frac{3\pi}{2}, u+\frac{\pi}{2}]\}$ bounded from above by the upper envelope of the ρ_i^{k+} and bounded from below by the lower envelope of the ρ_i^{k-}.

The union of Ω_1 and Ω_2, which is $\cup_i \mathcal{Z}_i$, can be achieved by computing, on one hand, the intersection between the upper edge chain of $\cup_{\mathcal{Z}_i^k \in \Omega_1} \mathcal{Z}_i^k$ with the lower edge chain of $\cup_{\mathcal{Z}_i^k \in \Omega_2} \mathcal{Z}_i^k$ (which both belong to $\{(u,\theta) \in S^1 \times S^1 \mid \theta \in [u+\frac{\pi}{2}, u+\frac{3\pi}{2}]\}$), and on the other hand, the intersection between the upper edge chain of $\cup_{\mathcal{Z}_i^k \in \Omega_2} \mathcal{Z}_i^k$ with the lower edge chain of $\cup_{\mathcal{Z}_i^k \in \Omega_1} \mathcal{Z}_i^k$ (which both belong to $\{(u,\theta) \in S^1 \times S^1 \mid \theta \in [u-\frac{\pi}{2}, u+\frac{\pi}{2}]\}$). These intersections can simply be performed by following the two edge chains for u from 0 to 2π, since they are monotone in u by Lemma 4.3.

Let us analyze the complexity of the above construction. The k_{i_0} helicoidal volumes $\mathcal{H}_i$ that intersect $\mathcal{C}_{i_0}$ can be found in $O(k_{i_0})$ amortized time once the Delaunay triangulation of the footholds has been computed, which can be done in $O(n \log n)$ time [6, 17]. By Proposition 4.9, the upper and lower envelopes can be computed in $O(k_{i_0} \log k_{i_0})$ time using $O(k_{i_0}\alpha(k_{i_0}))$ space where α is the pseudo inverse of Ackerman's function [7]. Also by Proposition 4.9, the union of Ω_1 and Ω_2 can be done in linear time in the size of the edge chains, that is $O(k_{i_0}\alpha(k_{i_0}))$ time. Thus, we can compute $\cup_i \mathcal{Z}_i$ in $O(k_{i_0} \log k_{i_0})$ time using $O(k_{i_0}\alpha(k_{i_0}))$ space after $O(n \log n)$ preprocessing time. We can compute $\cup_{i \neq i_0} \mathcal{Z}_i$ similarly by removing $\mathcal{Z}_{i_0}^1$ and $\mathcal{Z}_{i_0}^2$ from Ω_2.

The contribution of C_{i_0} to $\partial(\mathcal{F})$ is, according to Proposition 4.1, $C_{i_0} \cap \partial(\mathcal{F})$ $= \mathit{compl}(p_{/\!/\theta}(\mathit{compl}(\cup_i \mathcal{Z}_i))) \setminus \mathit{int}(\mathit{compl}(p_{/\!/\theta}(\mathit{compl}(\cup_{i \neq i_0} \mathcal{Z}_i))))$. By Remark 3.7, $\mathit{compl}(p_{/\!/\theta}(\mathit{compl}(\cup_i \mathcal{Z}_i)))$ and $\mathit{compl}(p_{/\!/\theta}(\mathit{compl}(\cup_{i \neq i_0} \mathcal{Z}_i)))$ are the projections onto C_{i_0} of the largest vertical strips Σ_{i_0} and Σ'_{i_0} included in $\cup_i \mathcal{Z}_i$ and $\cup_{i \neq i_0} \mathcal{Z}_i$, respectively (see Figure 4.2). These projections are easily computed because the edges of $\cup_i \mathcal{Z}_i$ and $\cup_{i \neq i_0} \mathcal{Z}_i$ are monotone with respect to u (Lemma 4.3). These projections, and therefore the computation of $C_{i_0} \cap \partial(\mathcal{F})$, can thus be done in linear time and space in the size of $\cup_i \mathcal{Z}_i$ and $\cup_{i \neq i_0} \mathcal{Z}_i$, that is $O(k_{i_0}\alpha(k_{i_0}))$.

Moreover, we label an arc of $\partial(\mathcal{F})$ either by i if the arc belongs to the circle C_i or by (i,j) if the arc belongs to the straight line segment $[s_i, s_j]$. The labels of the edges of $\partial(\mathcal{F})$ incident to C_{i_0} can be found as follows, without increasing the complexity. An arc of $\partial(\mathcal{F}) \cap C_{i_0}$ corresponds to a vertical strip $\Sigma_{i_0} \setminus \Sigma'_{i_0}$. An endpoint P of such an arc is the projection of a vertical edge or the projection of a point of intersection between two curved edges. In the first case, P is the intersection of C_{i_0} with some C_i and in the second case, P is the intersection of C_{i_0} with some line segment $[s_i, s_j]$. By the general position assumption, among the circles $C_1, \ldots, C_n$ and the line segments joining two footholds, the intersection between three circles, two circles and a line segment, or one circle and two line segments is empty. Thus, P is the intersection between C_{i_0} and either a unique C_i or a unique line segment $[s_i, s_j]$. Therefore, the edge of $\partial(\mathcal{F})$ incident to C_{i_0} at P is either a circular arc supported by C_i or a line segment supported by $[s_i, s_j]$. Hence, the labels of the edges of $\partial(\mathcal{F})$ incident to C_{i_0} can be found at no extra cost during the construction.

Since $\mathcal{A}$ is the arrangement of the circles of radius R centered at the footholds, $\sum_{i_0=1}^{n} k_{i_0} = O(|\mathcal{A}|)$. The above considerations yield the following theorem.

THEOREM 4.10. *We can compute $\partial(\mathcal{F}) \cap \mathcal{A}$ and the labels of the edges of $\partial(\mathcal{F})$ incident to the arcs of $\partial(\mathcal{F}) \cap \mathcal{A}$ in $O(|\mathcal{A}| \log n)$ time using $O(|\mathcal{A}|\alpha(n))$ space.*

4.4. Computation of the arcs of $\partial(\mathcal{F})$ issued from a foothold. The previous section has shown how to compute all the vertices of $\mathcal{F}$ that are incident to at least one circular arc. It remains to find the vertices of $\mathcal{F}$ incident to two straight edges. As we have seen in section 2, a vertex of $\mathcal{F}$ incident to two straight edges of $\partial(\mathcal{F})$ is a foothold. Furthermore, considering a foothold s_{i_0} in a cell Γ of $\mathcal{A}$, s_{i_0} is a vertex of $\mathcal{F}$ incident to two straight edges of $\partial(\mathcal{F})$ if and only if s_{i_0} is a vertex of the convex hull of the footholds reachable from s_{i_0}. The k'_{i_0} footholds contained in the disk of radius R centered at s_{i_0} can be found in $O(k'_{i_0})$ amortized time because we have already computed the Delaunay triangulation of the footholds [6, 17]. Thus, we can decide if s_{i_0} is a vertex of the convex hull of these k'_{i_0} footholds in $O(k'_{i_0})$ time and space. When s_{i_0} is a vertex of the convex hull, we can also find the two edges of the convex hull adjacent to s_{i_0} in $O(k'_{i_0})$ time and space. As the sum of the k'_i for $i \in \{1, \ldots, n\}$ is bounded by the size of $\mathcal{A}$, we obtain the following theorem.

THEOREM 4.11. *The footholds belonging to $\partial(\mathcal{F})$ and the labels of the arcs of $\partial(\mathcal{F})$ issued from these footholds can be found in $O(|\mathcal{A}|)$ time and space.*

4.5. Construction of $\mathcal{F}$.

THEOREM 4.12. *The free space of the spider robot can be computed in $O(|\mathcal{A}| \log n)$ time using $O(|\mathcal{A}|\alpha(n))$ space.*

Proof. By Theorem 4.10, we have computed all the circular arcs of $\partial(\mathcal{F})$ and the labels of the edges of $\partial(\mathcal{F})$ incident to them. By Theorem 4.11, we have computed all the vertices of $\partial(\mathcal{F})$ that are incident to two straight edges of $\partial(\mathcal{F})$ and the label of these two edges. It remains to sort the vertices of $\partial(\mathcal{F})$ that appear on the line segments $[s_i, s_j]$. We only consider the line segments $[s_i, s_j]$ such that the corresponding label (i, j) appears during previous computations. Then, we sort the vertices of $\partial(\mathcal{F})$ that belong to each such relevant line. Since $|\partial(\mathcal{F})| = \Theta(|\mathcal{A}|)$ [2], sorting all these vertices can be done in $O(|\mathcal{A}| \log n)$ time. A complete description of $\partial(\mathcal{F})$ then follows easily. □

5. Generalization to polygonal foothold regions.

5.1. Introduction and preliminaries. We consider now the case where the set of footholds is no longer a set of points but a set $\mathcal{S}$ of pairwise disjoint polygonal regions bounded by n line segments $e_1, \ldots, e_n$. Clearly, $\mathcal{S}$ is a subset of the free space $\mathcal{F}$ of the spider robot. Let $\mathcal{F}_e$ denote the free space of the spider robot using as foothold regions only the edges $e_1, \ldots, e_n$. Suppose that the spider robot admits a stable placement outside $\mathcal{S}$ with its feet inside some polygonal footholds; then the placement remains stable if it retracts its legs on the boundary of these polygonal regions. Hence, $\mathcal{F} = \mathcal{F}_e \cup \mathcal{S}$. We show how to compute $\mathcal{F}_e$.

As observed in Remark 3.8, the results of section 3 remain true if the foothold regions are line segments provided that $\mathcal{H}_i$ is replaced by $\mathcal{H}_{e_i}$, the generalized helicoidal volume defined by (see Figure 5.1)

$$\mathcal{H}_{e_i} = \{(P, \theta) \in \mathbb{R}^2 \times S^1 \mid P \in HD(s, \theta),\ s \in e_i\}.$$

The helicoidal volume associated to a point site s_i will be, henceforth, denoted by $\mathcal{H}_{s_i}$.

Similarly, we define the generalized circle C_{e_i} as the set of points at distance R from e_i. Let $\mathcal{A}_e$ denote the arrangement of the n generalized circles $C_{e_1}, \ldots, C_{e_n}$. Notice that $|\mathcal{A}_e| = \Theta(n^2)$.

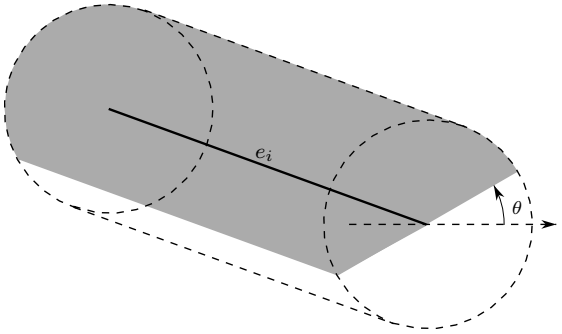


FIG. 5.1. *Section of $\mathcal{H}_{e_i}$ by the "plane" Π_θ.*

Each arc of the boundary $\partial(\mathcal{F}_e)$ of $\mathcal{F}_e$ is either an arc of C_{e_i} corresponding to a maximal extension of one leg or an arc corresponding to placements at the limit of stability of the spider robot. Similar to what we did in section 4, we compute first the contribution of each C_{e_i} to $\partial(\mathcal{F}_e)$ (section 5.2). Thereafter, we compute the arcs of $\partial(\mathcal{F}_e)$ that correspond to placements where the spider robot is at the limit of stability (section 5.3). Finally, we show how to construct $\mathcal{F}_e$ (and $\mathcal{F}$) in section 5.4.

Figure 5.2 shows an example of free space $\mathcal{F}_e$ for polygonal foothold regions.

5.2. Computation of $\partial(\mathcal{F}_e) \cap \mathcal{A}_e$. We compute the contribution to $\partial(\mathcal{F}_e)$ of each generalized circle C_{e_i} in turn. We consider the contribution of $C_{e_{i_0}}$ to $\partial(\mathcal{F}_e)$ for some $i_0 \in \{1, \ldots, n\}$. $C_{e_{i_0}}$ is composed of two half-circles and two straight line segments. In order to compute the contribution of $C_{e_{i_0}}$ to $\partial(\mathcal{F}_e)$, first we evaluate the contribution of the half-circles and then the contribution of the straight line segments. For convenience, we will not compute the contribution of the half-circles to $\partial(\mathcal{F}_e)$ but the contribution of the whole circles. Similarly, we will compute the contribution of the whole straight lines supporting the line segments of $C_{e_{i_0}}$.

Let s_{i_0} and s'_{i_0} denote the two endpoints of the line segment e_{i_0}, and let $C_{s_{i_0}}$ and $C_{s'_{i_0}}$ denote the unit circles centered at s_{i_0} and s'_{i_0}, respectively. Let l_{i_0} and l'_{i_0} denote the two straight line segments of $C_{e_{i_0}}$, and L_{i_0} and L'_{i_0} their supporting lines. We show how to compute the contributions of $C_{s_{i_0}}$ and L_{i_0} to $\partial(\mathcal{F}_e)$; the contributions of $C_{s'_{i_0}}$ and L'_{i_0} can be computed likewise.

Let $\mathcal{C}_{s_{i_0}} = C_{s_{i_0}} \times S^1$ and $\mathcal{L}_{i_0} = L_{i_0} \times S^1$. Basically, we compute $\partial(\mathcal{F}_e) \cap C_{s_{i_0}}$ and $\partial(\mathcal{F}_e) \cap L_{i_0}$, as explained in section 4.1, by computing $\cup_i(\mathcal{H}_{e_i} \cap \mathcal{C}_{s_{i_0}})$, $\cup_{i \neq i_0}(\mathcal{H}_{e_i} \cap \mathcal{C}_{s_{i_0}})$, $\cup_i(\mathcal{H}_{e_i} \cap \mathcal{L}_{i_0})$, and $\cup_{i \neq i_0}(\mathcal{H}_{e_i} \cap \mathcal{L}_{i_0})$. The properties of the new regions $\mathcal{Z}_{e_i} = \mathcal{H}_{e_i} \cap \mathcal{C}_{s_{i_0}}$ and $\mathcal{Y}_{e_i} = \mathcal{H}_{e_i} \cap \mathcal{L}_{i_0}$ are different though similar to the properties of $\mathcal{Z}_{s_i} = \mathcal{H}_{s_i} \cap \mathcal{C}_{s_{i_0}}$ described in section 4.2. The analysis of $\mathcal{Z}_{e_i}$ and $\mathcal{Y}_{e_i}$ is subdivided into two parts: first, we consider the line D_i supporting e_i and we examine the regions $\mathcal{Z}_{D_i} = \mathcal{H}_{D_i} \cap \mathcal{C}_{s_{i_0}}$ and $\mathcal{Y}_{D_i} = \mathcal{H}_{D_i} \cap \mathcal{L}_{i_0}$, where $\mathcal{H}_{D_i}$ is the generalized helicoidal volume induced by D_i,

$$\mathcal{H}_{D_i} = \{(P, \theta) \in \mathbb{R}^2 \times S^1 \mid P \in HD(s, \theta),\ s \in D_i\}.$$

Then we deduce $\mathcal{Z}_{e_i}$ (respectively, $\mathcal{Y}_{e_i}$) from $\mathcal{Z}_{D_i}$, $\mathcal{Z}_{s_i}$, and $\mathcal{Z}_{s'_i}$ (respectively, $\mathcal{Y}_{D_i}$, $\mathcal{Y}_{s_i} = \mathcal{H}_{s_i} \cap \mathcal{L}_{i_0}$, and $\mathcal{Y}_{s'_i}$), where s_i and s'_i are the two endpoints of e_i. Thereafter, we compute the contribution of $C_{e_{i_0}}$ to $\partial(\mathcal{F}_e)$ in a way similar to what we did in section 4.3. The following theorem sums up these results.

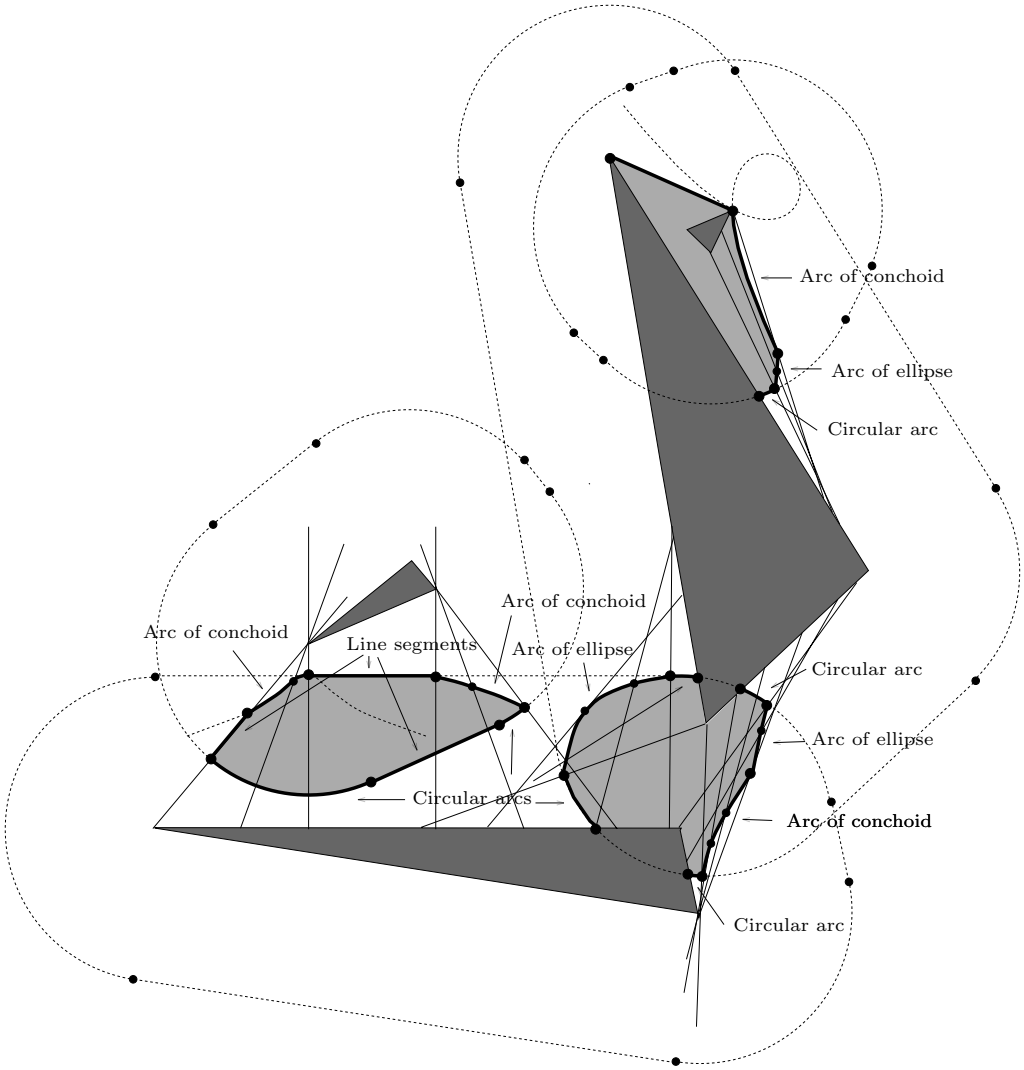


FIG. 5.2. *Example of free space $\mathcal{F}_e$ for polygonal foothold regions. The polygonal foothold regions are shown in dark grey. The other parts of $\mathcal{F}_e$ are in light grey. The C_{e_i} and some arcs of conchoid are dashed. All the line segments touching the polygons in two points are of length $2R$ and represent the ladder introduced in section* 5.3.

THEOREM 5.1. *We can compute $\partial(\mathcal{F}_e) \cap \mathcal{A}_e$ and the labels of the edges of $\partial(\mathcal{F}_e)$ incident to the arcs of $\partial(\mathcal{F}_e) \cap \mathcal{A}_e$ in $O(|\mathcal{A}_e|\alpha_7(n)\log n)$ time using $O(|\mathcal{A}_e|\alpha_8(n))$ space.*

The proof of this theorem, omitted here, is a direct generalization of the proof of Theorem 4.10. Details are given in [5] and [12].

5.3. Arcs of $\partial(\mathcal{F}_e)$ corresponding to the placements where the spider robot is at the limit of stability. We now have to compute the edges of $\mathcal{F}_e$ that do not belong to $\mathcal{A}_e$. The arcs of $\partial(\mathcal{F}_e) \cap \mathcal{A}_e$ correspond to placements at the limit of accessibility of the spider robot, and vice versa. Thus, other edges of $\mathcal{F}_e$ correspond to

placements at the limit of stability of the spider robot. We denote by $\partial(\mathcal{F}_e)_{stab}$ the set of those edges. A placement P of the spider robot is at the limit of stability if and only if there exists a closed half-disk of radius R centered at P that does not contain any foothold except at least two footholds located on the diameter of the half-disk such that P is between these footholds (see Figure 5.3). Therefore, the edges of $\partial(\mathcal{F}_e)_{stab}$ are portions of the curves drawn by the midpoint of a ladder of length $2R$ moving by translation and rotation such that the ladder touches the boundary of the foothold regions in two points but does not intersect the interior of the foothold regions. Hence, the edges of $\partial(\mathcal{F}_e)_{stab}$ are supported by the projection (onto $\mathbb{R}^2$) of the edges of the boundary of the free space of the ladder moving by translation and rotation amidst the foothold regions considered as obstacles, i.e., the set of $(P,\theta) \in \mathbb{R}^2 \times \mathbb{R}/\pi\mathbb{Z}$ such that the ladder of length $2R$ that has its midpoint at P and makes an angle θ with the x-axis does not properly intersect the interior of the foothold regions. According to [16], the edges of the boundary of the free space of the ladder can be computed in $O(|\mathcal{A}_e| \log n)$ time using $O(|\mathcal{A}_e|)$ space. The projection (onto $\mathbb{R}^2$) of each edge can easily be computed in constant time. Thus, we can compute, in $O(|\mathcal{A}_e| \log n)$ time and $O(|\mathcal{A}_e|)$ space (using [16]), a set of curves in $\mathbb{R}^2$ that contains the arcs of $\partial(\mathcal{F}_e)$ that correspond to placements at the limit of stability of the spider robot. However, it remains to compute the portions of these curves that belong to $\partial(\mathcal{F}_e)$.

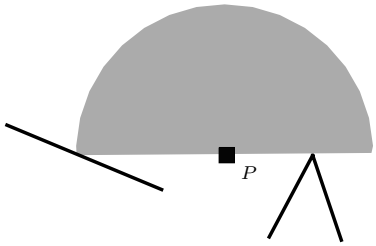


FIG. 5.3. *Example of placement P at the limit of stability.*

5.3.1. Notations and definitions. The relative interior of an e_i is called a *wall*. An endpoint of an e_i is called a *corner* (when several walls share an endpoint, we define only one corner at that point). The *ladder* is a line segment of length $2R$. A *placement* of the ladder is a pair $(P,\theta) \in \mathbb{R}^2 \times \mathbb{R}/\pi\mathbb{Z}$, where P is the location of the midpoint of the ladder and θ is the angle between the x-axis and the ladder. A *free placement* of the ladder is a placement where the ladder does not properly intersect the walls or partially lies on some walls and does not properly intersect the others. (If none of the polygonal regions of $\mathcal{S}$ are degenerated into line segments or points, then a free placement of the ladder is a placement where the ladder does not intersect the interior of the polygonal regions of $\mathcal{S}$.) A *placement of type corner-ladder* is a placement of the ladder such that the relative interior of the ladder touches a *corner*. A *placement of type wall-endpoint* is a placement of the ladder such that an endpoint of the ladder touches a *wall*. A *placement of type corner-endpoint* is a placement of the ladder such that an endpoint of the ladder touches a corner. We now define *k-contact placements* of the ladder.

A 1-*contact placement* is a free placement of type corner-ladder or wall-endpoint. A 2-*contact placement* is either the combination of two 1-contact placements or a free placement of type corner-endpoint. A 2-contact placement is said to be of type (corner-ladder)^2, (corner-ladder, wall-endpoint), (wall-endpoint)^2, or (corner-

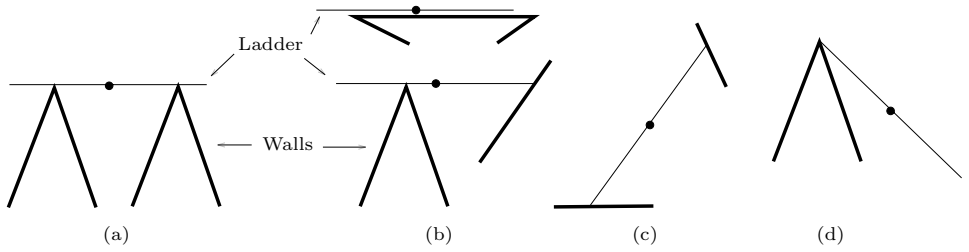


FIG. 5.4. *Examples of* 2*-contact placements of type* (a): *(corner-ladder)*^{2}, (b): *(corner-ladder, wall-endpoint),* (c): *(wall-endpoint)*^{2} *and* (d): *(corner-endpoint).*

endpoint), in accordance to the types of placements involved in the 2-contact placement (see Figure 5.4). Given two walls (respectively, a wall and a corner, two corners, one corner), the set of 2-contact placements induced by these two walls (respectively, the wall and the corner, the two corners, the single corner) is called a 2-*contact curve*. The type of a 2-contact curve is the type of the 2-contact placement defining the curve. Note that the 2-contact curves are defined in $\mathbb{R}^2 \times \mathbb{R}/\pi\mathbb{Z}$. A 3-*contact placement* is a combination of a 1-contact placement and a 2-contact placement. The types of 3-contact placements are naturally given by (corner-ladder)^3, (corner-endpoint, wall-endpoint). With this definition, we unfortunately cannot guarantee that all the 2-contact curves end at 3-contact placements. Indeed, a 2-contact curve defined by the ladder sliding along a wall (see Figure 5.4b) ends on one side (if no other wall blocks the sliding motion) at a 2-contact placement of type (corner-endpoint), where the ladder is collinear with the wall, without properly intersecting it. In order to ensure that all the 2-contact curves end at 3-contact placements, we consider these 2-contact placements as 3-contact placements and denote their type by (corner-endpoint, $\|$). A k-*contact placement*, $k > 3$, is the combination of p 1-contact placements, q 2-contact placements, and r 3-contact placements such that $p + 2q + 3r = k$.

Now, we define a 2-*contact tracing* as the projection onto $\mathbb{R}^2$ of a 2-contact curve. Similarly as above, we define the types of the 2-contact tracings. Notice that, to any point P on a given 2-contact tracing $\mathcal{K}$ corresponds a unique placement (P, θ) on the 2-contact curve that projects onto $\mathcal{K}$. It follows that to any point P on a 2-contact tracing $\mathcal{K}$ corresponds a unique pair (M, N) of points of contact between the ladder at (P, θ) and the walls (M and N are equal when $\mathcal{K}$ is a 2-contact tracing of type (corner-endpoint)); when P is an endpoint of $\mathcal{K}$, a 3-contact placement corresponds to P, however, (M, N) is uniquely defined by continuity. The points M and N are called the *contact points corresponding to* $P \in \mathcal{K}$. We also define the three contact points corresponding to a 3-contact placement.

A 2-contact tracing is a straight line segment, an arc of ellipse, an arc of conchoid, or a circular arc. Indeed (see Figures 5.5, 5.6, 5.7, and 5.8), a 2-contact tracing of type (corner-endpoint) is a circular arc; a 2-contact tracing of type (wall-endpoint)^2 is an arc of ellipse; a 2-contact tracing of type (corner-ladder, wall-endpoint) is an arc of conchoid (see [5]); and a 2-contact tracing of type (corner-ladder)^2 is a straight line segment. As we said before, we can compute all these 2-contact tracings in $O(|\mathcal{A}_e| \log n)$ time using $O(|\mathcal{A}_e|)$ space [16], and it remains to compute the portions of these curves that belong to $\partial(\mathcal{F}_e)$.

5.3.2. Overview. We first show that only some portions of the 2-contact tracings correspond to positions at the limit of stability of the spider robot (section 5.3.3).

These portions are called the relevant 2-contact tracings. Then, we prove that we do not have to take into consideration the intersections between the relative interior of relevant 2-contact tracings (Proposition 5.2). We also show that, if a point A is an endpoint of several relevant 2-contact tracings, only two of them can support edges of $\partial(\mathcal{F}_e)_{stab}$ in the neighborhood of A (Propositions 5.3). Finally (section 5.3.4), we compute a graph whose edges are relevant 2-contact tracings and where the degree of each node is at most two. This graph induces a set Δ of curves supporting $\partial(\mathcal{F}_e)_{stab}$ (Theorem 5.4) that will allow us to compute $\partial(\mathcal{F}_e)_{stab}$ in section 5.4.

5.3.3. Relevant 2-contact tracings. As mentioned above, a placement P of the spider robot is at the limit of stability if and only if there exists a closed half-disk of radius R centered at P that does not contain any foothold except at least two footholds located on the diameter of the half-disk, one on each side of P. Thus, a point P of a 2-contact tracing $\mathcal{K}$ belongs to an arc of $\partial(\mathcal{F}_e)_{stab}$ only if P lies between the two contact points corresponding to $P \in \mathcal{K}$. The portions of the 2-contact tracings for which that property holds are called the *relevant 2-contact tracings*. The other portions are called the *irrelevant 2-contact tracings*. We now show how to compute the relevant 2-contact tracings for each type of contact. Let $\mathcal{K}$ denote a 2-contact tracing, let $P \in \mathcal{K}$ and let M and N be the two contact points corresponding to $P \in \mathcal{K}$. In Figures 5.5, 5.6, 5.7, and 5.8, the walls and the relevant 2-contact tracings are thick, the irrelevant 2-contact tracings are dashed thick, and the ladder is thin.

Type (corner-endpoint): $\mathcal{K}$ is a circular arc, and M and N coincide with one endpoint of the ladder. Thus, all the 2-contact tracings of type (corner-endpoint) are wholly irrelevant.

Type (wall-endpoint)^2: $\mathcal{K}$ is an arc of ellipse, M and N are the endpoints of the ladder, and thus, P lies between them. Therefore, all the 2-contact tracings of type (wall-endpoint)^2 are wholly relevant.

Type (corner-ladder, wall-endpoint): $\mathcal{K}$ is an arc of conchoid. If the distance between the corner and the wall is greater than R, then $\mathcal{K}$ is wholly relevant.

Otherwise, if that distance is smaller than R, then the two relevant portions and the irrelevant portion of $\mathcal{K}$ are incident to the corner involved in the type of $\mathcal{K}$.

Notice that if the corner is an endpoint of the wall (see Figure 5.4b), then $\mathcal{K}$ degenerates into a line segment, and the irrelevant portion of $\mathcal{K}$ is the portion which is not supported by the wall.

Type (corner-ladder)^2: $\mathcal{K}$ is a line segment. If the distance between the two corners is greater than R, then $\mathcal{K}$ is wholly relevant; otherwise, the portion of $\mathcal{K}$ which is relevant is the line segment joining the two corners.

We now show that the intersections between the relative interiors of the relevant 2-contact tracings are not interesting for the spider robot motion problem. We recall that if a vertex A of $\partial(\mathcal{F}_e)$ belongs to $\mathcal{A}_e$, then we know by Theorem 5.1 the labels of the edges of $\partial(\mathcal{F}_e)$ incident to A. Otherwise, if $A \notin \mathcal{A}_e$, then the two edges of $\partial(\mathcal{F}_e)$ that end at A correspond to placements at the limit of stability of the spider robot.

PROPOSITION 5.2. *Any vertex A of $\partial(\mathcal{F}_e)$, such that $A \notin \mathcal{A}_e$, is an endpoint of the two relevant 2-contact tracings supporting the edges of $\partial(\mathcal{F}_e)$ ending at A.*

Proof. Since the two edges of $\partial(\mathcal{F}_e)$ that end at A correspond to placements at the limit of stability of the spider robot, they are both supported by some relevant 2-contact tracings. Thus, we only have to prove that A is an endpoint of these two relevant 2-contact tracings.

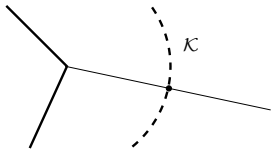


FIG. 5.5. *Irrelevant* 2*-contact tracing of type (corner-endpoint), i.e., circular arc.*

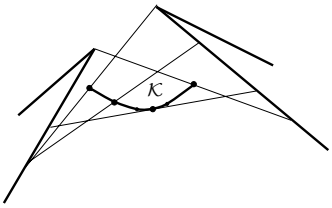


FIG. 5.6. *Relevant* 2*-contact tracing of type (wall-endpoint)*^{2}*, i.e., arc of ellipse.*

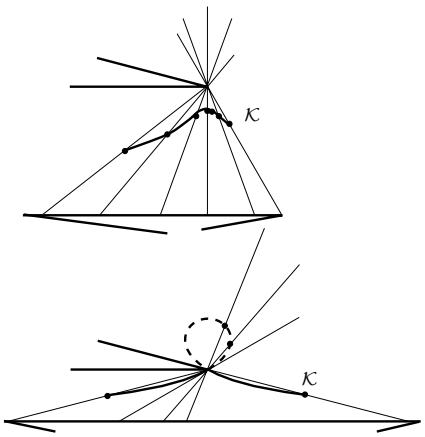


FIG. 5.7. *Relevant, and partially relevant,* 2*-contact tracings of type (corner-ladder, wall-endpoint), i.e., arcs of conchoid.*

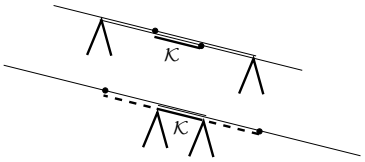


FIG. 5.8. *Relevant, and partially relevant,* 2*-contact tracings of type (corner-ladder)*^{2}*.*

Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be these two relevant 2-contact tracings and assume for a contradiction that A is not an endpoint of $\mathcal{K}_1$ (nothing is assumed for A with respect to $\mathcal{K}_2$). Let $L_1 = (A, \theta_1)$ (respectively, $L_2 = (A, \theta_2)$) be the placement of the ladder that correspond to $A \in \mathcal{K}_1$ (respectively, $A \in \mathcal{K}_2$) and let M_1 and N_1 (respectively, M_2 and N_2) be the corresponding contact points (see Figure 5.9). First, notice that $L_1 \neq L_2$. Indeed, otherwise, L_1 is at least a 3-contact placement, and then A must be an endpoint of $\mathcal{K}_1$, which contradicts our assumption.

By the definition of the relevant 2-contact tracings, A is between M_1 and N_1. Moreover, A cannot be equal to M_1 or N_1 since A is not an endpoint of $\mathcal{K}_1$. It follows that neither M_2 nor N_2 is equal to A, because otherwise L_1 would be a 3-contact placement. Therefore, A is strictly between M_1 and N_1 and strictly between M_2 and N_2. Thus, A is strictly inside the polygon $(M_1M_2N_1N_2)$.

On the other hand, since $A \notin \mathcal{A}_e$, A does not belong to any C_{e_i}, and therefore, the walls supporting M_1, N_1, M_2, and N_2 intersect the open disk D_A of radius R centered at A. Thus, there exist four points M_1', N_1', M_2', and N_2' on these walls and in D_A that are close enough to M_1, N_1, M_2, and N_2, respectively, to ensure that A belongs to the interior of the polygon $(M_1'M_2'N_1'N_2')$. Since the distances from A to M_1', N_1', M_1', and N_2', are strictly smaller than R, A belongs to the interior of $\mathcal{F}_e$. This contradicts our assumption that A is a vertex of $\partial(\mathcal{F}_e)$ and yields the result. □

Consider now the adjacency graph $\mathcal{G}$ of the relevant 2-contact tracings such that two relevant 2-contact tracings are connected in $\mathcal{G}$ if and only if they have a common

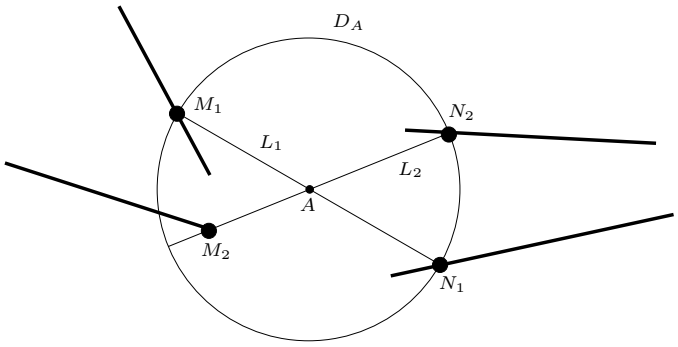


FIG. 5.9. *For the proof of Proposition* 5.2.

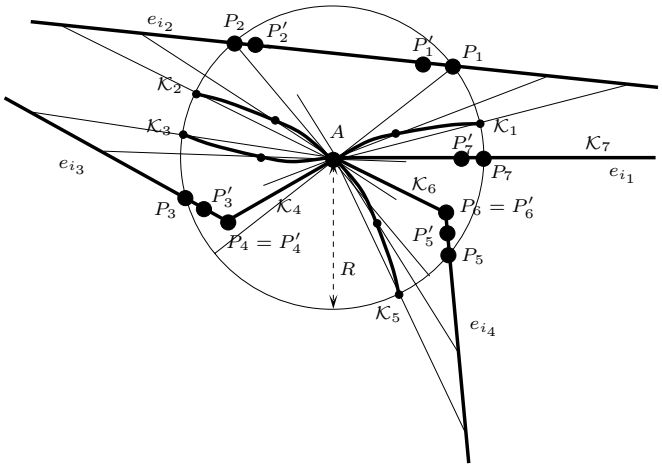


FIG. 5.10. *Relevant* 2*-contact tracings* $\mathcal{K}_1, \ldots, \mathcal{K}_7$ *ending at* A. $\mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3$, *and* $\mathcal{K}_5$ *are* 2*-contact tracings of type (corner-ladder, wall-endpoint) (i.e., arcs of conchoid).* $\mathcal{K}_7$ *is a degenerated* 2*-contact tracing of type (corner-ladder, wall-endpoint) (i.e., a line segment).* $\mathcal{K}_4$ *and* $\mathcal{K}_5$ *are* 2*-contact tracings of type (corner-ladder)*^{2} *(i.e., line segments).*

endpoint (the intersections between the relative interiors of the relevant 2-contact tracings are not considered). Notice that, given the set of relevant 2-contact tracings, $\mathcal{G}$ can be easily computed in $O(|\mathcal{A}_e| \log n)$ time. Now, given two vertices of $\partial(\mathcal{F}_e) \cap \mathcal{A}_e$ that are connected along $\partial(\mathcal{F}_e)$ by arcs of $\partial(\mathcal{F}_e)_{stab}$, we want to compute these arcs. For computing these arcs, we cannot simply use the graph $\mathcal{G}$ because the degree of some nodes of $\mathcal{G}$ may be arbitrarily large (see Figure 5.10). We show in the next proposition that we can deduce from $\mathcal{G}$ a graph $\mathcal{G}^*$ such that the degree of each node of $\mathcal{G}^*$ is at most two and that $\mathcal{G}^*$ supports any portion of $\partial(\mathcal{F}_e)$ which is the concatenation of arcs of $\partial(\mathcal{F}_e)_{stab}$.

We consider four hypotheses (H1,...,H4) that obviate the need to consider degenerate cases. They are not essential but substantially simplify the proof of the following proposition. The first three hypotheses are made to ensure that the degree of each vertex of the free space of the ladder is 3.

(H1) The line segments $e_1, \ldots, e_n$ compose the boundary of a set of nondegenerated polygons (i.e., no polygon is reduced to a line segment or to a point).

(H2) The ladder does not admit any 4-contact placement.

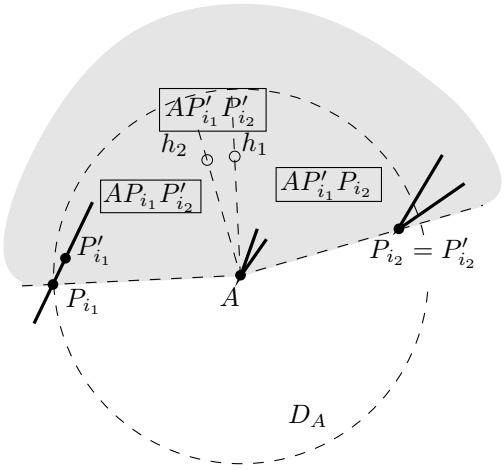


FIG. 5.11. *Wedge $P_{i_1}AP_{i_2}$ is in $\mathcal{F}_e$ near A.*

(H3) The arc (of conchoid) drawn by an endpoint of the ladder when its other endpoint moves along a wall while the ladder remains in contact with a corner is not tangent to any other wall.

(H4) The ladder does not admit any 3-contact placement when its midpoint is located at a corner.

PROPOSITION 5.3. *For any node A of $\mathcal{G}$ of degree k such that $A \notin \mathcal{A}_e$, at most two relevant 2-contact tracings can support $\partial(\mathcal{F}_e)$ in a sufficiently small neighborhood of A. Moreover, we can determine these at most two curves in $O(k \log k)$ time using $O(k)$ space.*

Proof. Let $A \notin \mathcal{A}_e$ be a node of $\mathcal{G}$ of degree k. We assume that $k > 2$, otherwise Proposition 5.3 is trivial. Let $\mathcal{K}_1, \ldots, \mathcal{K}_k$ be the relevant 2-contact tracings that end at A and let $L_i = (A, \phi_i)$ be the placement of the ladder that corresponds to $A \in \mathcal{K}_i$. D_A is the open disk of radius R centered at A. We distinguish two cases whether A is a corner or not.

Case 1. A is a corner. (See Figure 5.10.)

The 2-contact tracing $\mathcal{K}_i$ involves at least another contact than the corner-ladder contact at A. This contact cannot be of type corner-endpoint by (H4). If the contact is of type wall-endpoint, we define P_i as the contact point between this wall and the ladder at placement L_i (see Figure 5.11). Since $A \notin \mathcal{A}_e$, the wall must intersect D_A and we define P'_i as a point close to P_i in that intersection. If the contact is of type corner-ladder, we define $P_i = P'_i$ as the corner (distinct from A) involved in this contact (notice that $P_i = P'_i \in D_A$ by (H4)).

Fact. For all $i \neq j$, $\phi_i \neq \phi_j$.

Otherwise, $L_i = L_j$ is a 3-contact placement contradicting (H4).

Fact. A is a nonflat vertex of CH$(A, P_1, \ldots, P_k)$ or belongs to the interior of $\mathcal{F}_e$.

Assume that $A \in \partial(\mathcal{F}_e)$. Then A lies on the boundary of CH$(A, P_1, \ldots, P_k)$ because otherwise the P'_i provides footholds such that the spider robot can move in a neighborhood of A. Furthermore, A must be a nonflat vertex of CH$(A, P_1, \ldots, P_k)$ by (H4).

Assume now that $A \in \partial(\mathcal{F}_e)$, and let P_{i_1} and P_{i_2} be the two vertices of CH$(A, P_1, \ldots, P_k)$ such that P_{i_1}, A, and P_{i_2} are consecutive along the boundary of CH$(A,$

$P_1, \ldots, P_k$) (see Figure 5.11). We will exhibit a stable placement for the spider robot at any position P inside the intersection of the wedge $P_{i_1}AP_{i_2}$ and a neighborhood of A. Let h_1 and h_2 be two points in the wedge $P_{i_1}AP_{i_2}$ such that the wedges $P_{i_1}Ah_1$ and $h_2AP_{i_2}$ are right (see Figure 5.11).

— If P is in the wedge $P_{i_1}Ah_2$ and is close enough to A, the footholds A, P_{i_1}, and P'_{i_2} yield a stable placement for the spider robot.

— If P is in the wedge h_2Ah_1 and is close enough to A, the footholds A, P'_{i_1}, and P'_{i_2} yield a stable placement for the spider robot.

— If P is in the wedge $h_1AP_{i_1}$ and is close enough to A, the footholds A, P'_{i_1}, and P_{i_2} yield a stable placement for the spider robot.

Fact. $\mathcal{K}_i$, $i \notin \{i_1, i_2\}$, *cannot support an edge of* $\partial(\mathcal{F}_e)$ *incident to* A.

We assume that $A \in \partial(\mathcal{F}_e)$, because otherwise the claim is obvious. It follows that A is a nonflat vertex of $\mathrm{CH}(A, P_1, \ldots, P_k)$. A 2-contact tracing $\mathcal{K}_i$, $i \notin \{i_1, i_2\}$ cannot be an arc of ellipse, because otherwise L_i is a 3-contact placement (because A is a corner here) contradicting (H4). Then, $\mathcal{K}_i$ can be either the segment AP_i or an arc of conchoid. If $\mathcal{K}_i$ is an arc of conchoid, then, by the general properties of conchoids (see [5]), $\mathcal{K}_i$ is tangent to the segment AP_i at A. Thus, $\mathcal{K}_i$ is always tangent to the segment AP_i at A. The point P_i strictly belongs to the wedge $P_{i_1}AP_{i_2}$, because we have shown that $\phi_i \notin \{\phi_{i_1}, \phi_{i_2}\}$. Thus, in a neighborhood of A, $\mathcal{K}_i$ is strictly inside the wedge $P_{i_1}AP_{i_2}$ and thus strictly inside $\mathcal{F}_e$. Therefore, $\mathcal{K}_i$ cannot support $\partial(\mathcal{F}_e)$ in a neighborhood of A.

Hence, by sorting the P_i by their polar angles around A, we can determine, in $O(k \log k)$ time, if A is a nonflat vertex of $\mathrm{CH}(A, P_1, \ldots, P_k)$ and, if so, determine i_1 and i_2. If A is a nonflat vertex of $\mathrm{CH}(A, P_1, \ldots, P_k)$, then only $\mathcal{K}_{i_1}$ and $\mathcal{K}_{i_2}$ can support an edge of $\partial(\mathcal{F}_e)$ incident to A. Otherwise, A belongs to the interior of $\mathcal{F}_e$ and none of the 2-contact tracings $\mathcal{K}_1, \ldots, \mathcal{K}_k$ can support an edge of $\partial(\mathcal{F}_e)$ incident to A.

Case 2. A is not a corner.

Fact. If there exists $i \neq j$ *such that* $\phi_i \neq \phi_j$, *then* A *belongs to the interior of* $\mathcal{F}_e$.

For each relevant 2-contact placement $L_i = (A, \phi_i)$, there exists two contact points M_i and N_i on each side of A at distance less than or equal to R. Since A is not a corner, neither M_i nor N_i is equal to A, thus A belongs to the relative interior of the segment M_iN_i. When $\phi_i \neq \phi_j$, it follows that A belongs to the interior of the polygon $(M_iM_jN_iN_j)$ (see Figure 5.9). Similarly as in the proof of Proposition 5.2, since $A \notin \mathcal{A}_e$, there exist four footholds M'_i, N'_i, M'_j, N'_j in D_A and in some neighborhoods of M_i, N_i, M_j, N_j, respectively, such that A belongs to the interior of the polygon $(M'_iM'_jN'_iN'_j)$. Thus, A belongs to the interior of $\mathcal{F}_e$.

Hence, if there exists $i \neq j$ such that $\phi_i \neq \phi_j$, none of the 2-contact tracings $\mathcal{K}_1, \ldots, \mathcal{K}_k$ can support an edge of $\partial(\mathcal{F}_e)$ incident to A. We now assume that $\phi_i = \phi_j$ $\forall i, j$.

Fact. There are at most six 2*-contact tracings incident to* A.

The general position hypothesis (H2) forbids k-contacts for $k > 3$; thus A corresponds to a 3-contact placement. The three possible choices of two contacts among three give three 2-contact tracing intersecting in A and thus six arcs incident to A.

Fact. There are three 2*-contact tracings incident to* A.

If the 3-contact placement L is of type (corner-endpoint, $\|$), then there are only three 2-contact tracings incident to A that are two circular arcs and one line segment. Otherwise, it comes from the general position hypotheses (H1), (H2), and (H3) (designed to ensure that property) that a 2-contact tracing cannot be valid on both side

of the 3-contact, i.e., on one side of the 3-contact placement, the placements are not free. The proof that the hypotheses ensured that fact is detailed in [5].

Fact. There are two relevant 2*-contact tracings incident to* A.

Since A is not a corner, at the 3-contact placement L, two contact points are on the same side of A. Thus, only two of the three 2-contact tracings incident to A are relevant. □

5.3.4. Construction of Δ. Now consider the graph $\mathcal{G}$ and each node A in turn. If $A \in \mathcal{A}_e$, we disconnect all the edges of $\mathcal{G}$ that end at A. Notice that for each such node A, we know, by Theorem 5.1, whether $A \in \partial(\mathcal{F}_e)$ and, in such a case, the labels of the edges of $\partial(\mathcal{F}_e)$ incident to A. If $A \notin \mathcal{A}_e$, we disconnect the edges ending at A except those (at most two) that may support $\partial(\mathcal{F}_e)$ in a neighborhood of A (see Proposition 5.3). In this way, we obtain a graph $\mathcal{G}^*$ such that the degree of each node is one or two. We consider each connected component of this new graph as a curve. Let Δ be this set of curves. These curves are represented in $\mathcal{G}^*$ as chains (open or closed). Even if a curve is not simple, it follows that there exists a natural order along the curve. Then, according to Propositions 5.2 and 5.3, we get the following theorem.

THEOREM 5.4. *We can compute, in* $O(|\mathcal{A}_e| \log n)$ *time using* $O(|\mathcal{A}_e|)$ *space, a set* Δ *of curves that support the edges of* $\partial(\mathcal{F}_e)$ *corresponding to placements at the limit of stability of the spider robot. Moreover, any portion* $\mathcal{P}$ *of* $\partial(\mathcal{F}_e)$ *either intersects* $\mathcal{A}_e$ *or belongs to a unique curve of* Δ.

5.4. Construction of $\mathcal{F}_e$ and $\mathcal{F}$. We can now construct $\mathcal{F}_e$ and $\mathcal{F}$. Let $\lambda_k(n)$ denote the maximum length of the Davenport–Schinzel sequence of order k on n symbols and $\alpha_k(n) = \lambda_k(n)/n$. Note that $\alpha_3(n) = \alpha(n)$.

THEOREM 5.5. *Given, as foothold regions, a set of* n *nonintersecting straight line segments that satisfies* (H1), (H2), (H3), *and* (H4), *we can compute the free space* $\mathcal{F}_e$ *of the spider robot in* $O(|\mathcal{A}_e|\alpha_8(n) \log n)$ *time using* $O(|\mathcal{A}_e|\alpha_8(n))$ *space.*

Proof. By Theorem 5.1, we can compute the contribution of $\mathcal{A}_e$ to $\partial(\mathcal{F}_e)$ and the label of the edges of $\partial(\mathcal{F}_e)$ incident to them in $O(|\mathcal{A}_e|\alpha_7(n) \log n)$ time using $O(|\mathcal{A}_e|\alpha_8(n))$ space. By Theorem 5.4, we can compute, in $O(|\mathcal{A}_e| \log n)$ time using $O(|\mathcal{A}_e|)$ space, a set Δ of curves that support the edges of $\partial(\mathcal{F}_e)$ that do not belong to $\mathcal{A}_e$. Moreover, any portion $\mathcal{P}$ of $\partial(\mathcal{F}_e)$ such that $\mathcal{P} \cap \mathcal{A}_e = \emptyset$ belongs to a unique curve of Δ. Thus, by sorting all the vertices of $\partial(\mathcal{F}_e) \cap \mathcal{A}_e \cap \Delta$ on the relevant curves of Δ, we obtain all the edges of $\partial(\mathcal{F}_e)$ that belong to a connected component of $\partial(\mathcal{F}_e)$ intersecting $\mathcal{A}_e$. Indeed, for each vertex $A \in \partial(\mathcal{F}_e) \cap \mathcal{A}_e \cap \Delta$, we know, in a neighborhood of A, the portion of the curve of Δ that belongs to $\partial(\mathcal{F}_e)$ because we can simply determine, for each edge, a side of the edge that belongs to $\mathcal{F}_e$. (The contact points corresponding to the edges determine a side that necessarily belongs to $\mathcal{F}_e$.)[4] Then it is an easy task to deduce all the connected components of $\partial(\mathcal{F}_e)$ that intersect $\mathcal{A}_e$.

It remains to compute the connected components of $\partial(\mathcal{F}_e)$ that do not intersect $\mathcal{A}_e$. Each of these components must be a closed curve of Δ. Moreover, all the curves of Δ belong to $\mathcal{F}_e$. Thus, according to Theorem 5.4, any closed curve $\mathcal{K}$ of Δ that does not intersect $\mathcal{A}_e$ is either a connected component of $\partial(\mathcal{F}_e)$ or is strictly included in $\mathcal{F}_e$. Therefore, by considering, in addition, all the closed curves of Δ that do not intersect $\mathcal{A}_e$, we finally obtain a set Ψ of closed curves that contains $\partial(\mathcal{F}_e)$ such that any curve of Ψ is either a connected component of $\partial(\mathcal{F}_e)$ or is strictly included in $\mathcal{F}_e$.

[4] Observe that when the edge belongs to $\mathcal{F}_e$, its two sides belong to $\mathcal{F}_e$.

At last, as we can simply determine, for each curve of Ψ, a side of the edge that belongs to $\mathcal{F}_e$, we can easily deduce from Ψ the free space $\mathcal{F}_e$. That concludes the proof since all these computations can be done in $O(|\mathcal{A}_e|\alpha_8(n)\log n)$ time using $O(|\mathcal{A}_e|\alpha_8(n))$ space. □

As we said at the beginning of section 5, the free space of the spider robot using as foothold regions a set of polygonal regions is obtained by adding these polygonal regions to $\mathcal{F}_e$. This does not increase the geometric complexity of the free space the complexity of the computation. Thus, we get the following theorem.

THEOREM 5.6. *Given a set of pairwise disjoint polygonal foothold regions with n edges in total that satisfies* (H1), (H2), (H3), *and* (H4), *we can compute the free space $\mathcal{F}$ of the spider robot in $O(|\mathcal{A}_e|\alpha_8(n)\log n)$ time using $O(|\mathcal{A}_e|\alpha_8(n))$ space.*

The function $\alpha_8(n)$ is extremely slowly growing and can be considered as a small constant in practical situations. This result is almost optimal since, as shown in [2], $\Omega(|\mathcal{A}_e|)$ is a lower bound for the size of $\mathcal{F}$.

6. Conclusion. We have seen in Theorem 4.12 that, when the foothold regions are n points in the plane, the free space of the spider robot can be computed in $O(|\mathcal{A}|\log n)$ time using $O(|\mathcal{A}|\alpha(n))$ space, where $\alpha(n)$ is the pseudo inverse of Ackerman's function and $\mathcal{A}$ the arrangement of the n circles of radius R centered at the footholds. By [2] the size of $\mathcal{F}$ is known to be $\Theta(|\mathcal{A}|)$. The size of $\mathcal{A}$ is $O(n^2)$ but it has been shown in [15] that $|\mathcal{A}| = O(kn)$, where k denotes the maximum number of disks of radius R centered at the footholds that can cover a point of the plane. Thus, in case of sparse footholds, the sizes of $\mathcal{A}$ and $\mathcal{F}$ are linearly related to the number of footholds.

When the foothold regions are polygons with n edges in total, the free space of the spider robot can be computed in $O(|\mathcal{A}_e|\alpha_8(n)\log n)$ time using $O(|\mathcal{A}_e|\alpha_8(n))$ space, where $n\alpha_k(n) = \lambda_k(n)$ is the maximum length of a Davenport–Schinzel sequence of order k on n symbols, and $\mathcal{A}_e$ is the arrangement of the n curves consisting of the points lying at distance R from the straight line edges. Note that the size of $\mathcal{A}_e$ is $O(n^2)$.

It should be observed that, in the case of point footholds, our algorithm implies that $O(|\mathcal{A}|\alpha(n))$ is an upper bound for $|\mathcal{F}|$. However, this bound is not tight since $|\mathcal{F}| = \Theta(|\mathcal{A}|)$ [2]. In the case of polygonal footholds, our analysis implies that $O(|\mathcal{A}_e|\alpha_8(n))$ is an upper bound for $|\mathcal{F}|$. We leave as an open problem to close the (small) gap between this upper bound and the $\Omega(|\mathcal{A}_e|)$ lower bound.

Once the free space $\mathcal{F}$ is known, several questions can be answered. In particular, given two points in the same connected component of $\mathcal{F}$, the algorithm in [2] computes a motion of the spider robot, i.e., a motion of the body and a corresponding sequence of legs assignments that allows the robot to move from one point to the other.

The motion planning problem for other types of legged robots remains to be studied. The case where all the legs are not attached at the same point on a polygonal/polyhedral body is particularly relevant. A spider robot for which all the legs are not of the same length is also an interesting model.

Acknowledgment. We would like to thank Joseph O'Rourke for helpful comments.

REFERENCES

[1] J. E. BARES AND W. L. WHITTAKER, *Configuration of autonomous walkers for extreme terrain*, Internat. J. Robot. Res., 12 (1993), pp. 535–559.

[2] J.-D. Boissonnat, O. Devillers, L. Donati, and F. Preparata, *Motion planning of legged robots: the spider robot problem*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 3–20.
[3] J.-D. Boissonnat, O. Devillers, and S. Lazard, *From spiders robots to half-disks robots*, in Proceedings of the 11th IEEE Internat. Conf. Robot. Autom., San Diego, CA, 1994, pp. 953–958.
[4] J.-D. Boissonnat, O. Devillers, and S. Lazard, *Motion planning of legged robots*, in the First Workshop on the Algorithmic Foundations of Robotics, A. K. Peters, Boston, MA, 1994.
[5] J.-D. Boissonnat, O. Devillers, and S. Lazard, *Motion Planning of Legged Robots*, Research Report 3214, INRIA, BP93, 06902 Sophia-Antipolis, France, 1997.
[6] M. Dickerson and R. L. Drysdale, *Fixed radius search problems for points and segments*, Inform. Process. Lett., 35 (1990), pp. 269–273.
[7] J. Hershberger, *Finding the upper envelope of n line segments in $O(n \log n)$ time*, Inform. Process. Lett., 33 (1989), pp. 169–174.
[8] S. Hirose and O. Kunieda, *Generalized standard foot trajectory for a quadruped walking vehicle*, Internat. J. Robotics Research, 10 (1991), pp. 3–12.
[9] S. Hirose, M. Nose, H. Kikuchi, and Y. Umetani, *Adaptive gait control of a quadruped walking vehicle*, in Proceedings of the Internat. Symposium on Robotics Research, MIT, Cambridge, MA, 1984, pp. 253–277.
[10] K. Kedem and M. Sharir, *An efficient motion planning algorithm for a convex rigid polygonal object in 2-dimensional polygonal space*, Discrete Comput. Geom., 5 (1990), pp. 43–75.
[11] K. Kedem, M. Sharir, and S. Toledo, *On critical orientations in the Kedem–Sharir motion planning algorithm for a convex polygon in the plane*, Discrete Comput. Geom., 17 (1997), pp. 227–240.
[12] S. Lazard, *Planification de trajectoires de robots mobiles non-holonomes et de robots à pattes*, Thèse de doctorat en sciences, Université Paris 6, France, 1996.
[13] *Special issue on legged locomotion*, Internat. J. Robot. Res., 3 (1984).
[14] *Special issue on legged locomotion*, Internat. J. Robot. Res., 9 (1990).
[15] M. Sharir, *On k-sets in arrangements of curves and surfaces*, Discrete Comput. Geom., 6 (1991), pp. 593–613.
[16] S. Sifrony and M. Sharir, *A new efficient motion-planning algorithm for a rod in two-dimensional polygonal space*, Algorithmica, 2 (1987), pp. 367–402.
[17] V. Turau, *Fixed-radius near neighbors search*, Inform. Process. Lett., 39 (1991), pp. 201–203.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 247–268

TIGHT FAULT LOCALITY*

SHAY KUTTEN† AND DAVID PELEG‡

Abstract. This paper lays a theoretical foundation for scaling fault tolerant tasks to large and diversified networks such as the Internet. In such networks, there are always parts of the network that fail. On the other hand, various subtasks interest only parts of the network, and it is desirable that those parts, if nonfaulty, do not suffer from faults in other parts. Our approach is to refine the previously suggested notion of fault local algorithms (that was best suited for global tasks) for which the complexity of recovering was proportional to the number of faults. We refine this notion by introducing the concept of *tight fault locality* to deal with problems whose complexity (in the absence of faults) is sublinear in the size of the network. For a problem whose time complexity on an n-node network is $T(n)$ (where possibly $T(n) = o(n)$), a tightly fault local algorithm recovers a legal global state in $O(T(x))$ time when the (unknown) number of faults is x.

This concept is illustrated by presenting a general transformation for maximal independent set (MIS) algorithms to make them tightly fault local. In particular, our transformation yields an $O(\log x)$ randomized mending algorithm and an $\exp(O(\sqrt{\log x}))$ deterministic mending algorithm for MIS. The methods used in the transformation may be of interest by themselves.

Key words. fault tolerance, recovery, distributed algorithms, stabilization

AMS subject classifications. 68W15, 68W10, 68W20, 94C15, 68R25

PII. S0097539797319109

1. Introduction.

1.1. The problem. Many parallel models assume that information can be collected from all processors quickly. In real computer networks, however, communication is local; hence it takes at least diameter time to collect all information, leading to *global* (i.e., linear in the number of nodes) running times on worst-case topologies. Consequently, Linial and others (cf. [L92, AGLP]) promoted the notion of *local* algorithms, namely, algorithms that require the collection of data only from small neighborhoods. We refer to these algorithms also as *sublinear*, since their running time is sublinear in the diameter (or the number of nodes).

The use of sublinear or local algorithms is becoming more and more essential in order for solutions to scale to the emerging huge networks of today, e.g., the Internet. However, one striking characteristic of the research on distributed fault tolerance is that usually faults are corrected *globally*, i.e., by algorithms involving the entire system. In other cases, the running time of the correction algorithm is the same as that of the algorithm for recomputing the function from scratch (cf. [ACS94, AGLP]). Fast and local fault correction is the topic of the current paper.

We first describe the types of faults dealt with in this paper. The global state of a distributed system can be represented as a vector whose components are the local

*Received by the editors March 31, 1997; accepted for publication (in revised form) January 22, 1999; published electronically May 2, 2000.
http://www.siam.org/journals/sicomp/30-1/31910.html

†I.B.M. T.J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598, and Department of Industrial Engineering and Management, Technion, Haifa 32000, Israel (kutten @ie.technion.ac.il).

‡The Norman D. Cohen Professorial Chair of Computer Sciences, Department of Applied Mathematics and Computer Science, The Weizmann Institute of Science, Rehovot, 76100 Israel (peleg@wisdom.weizmann.ac.il). The work of this author was supported in part by a Walter and Elise Haas Career Development Award and by a grant from the Israel Science Foundation. Part of the author's work was done while visiting the IBM T.J. Watson Research Center.

states of the various nodes of the system. A transient fault in one node may change the node's state to some other local state that still looks legal. However, the global state may no longer be legal. Intensive research efforts were invested in dealing with this situation and bringing the system to a correct legal state. A common methodology is based on separating the task into two distinct subtasks, *detection* and *correction*. In this paper we concentrate on the development of fault-local variants for the correction phase.

The study of fault-local algorithms was initiated in [KP95], where the following basic question was raised. Consider a problem Π on graphs, whose solutions are representable as a function $\mathcal{V}$ from the inputs of the network vertices to their outputs. The set of legal solutions of Π on a given graph G is denoted by $\Pi(G)$. Consider a distributed network, whose nodes collectively store the representation of some solution $\mathcal{V} \in \Pi(G)$ of the problem Π on graph G. Suppose that at time t_0, the memory content stored at some subset F of the network nodes is distorted due to some transient failures. As a result, while the stored values still look locally legal, the global representation of $\mathcal{V}$ has changed into some inconsistent function $\mathcal{V}'$ that is no longer valid.

It is clear that, if the problem Π is computable, then investing sufficient computational efforts it is possible to *mend* the function, namely, change the values at some of (or all) the nodes and reconstruct a valid representation of a (possibly different) solution of the same type, $\mathcal{V}' \in \Pi(G)$. The question raised in [KP95] was whether it is possible to take advantage of the relative rarity of faults and distributively mend the function in time complexity dependent on the number of *failed nodes*, $|F|$, rather than on the size of the entire network, n. This operation (if and when possible) was termed *fault-local mending*. The problem Π is *fault locally T-mendable* if, following the occurrence of faults in a set F of nodes, the solution can be mended in $O(T(|F|))$ time. A problem Π is *fault-locally mendable* if there exists some complexity function T such that Π is fault-locally T-mendable.

Note, though, that this definition of [KP95] makes no special requirements of the function T. For example, an easy-to-solve problem Π (say, solvable from scratch by an algorithm of complexity log-logarithmic in n) may be considered fault-locally mendable if it is fault-locally T-mendable for a highly costly function T (say, doubly exponential in $|F|$).

In [KP95] it is shown that the actual situation is not all that bad for global (linear) functions. First, *every* (computable) problem is fault locally mendable. Moreover, every problem is fault locally $c|F| \log |F|$-mendable for a small constant c. Hence the complexity of mending is close to linear in $|F|$.

Fault-local mending may make a lot of sense *especially* for local functions, since faults are very often of extremely *local* nature and involve only a small number of hosts. (For example, the famous crash of the ARPANET, Internet's former incarnation, was caused by a single node giving all other nodes wrong routing information [R81].) Moreover, systems reliability has been increasing, so the number of faults grows much more slowly than the network sizes, especially in domains (or localities) of the network that employ high security and quality standards.

Yet, the motivating observation of the current paper is that for easy-to-solve (or sublinear) local problems, fault-local mending can still be very bad in the worst case. As our illustrative example, we take Π to be the problem of computing the MIS of the graph. MIS enjoys a randomized logarithmic time algorithm [L86]. Recomputing the MIS from scratch using Luby's algorithm will only take logarithmic time, whereas

the running time of the mending algorithm of [KP95] will be exponentially higher if $|F|$ is large. Hence for such sublinear problems we may hope to be able to do much better, although this may require a more careful definition as well as a more elaborate algorithm.

Toward this goal, we now introduce the new notion of *tight* fault-local mending. If the cost of computing Π from scratch on an n-vertex network is $\Omega(T(n))$ and Π is fault locally T-mendable, then we say that Π is *tightly locally mendable* (or simply *tightly mendable*). If Π is only fault locally $poly(T)$-mendable, then we say that Π is *near-tightly locally mendable*. For randomized algorithms we use analogous terminology. In particular, if $T(n)$ is the complexity of a randomized algorithm for computing Π, then we say that Π is randomly locally T-mendable. The notion of tightness is defined similarly.

We should point out two inherent limitations of any kind of mending, or recovery in general (including, of course, the fault-local approach). First, note that the precise value of the original function $\mathcal{V}$ may not be recoverable, since we do not necessarily know the fault configuration, namely, which nodes suffered faults, or even how many faults occurred. (In fact, it is possible for the faults to change the global state to look precisely as some other legal solution $\mathcal{V}'$, in which case no problem will ever be detected!) Note that this limitation applies, of course, to any kind of correction.

While considered here as a limitation, it should be pointed out that viewed from another angle, our definition of *tight* fault locality implies a property that looks stronger than the one stated. This property can serve as another way to view the difference between fault locality and tight fault locality.

Consider a fault locally T-mendable problem Π. Suppose that we started from a legal solution $\mathcal{V}$ and through failures at a set of nodes F reached an illegal function $\mathcal{V}'$. As discussed above, there is no way for the mending algorithm to know which nodes suffered faults or what the original solution was. It follows that the mending algorithm is forced to mend $\mathcal{V}'$ to the legal solution $\mathcal{V}''$ *closest* to $\mathcal{V}'$; otherwise it might be too costly if $|F|$ were small. This may in fact be cheaper in some cases than returning to the original $\mathcal{V}$. The definition of a "close" solution for tightly fault local algorithm is different (stronger) than previous definitions.

Put more formally, let $d(\mathcal{V}, \mathcal{V}')$ denote the Hamming distance between the two solutions $\mathcal{V}$ and $\mathcal{V}'$, namely, the number of nodes on which they disagree:

$$d(\mathcal{V}, \mathcal{V}') = |\{v \mid \mathcal{V}(v) \neq \mathcal{V}'(v)\}|.$$

For a problem Π, a graph G and a function $\mathcal{V}'$, let $\mathit{diff}(\mathcal{V}', \Pi, G)$ denote the distance of $\mathcal{V}'$ from $\Pi(G)$:

$$\mathit{diff}(\mathcal{V}', \Pi, G) = \min\{d(\mathcal{V}', \mathcal{V}) \mid \mathcal{V} \in \Pi(G)\}.$$

Clearly, if $\mathcal{V}'$ is obtained from $\mathcal{V} \in \Pi(G)$ through failures in a set of nodes F, then $\mathit{diff}(\mathcal{V}', \Pi, G)$ is bounded from above by $|F|$. Our observation implies that the complexity of mending a given faulty solution $\mathcal{V}'$ of Π must thus be $O(T(\mathit{diff}(\mathcal{V}', \Pi, G)))$, even if, in fact, the number of faults $|F|$ is much larger than $O(T(\mathit{diff}(\mathcal{V}', \Pi, G)))$. This is due to the fact that we cannot eliminate the possibility that indeed $\mathcal{V}''$ was the original solution, rather than $\mathcal{V}$, so we must assume the case that allows us the smallest cost for correction. Note that this does not always imply that the number of nodes whose state can be changed by the mending algorithm is $O(T(\mathit{diff}(\mathcal{V}', \Pi, G)))$, since changing the states of a large number of nodes may not take a large amount of time, e.g., in the case that they are "packed" in a small neighborhood.

Considered from this point of view, our tightly fault-local mending techniques can be thought of as facilitating an approach defined formally and promoted in [DH95] as a basic goal of any algorithm for adjusting to topological changes and faults. That paper suggests the idea that the recovery algorithm should bring the system from its currently faulty global state to a "closest" global state. However, the definition of closeness in [DH95] is somewhat different; they count the *number* of nodes by which the states differ, rather than the *complexity* (as a function of that number) of reaching from one state to the other. Note also that achieving the goal of [DH95] does not imply a fault-local algorithm and thus, of course, it does not imply a tightly fault-local algorithm. For example, the algorithm presented there for achieving closeness performs a global computation even in response to a single change and is thus *not* fault local.

The second inherent limitation of fault-local mending has to do with the specific representation of the solution function $\mathcal{V}$ in the network. A crucial observation is that if the function is represented *minimally*, then fault-local mending may be impossible (see [KP95]). Hence any solution approach must be based on making use of additional data structures at the various nodes, typically storing information about the local states of their neighbors. It should be clear that the use of such data structures does not by itself suffice to solve the problem. In fact, it may increase its complexity, since any additional data stored at the nodes of the system is just as prone to erasure or distortion due to faults as is the basic data.

1.2. Contributions. In this paper, we define the concept of *tight fault locality* to better capture the desired performance of a mending algorithm for sublinear functions. We focus mainly on the MIS problem and examine its fault-locality properties. An MIS of G is a maximal set $M \subseteq V$ of nodes such that no two nodes in M are neighbors. This is a degenerate case of an input/output function, which has no input. That is, a solution $\mathcal{V}$ is actually a Boolean function of the vertices whose value is $\mathcal{V}(v) = 1$ if v belongs to the MIS, and 0 otherwise, and given a graph G, the set of legal solutions $MIS(G)$ contains precisely those functions $\mathcal{V}$ that represent an MIS of G. Transient faults may change some 0's to 1's (such that the resulting set $\mathcal{V}'$ is no longer independent) and some 1's to 0's (such that the resulting set is no longer maximal).

The MIS problem has been intensively studied before in the context of understanding the nature of sublinear distributed algorithms. A number of sublinear algorithms exist for it, and hence it is a natural candidate for demonstrating the concepts and transformation technique presented in this paper. Moreover, MIS is of special interest in the context of fault-local mending due to the fact that it is used as a paradigm for several purposes in a distributed system. One example is for scheduling access to nearby resources [L80] (a generalization of the drinking philosophers problem [CM84]), e.g., a communication channel. It models a setting in which whenever one node accesses a resource, its neighbors are prevented from using that resource.

Our results concerning MIS are the following. We present a generic mending algorithm for MIS, named $Mend$. This algorithm employs an MIS construction procedure P as a subroutine, and we denote the resulting algorithm by $Mend[P]$. The properties of this algorithm depend on the particular procedure chosen. First, employing the randomized algorithm of [L86], MIS_L, as our MIS procedure, we demonstrate that the MIS problem is randomly locally $\log|F|$-mendable. That is, if only $|F|$ faults occurred, then the expected running time of the randomized mending algorithm, $T_{Mend[L]}$, is only $O(\log|F|)$ rather than $O(\log n)$. Thus the (randomized expected) complexity of

locally mending MIS is as good as that of the best (randomized expected) algorithm known for distributively computing MIS from scratch.

Since we have no nontrivial lower bound on the (randomized or deterministic) complexity of distributed MIS, the above result serves only to show that MIS is randomly *near*-tightly locally mendable. Actually, we show a more general result, namely, that the existence of any randomized MIS algorithm whose time complexity is a "reasonably nice" function T (to be defined precisely later) implies that MIS is randomly locally T-mendable. This implies, in particular, that if the true randomized complexity of MIS is such a nice function, then MIS is randomly tightly locally mendable.

As for deterministic mending we show similar results. Specifically, relying on the deterministic algorithm of [PS92], MIS_{PS}, as our MIS procedure, we show that MIS is fault locally $2^{O(\sqrt{\log |F|})}$-mendable; i.e., the expected running time of the mending algorithm with this procedure, $Mend[PS]$, is $T_{Mend[PS]} = 2^{O(\sqrt{\log |F|})}$. Again, we show a general result stating that the existence of any deterministic MIS algorithm with "reasonably nice" time complexity T implies that MIS is fault locally $poly(T)$-mendable. This implies, in particular, that if the true deterministic complexity of MIS is such a nice function, then MIS is near-tightly locally mendable.

The methods proposed here for mending MIS can be applied also for obtaining near-tight mending algorithms for a number of other problems, such as coloring and scheduling problems.

Note that the results listed above establish with certainty only the existence of *near*-tightly locally mendable problems. (Of course, problems that can be checked and computed in $O(1)$ time are trivially tightly mendable. Such problems are presented in [NS93].)

1.3. Related work. The problem of distributed fault correction has been the subject of much research in the area of *dynamic networks*, where the most common type of faults is the crash of a communication link. An approach with mostly theoretical appeal was to run a global "reset" protocol that enables restarting the computation from scratch [F79, AAG87]. Another approach, more common in practice, is to disseminate every piece of local information globally, so that any global function (e.g., routing) can be corrected (when faults or changes occur) by every node, simply by computing the new value from the known global information [MRR80, CGKK95]. Note that in both approaches, every node must participate in the modification process for every topological change, e.g., the addition or the crash of a single node; hence their worst-case complexity is not very attractive. However, when the number of changes is small, the update approach may lead to considerable savings in communication for many problems, since most of the information does not need to be redistributed. It is argued [ACK90] that the reset approach can be avoided, since other approaches can be made more efficient even in the worst case.

In [ACK90], a spanning tree is maintained rather than recomputed. When a tree edge fails, the algorithm replaces it by a single other tree edge (if possible), keeping the rest of the tree intact. This property is called there *path preservation*, and it is argued that it keeps the routing on the tree intact for every route that does not use the failed edge. Similar examples appear in [CP87, CK85, NS93, MNS95, DH95]. In [DH95] this idea was generalized and formalized as a requirement that an illegal global state is corrected to a global state that is the "closest." A general global algorithm (*superstabilization*) is presented there to perform this task for any problem if the number of detectable faults is precisely 1. This correctness notion, however,

is not related to complexity: the *superstabilizing* general algorithm given therein has an $\Omega(diameter)$ running time even when only one fault occurs (hence it is not fault local). Thus, the intuition captured by [DH95] may prove a good generalization for the *path preservation* (i.e., formalizing the intuitive notion of small disruption), and fault locality formalizes another intuitive notion—that of efficiency. On the other hand, the definition of *tight fault locality*, introduced in the current paper, attempts to capture both of these intuitive notions.

All of the previous approaches mentioned above are more suitable for global (linear) problems; and even for those problems they (with the exception of the fault local approach of [KP95]) consume at least $\Omega(diameter)$ time, even for a small number of faults. This means that even one fault can cause a global computation. Moreover, these approaches (especially reset) are mainly suitable for a distributed system that cannot produce "useful work" when some of its nodes suffer a transient fault. A more dynamic solution to the MIS problem, in which the system corrects itself as locally as possible, letting undamaged regions of the system operate as usual in the meantime, appears in [ACS94]. However, the running time of that correction algorithm is the same as that of recomputing the function from scratch. On the other hand, the algorithm in [ACS94] is based on less synchrony assumptions than those used in this paper; in particular, it is not assumed there that all the nodes start the algorithm at the same time or that the network is synchronous.

A generalization of dynamic networks to deal with self-stabilization is proposed in [AKY90]. The reset algorithms in this context (e.g., [AKY90, APV91, AV91, AK93, APVD94]) suffer from the same disadvantages mentioned above. Other models deal with other (sometimes even stronger) types of faults, or try to treat several kinds of faults simultaneously [GP93, DH95]. Other models studied in the literature allow efficient algorithms through imposing various restrictions; e.g., only one fault may occur at a time, or the network is a complete graph, etc.

2. Preliminaries.

2.1. Model and definitions. We model a distributed system as a graph $G = (V, E)$, where V is the set of the system (or network) nodes, $|V| = n$, and E is the set of links. Nodes communicate by sending messages over the links. The network is synchronous; that is, communication proceeds in rounds, where in every round each node receives all the messages sent to it from its neighbors in the previous round, and sends messages to some of its neighbors. Moreover, we assume that all faults occur simultaneously at time t_0. To focus on time complexity, we adopt the model employed in previous studies of locality issues [GPS87, L92], in which message complexity is abstracted away by allowing the transmission of arbitrary size messages in a single time unit. (Our messages are nevertheless not very large.)

We consider the problem of mending an MIS of a given graph G. The MIS is represented distributively by a vector of bits $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_n})$, with each node v storing its own bit $\mathcal{M}_v$, such that $\mathcal{M}_v = 1$ iff v is in the MIS. Denote by $\mathcal{S}(\mathcal{M}) = \{v \mid \mathcal{M}_v = 1\}$ the set induced by $\mathcal{M}$. Hereafter, we shall occasionally use $\mathcal{M}$ for the MIS $\mathcal{S}(\mathcal{M})$, where the intention is clear from the context.

Our approach allows a more general representation, in which in addition to the output bit $\mathcal{M}_v$ each node v may maintain an additional data structure $\mathcal{D}_v$, which is a function of $\mathcal{M}$ and the particular algorithm. Note that $\mathcal{M}$ is a part of the definition of the problem, whereas different algorithms may use different definitions (and different value ranges) for $\mathcal{D}_v$.

We distinguish two important global states: the one just before the faults and the one just after. Let $\mathcal{M}_v^*$ be the representation of the MIS before the faults, and let $\mathcal{M}_v$ be the current representation of the MIS (after the faults). Likewise, let $\mathcal{D}_v^*$ be the data structure stored at v before the faults, and let $\mathcal{D}_v$ be the current structure stored at v. Thus $\mathcal{M}_v$ and $\mathcal{D}_v$ are the inputs for the mending task. (Clearly, $\mathcal{M}_v^*$ and $\mathcal{D}_v^*$ are not known to v.)

Formally, the set F of faulty nodes and the set H of healthy nodes are defined, respectively, as

$$F = \{v \mid \mathcal{M}_v \neq \mathcal{M}_v^* \text{ or } \mathcal{D}_v \neq \mathcal{D}_v^*\} \quad \text{and} \quad H = \{v \mid \mathcal{M}_v = \mathcal{M}_v^* \text{ and } \mathcal{D}_v = \mathcal{D}_v^*\}.$$

If the resulting set $\mathcal{S}(\mathcal{M})$ is not an MIS on a given graph, then there are two possible types of violations. A node v is said to be *uncovered* if neither v nor any of its neighbors are in $\mathcal{S}(\mathcal{M})$. A node v is said to be *collided* if both v and one of its neighbors are in $\mathcal{S}(\mathcal{M})$.

DEFINITION 2.1. *The* conflict set *of the assignment $\mathcal{M}$, denoted $\mathcal{C}(\mathcal{M})$, is the set of nodes that can detect locally that $\mathcal{S}(\mathcal{M})$ is not a legal MIS, namely, the uncovered nodes and the collided nodes.*

Finally, we need a few definitions concerning the immediate surroundings of sets of nodes. Let $\Gamma_i(v)$ be the set of all nodes whose distance from v in G is less than or equal to i. For a set of nodes W, let $\Gamma_i(W) = \bigcup_{w \in W} \Gamma_i(w)$. The *complement* of a set of nodes $U \subseteq V$ is defined to be

$$Co(U) = V \setminus U.$$

The *i-border* of the set U is defined to be the set of nodes in the i most external layers of U or, formally,

$$Border_i(U) = \Gamma_i(Co(U)) \cap U.$$

$Border_1(U)$ is denoted simply $Border(U)$.

2.2. Utilizing MIS procedures. To date, the precise complexity of distributed MIS is not known. Hence to cast our results in as general a form as we can, we describe our mending algorithm using any distributed MIS procedure as a black box. We are interested in showing that MIS is tightly locally mendable (or at least near-tightly locally mendable). We can prove that subject to the assumption that the complexity of the optimal algorithm is a "reasonably nice" function. To establish that, we show that the existence of any MIS algorithm whose time complexity is a "reasonably nice" function T implies that MIS is fault-locally T-mendable in the randomized case and fault-locally $poly(T)$-mendable in the deterministic case.

More precisely, by a "reasonably nice" complexity function we mean the following. A function $T : N \mapsto N$ is said to be *sublogarithmic* if it is nondecreasing and positive for $n \geq 1$, and in addition it satisfies the following condition:

$$\text{(SL1)} \qquad \forall x, y, \qquad T(xy) \leq T(x) + T(y).$$

The function T is *superlogarithmic* if instead of (SL1), it satisfies

$$\text{(SL2)} \qquad \forall x, y, \qquad T(xy) > T(x) + T(y).$$

Note that the log function is sublogarithmic and, in fact, most complexity functions likely to be used are either sublogarithmic or superlogarithmic. (Our results may

possibly hold also for functions that are neither sublogarithmic nor superlogarithmic, but this is left for further research.)

First consider deterministic MIS procedures. Our definition of "reasonable" complexity allows any deterministic MIS procedure MIS_D whose complexity $T_{MIS_D}(n)$ is either sublogarithmic or superlogarithmic. This applies, in particular, to the currently best known procedure, due to [PS92] (whose complexity is superlogarithmic).

LEMMA 2.2 (see [PS92]). *There exists a deterministic distributed algorithm MIS_{PS} for computing an MIS on an n-vertex graph in time $T_{MIS_{PS}}(n) = 2^{O(\sqrt{\log n})}$.*

(Note that $2^{O(\sqrt{\log n})}$ is asymptotically larger than any polylogarithmic function in n but smaller than n^{ϵ} for any $\epsilon > 0$.)

In our mending algorithm, we invoke an MIS procedure only for a prescribed number of steps τ, on some region G' of the graph, of size n'. If $\tau \geq T_{MIS_D}(n')$, then the procedure will indeed halt with the correct output. If this requirement does not hold, then we assume that the output is arbitrary.

We now turn to randomized MIS procedures. Since the best known randomized MIS algorithm is of expected time complexity $O(\log n)$, we need only consider procedures of sublogarithmic complexity. Consequently, we shall concentrate on randomized Las Vegas type MIS procedures MIS_R whose expected time complexity $T_{MIS_R}(n)$ is sublogarithmic. Relying on Markov's inequality, we will in fact make use of a corresponding Monte Carlo variant of the procedure, $MIS_R(\tau)$, in which the procedure is halted after τ steps. Based on well known techniques, it is easy to show the following properties.

LEMMA 2.3. *Given a randomized Las Vegas MIS procedure MIS_R of expected time complexity $T_{MIS_R}(n)$, there exists a constant $\tilde{c} \geq 1$, such that whenever executing the corresponding Monte Carlo variant $MIS_R(\tau)$ with $\tau \geq \tilde{c} \cdot T_{MIS_R}(n)$, the output is a legal MIS with probability* 3/4.

We point out that our randomized mending algorithm is nevertheless a Las Vegas algorithm. Also, note that again our requirement applies in particular to the currently best known randomized procedure due to [L86].

LEMMA 2.4 (see [L86]). *There exists a randomized distributed algorithm MIS_L for computing an MIS on an n-vertex graph in expected time $T_{MIS_L}(n) = O(\log n)$.*

3. Overview of the solution. A first naive solution one may suggest for the problem of mending an MIS assignment is to have the nodes in the conflict set $\mathcal{C}(\mathcal{M})$ run an MIS protocol, without involving the rest of the nodes of the graph. The problem with this approach is that, even assuming it is possible to efficiently correct $\mathcal{M}$ in such a way,[1] this alone cannot guarantee that the complexity of the mending process is bounded as a function of the number of corrupted nodes, since the conflict set might be much larger than the set of corrupted nodes. Consequently, in our solution every node stores information about its state at neighboring nodes. The obvious difficulty is that the data stored at a node for its neighbors might get corrupted just as easily as its own data. Moreover, simple voting schemes (if the node consults many neighbors) run into difficulties. See [KP95, LPRS93, P96].

We start by presenting a simplified solution for the case that $|F|$ is known in advance. First we describe the local data structures $\mathcal{D}_v$ employed in our solution. In addition to the output bit $\mathcal{M}_v$, each node v will store a bit $\mathcal{D}_v(u) = \mathcal{M}_u$ for each node u whose distance from v in the graph is 2 or less. Formally, $\mathcal{D}_v = \{\mathcal{D}_v(u) \in \{0,1\} \mid u \in \Gamma_2(v)\}$.

[1] It is probably not, since it entails solving the problem of MIS completion, which is NP-Hard.

Given $|F|$, partition the nodes into two sets: a set `Big` consisting of the nodes v whose distance-2 neighborhood $\Gamma_2(v)$ contains more than $2|F|$ nodes, and a set `Small` containing the rest. Each of these sets will be treated separately.

The output of a node v in `Big` is determined by a *majority vote* among its 2-neighbors (including itself), based on the value stored for v in their data structures. If the result of this vote is different than $\mathcal{M}_v$, then v changes its $\mathcal{M}_v$ accordingly. (The reasons for which we look at the distance-2 neighborhood of nodes rather than simply their direct neighborhoods will become clear later.) Note that following this vote the original correct value of $\mathcal{M}^*_v$ for any v in `Big` is recovered.

Now consider all the nodes in `Small`. Some of them may now belong to the conflict set $\mathcal{C}$. Denote this subset by $\hat{C} = \mathcal{C} \cap \texttt{Small}$. Notice that it is not enough to run an MIS algorithm for the nodes in $\hat{C}$ only, since some of them have neighbors in the complement $Co(\hat{C})$, and a combination of an arbitrary new MIS in $\hat{C}$ and the original MIS $\mathcal{M}$ in $Co(\hat{C})$ does not necessarily form a legal MIS. One would thus like to select a special kind of MIS in $\hat{C}$, namely, one that is constrained by the values of the original MIS $\mathcal{M}$ in $Co(\hat{C})$. More specifically, a node v in $\hat{C}$ whose neighbor w in $Co(\hat{C})$ has $\mathcal{M}_w = 1$ is not allowed to output $\mathcal{M}_v = 1$, and likewise, uncovered nodes in $Co(\hat{C})$ may impose a 1 value on their neighbors in $\hat{C}$.

Unfortunately, solving such a constrained MIS even by a sequential algorithm is NP-Hard. To overcome this difficulty, we resort to a more relaxed method of fusing the new MIS constructed for $\hat{C}$ to the existing one on $Co(\hat{C})$. We first employ a procedure that selects a subset L of some of the nodes in $Co(\hat{C})$ that are at distance 1 or 2 from $\hat{C}$. For L produced by this procedure, the original MIS $\mathcal{M}$ restricted to the resulting set $Co(\hat{C} \cup L)$ has the nice property that it is "shielded" from the influence of fusing to it a new MIS assignment on $\hat{C} \cup L$, no matter what MIS values are chosen for $\hat{C} \cup L$. More specifically, consider a node v in $Co(\hat{C} \cup L)$ that borders with $\hat{C} \cup L$. Then in the MIS $\mathcal{M}$ defined on $Co(\hat{C} \cup L)$, the value of v is $\mathcal{M}_v = 0$, and, moreover, v has a neighbor u in $Co(\hat{C} \cup L)$ such that $\mathcal{M}_u = 1$. We say that $\mathcal{M}$ is a *shielded MIS* on $Co(\hat{C} \cup L)$. Therefore, as we now compute a new MIS assignment on $\hat{C} \cup L$, it is easy to see that no matter what this MIS assignment is, its fusing together with $\mathcal{M}$ in $Co(\hat{C} \cup L)$ yields a legal MIS in G.

As for time complexity, the execution time of this protocol is bounded by ensuring that the size of $\hat{C} \cup L$ is $O(|F|)$. (The dependence on the particular MIS procedure used for computing the new MIS assignment on this region is discussed in the next subsection.) For bounding $|L|$, we rely on the fact that L consists of nodes that are in the 2-neighborhood of nodes in $\hat{C}$. This ensures that $|L| = O(|F|)$ as a result of our choice to include in $\hat{C}$ only nodes with small 2-neighborhoods. This is an additional reason for treating nodes with large 2-neighborhoods separately, through direct majority vote.

We now turn to the general case, where it is not assumed that $|F|$ is known. In this case, our algorithm runs in phases, performing a search on the number of faults $|F|$. In phase i it is "guessed" that the number $|F|$ of faults is not larger than some λ_i. The precise definition of λ_i, given later, depends on the complexity of the MIS procedure used and is aimed at preventing the number of phases from appearing as a multiplicative factor in the complexity of the algorithm.

Several things can go wrong when the guess is too small. One minor problem is that since F is larger than guessed (in the first few phases), the size of the conflict set $\mathcal{C}_i$ may also be larger than anticipated; hence, the MIS computation on $\mathcal{C}_i \cup L_i$ may be too long. An easy way to prevent that from happening is to execute the

MIS computation in the i's phase only for the time that would have been needed for this computation had the size of $\mathcal{C}_i$ indeed been λ_i. That is, phase i is run for $O(T_{MIS_R}(\lambda_i))$ time in the randomized case or $O(T_{MIS_D}(\lambda_i))$ in the deterministic case.

A more severe problem is that the computations carried out in the first phases may increase the number of "currently faulty" nodes. That is, denoting by F_i the set of nodes v whose MIS value $\mathcal{M}_v$ at the beginning of the ith phase is different from their original value $\mathcal{M}_v^*$, while $F_1 = F$, the set F_2 is affected by the outcome of phase 1: The assignment of new MIS values to the nodes of the conflict set $\mathcal{C}_1$ in phase 1 may cause many nodes whose original value was correct to erroneously change their value, thus making the set F_2 much larger than F and causing many new conflicts to show up in phase 2 and so on. This is the case even if phase 1 is terminated after the fixed time bound.

The only way to completely wipe out the influence of the previous phase would be to make all participating nodes return to their initial values. Indeed, this operation is applied at nodes that decide at the end of a phase to participate in subsequent phases (using the initial values, saved in variables $\mathcal{M}_v^{sav}$). But applying such a reinitialization operation globally may be highly time consuming, since it may happen that the assignment in the neighborhood of some nodes in the conflict set $\mathcal{C}_i$ is consistent at the end of the phase, and hence these nodes are ready to terminate, while some other nodes observe inconsistencies, implying the necessity of another phase. Therefore, reinitialization of all nodes requires broadcast of a message from the latter nodes to the former, which may be time consuming.

It is therefore necessary to ensure two things: first, that the algorithm is able to detect that the guess was wrong, and, second, that the operations of the algorithm under the wrong guess did not cause "too much damage." We achieve this behavior by attempting to prevent seemingly faulty nodes from participating in the voting process and gaining undue influence. Toward that end, we introduce one additional "screening" step before performing our votes. The idea is that nodes with a suspiciously high number of conflicts with neighbors (where "high" here is relative to the phase) will be "guessed" faulty and consequently will be barred from participating in the voting. It is important to note that despite the possibility of nonvoters, the votes carried at the nodes of `Big` will still require a strict majority, namely, more than half of the entire 2-neighborhood (not just of the voters).

4. MIS completions. Formally, the problem of *relaxed MIS completion* requires us to compute an MIS, given a partial MIS assignment. It is permitted to change any part of the partial MIS assignment, as long as the end result is a legal MIS. Note that the only difference between this problem and the problem of MIS is that the complexity of this problem may be lower, due to the given partial information. We comment that in our specific solution, the only values of the given partial MIS assignment U that may be changed are at the set $Border_2(U)$.

4.1. Shielded MIS, fringed MIS, and shielded kernels. We now describe a distributed procedure Rel_Comp$(G, W, \mathcal{M}, \tau)$ that, given a graph G, a subset W of the vertices, a partial MIS assignment $\mathcal{M}$ on W (possibly with some uncovered nodes on the border of W), and a time bound τ, computes a relaxed completion of $\mathcal{M}$ on the rest of the graph (with minimal penetration to the region of W). The procedure makes use of an MIS procedure, which can be either deterministic or randomized and is applied only for τ steps.

Definition 4.1. *Given a graph $G = (V, E)$, a subset $U \subseteq V$, and an MIS $\mathcal{R}$ on the subgraph induced by U, we say that $\mathcal{R}$ is a* Shielded MIS *for U if it contains no*

1. Initialize U to be W.
2. Let $X = \{v \in \mathit{Border}(U) \mid \mathcal{M}_v = 1\}$, and eliminate the nodes of X from U.
3. Let $Y = \{v \in U \mid v \text{ is uncovered}\}$, and eliminate the nodes of Y from U.
4. Define $\mathcal{R} = \{v \in U \mid \mathcal{M}_v = 1\}$.
5. Output $(\mathcal{R}, U)$.

FIG. 1. *Procedure* SHIELD_MIS$(G, W, \mathcal{M})$.

nodes from Border(U).

DEFINITION 4.2. *Given a graph $G = (V, E)$, a set $W = \{v_1, \ldots, v_k\} \subset V$, and an* independent set *assignment $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_k})$ on W, we have that the set $\mathcal{S}(\mathcal{M})$ is a* fringed-MIS *of W if it is maximal everywhere except possibly on its border; namely, the only violations that prevent it from being a legal MIS involve nodes v in Border(W) that are uncovered.*

This definition is motivated by the fact that our mending problem leads to situations where some of the MIS was ruined, yet other segments remain intact. These segments now form a fringed-MIS, since although no changes have occurred on these nodes themselves, their border nodes may become uncovered due to changes in their neighbors.

DEFINITION 4.3. *Given a graph $G = (V, E)$ and a subset $W = \{v_1, \ldots, v_k\} \subset V$, a* shielded kernel *for W is a pair of subsets $(\mathcal{R}, U)$ such that*

1. *$\mathcal{R} \subseteq U \subseteq W$,*
2. *U contains (at least) all the internal layers of W up to the last two layers, namely, $W \setminus \mathit{Border}_2(W) \subseteq U$, and*
3. *$\mathcal{R}$ is a shielded MIS for U.*

Figure 1 describes a procedure SHIELD_MIS that, given a graph G, a set of nodes W, and a fringed-MIS $\mathcal{M}$ on W, generates a shielded kernel $(\mathcal{R}, U)$ for W. Figure 2 illustrates the operation of the procedure. (The procedure is sequential; we discuss its distributed implementation later.)

LEMMA 4.4. *Procedure* SHIELD_MIS *provides a shielded MIS for the given problem and requires $O(1)$ time for a distributed implementation.*

Proof. Property 1 in the definition of shielded MIS is guaranteed trivially by the procedure. For property 2, we need to argue that every node that is eliminated from U belongs to the external two layers of W. For the first elimination step, step 2, this is immediate. For the second elimination step, step 3, the claim follows from the fact that $\mathcal{M}$ is a fringed MIS; hence initially it could have uncovered nodes only on its border, and step 2 could introduce new uncovered nodes only in one of the two external layers of W.

It remains to prove property 3. We first argue that $\mathcal{R}$ forms an MIS on U. Since $\mathcal{S}(\mathcal{M})$ is a fringed MIS on W, and no new 1-valued nodes were introduced, there are no collision violations in $\mathcal{R}$. Step 3 directly guarantees that U has no uncovered nodes. Hence $\mathcal{R}$ is an MIS on U.

Finally, we need to argue that $\mathcal{R}$ contains no nodes from U's border. Assume, to the contrary, that $\mathcal{R}$ contains some node v from the border of U. If v was also on the border of W, then it should have been erased from U in step 2. Therefore, necessarily v was an internal node of W and was brought to the border of U due to the elimination of some of its neighbors. But step 2 eliminates only nodes v with $\mathcal{M}_v = 1$, none of which could have been a neighbor of v (since otherwise $\mathcal{M}$ contains a collision), and step 3 eliminates only nodes whose neighbors w in U all have $\mathcal{M}_w = 0$;

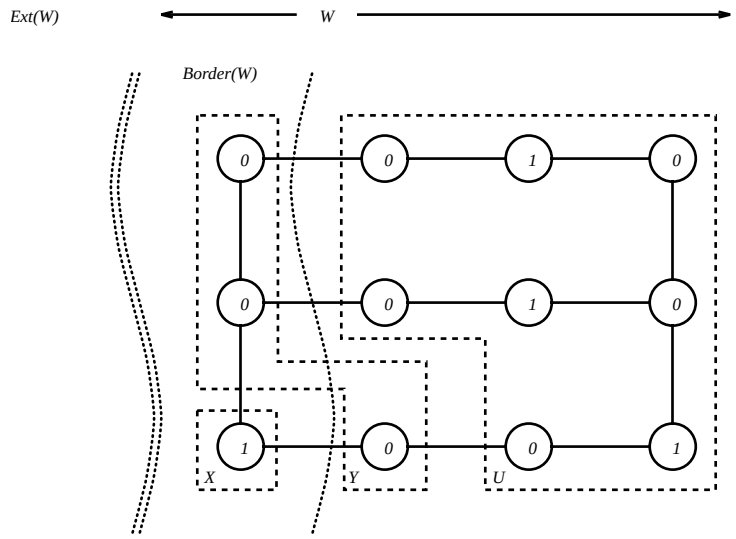


FIG. 2. *The operation of Procedure* SHIELD_MIS *on a set W with a fringed-MIS $\mathcal{M}$. ($\mathcal{M}$ is not an MIS because the upper left corner node is uncovered.)*

1. Applying Procedure SHIELD_MIS$(G, W, \mathcal{M})$, obtain a shielded kernel $(\mathcal{R}, U)$ for W.
2. Let $Q = Co(U)$.
3. Apply an MIS procedure to the subgraph induced by Q for τ steps, and get an MIS $\mathcal{R}'$.
   This can be a deterministic procedure MIS_D or a Monte Carlo type procedure $MIS_R(\tau)$.
4. Output $\mathcal{R} \cup \mathcal{R}'$ as an MIS for the entire graph.

FIG. 3. *Procedure* REL_COMP$(G, W, \mathcal{M}, \tau)$.

hence again none of them could be a neighbor of v, a contradiction. □

4.2. Using shielded MIS for relaxed MIS completion. We now use the procedure given in the previous section for deriving a procedure REL_COMP for relaxed completion of a given fringed MIS. We are given a graph $G = (V, E)$, a subset $W = \{v_1, \ldots, v_k\} \subset V$, and a fringed-MIS assignment $\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_k})$ on W. Procedure REL_COMP is described in Figure 3, and its properties are stated in the following lemma.

LEMMA 4.5.

1. *Procedure* REL_COMP *reassigns MIS values only at the nodes of a set Q restricted to $Co(W) \cup Border_2(W)$.*
2. *If a deterministic MIS procedure MIS_D is used, and the time limit τ satisfies $\tau \geq T_{MIS_D}(|Q|)$, then procedure* REL_COMP *provides a relaxed MIS completion for $\mathcal{M}$.*
3. *If a randomized MIS procedure $MIS_R(\tau)$ is invoked, and the time limit τ satisfies $\tau \geq \tilde{c} \cdot T_{MIS_R}(|Q|)$ (for the constant $\tilde{c}$ of Lemma* 2.3*), then procedure* REL_COMP *provides a relaxed MIS completion with probability* 3/4.

4. *The time complexity of Procedure* Rel_Comp *is* $O(\tau)$.

Proof. The fact that $\mathcal{R}$ is a shielded MIS for U guarantees that it can be combined without conflicts with the MIS $\mathcal{R}'$ computed for the complement of U to yield an MIS for the entire graph. The time complexity of the procedure is composed of that of procedure Shield_MIS, which is $O(1)$, and the time complexity of the MIS procedure on Q, which is limited to τ steps. The bound on the failure probability in the randomized case follows from Lemma 2.3. □

4.3. Distributed implementation. While the description of the procedure Rel_Comp and its subprocedure Shield_MIS is given in "sequential" form, it is clear that both procedures have a straightforward distributed implementation. The only delicate point is that these procedures are to be initiated (as we shall see later) simultaneously not only by the nodes of the set W, but also by some nodes of its complement, $Co(W)$. This does not create any special difficulties, though, since the exact time duration for all the computations is known to all the nodes in the graph. Moreover, these initiators are neighbors of W, and thus the nodes in W which perform the procedure get the initiation messages within a constant time.

5. MIS correction algorithm. We are now ready to describe our mending algorithm for the MIS problem. The algorithm, *Mend*, is given below in Figure 4. The algorithm starts by saving the input values of $\mathcal{M}$ and the data structure $\mathcal{D}$ in variables $\mathcal{M}^{sav}$ and $\mathcal{D}^{sav}$. Let $\mathcal{D}_v^{sav}(v) = \mathcal{M}_v^{sav}$. These input values will be used from time to time throughout the execution. The variables $\mathcal{M}$ and $\mathcal{D}$ will be used for recording the output of the computation. Specifically, whenever a node v selects a new value for its MIS assignment, it stores it in $\mathcal{M}_v$ as well as in $\mathcal{D}_w(v)$ at every node $w \in \Gamma_2(v)$.

The algorithm makes use of a *record inconsistency graph* G_{RI} defined on V, whose edges identify pairs of nodes of which one must be faulty. The edges of this graph are determined on the basis of an inconsistency between the $\mathcal{M}$ value of a node and the recorded value for that node in the data structure stored at a neighbor. Formally, this graph is defined as follows.

Definition 5.1. *The (undirected)* record inconsistency *graph* G_{RI} *consists of the following edges. For every two nodes* $u, w \in V$*, the edge* (u, w) *is included in* G_{RI} *if one of the following conditions holds:*

1. $\mathrm{dist}(u, w) \leq 2$ *and* $\mathcal{M}_u^{sav} \neq \mathcal{D}_w^{sav}(u)$ *or vice versa,*
2. $\mathcal{D}_u^{sav}(x) \neq \mathcal{D}_w^{sav}(x)$ *for some common distance-2 neighbor* x *of* u *and* w *(i.e.,* $x \in \Gamma_2(u) \cap \Gamma_2(w)$*).*

Note that G_{RI} can be constructed distributively in constant time, as each node can find out which other nodes neighbor it in G_{RI}. For every node v, let $\deg_{RI}(v)$ denote v's degree in G_{RI}.

As outlined earlier, the algorithm conducts majority votes to determine the output of some nodes, using their neighbors' data structures. However, a node v with high $\deg_{RI}(v)$ (where "high" here is w.r.t. the current phase) is "guessed" to be faulty and is not allowed to vote. In addition, the algorithm maintains a set `Detected` of nodes that detected a local inconsistency in their own initial output (i.e., every node v such that $\mathcal{M}_v^{sav} = 1$ and $\mathcal{D}_v^{sav}(w) = 1$ for some neighbor w, as well as every node v such that $\mathcal{M}_v^{sav} = 0$ and $\mathcal{D}_v^{sav}(w) = 0$ for every neighbor w). Since the nodes in `Detected` are known to be faulty, they too do not vote.

In each phase i, the algorithm separates nodes with large neighborhoods from ones with small neighborhoods using a threshold λ_i. The sequence $\{\lambda_i\}$ will be fixed

on a global *start* signal **do**

1. **For** every node v, set $\mathcal{M}_v^{sav} \leftarrow \mathcal{M}_v$ and $\mathcal{D}_u^{sav}(v) \leftarrow \mathcal{D}_u(v)$ **for** every $u \in \Gamma_2(v)$.
2. **For** $i = 1$ to J_{max} **do** (* i.e., when $\lambda_i \leq n$ *)
   (a) **For** every node $v \in \mathtt{Big}_i$ that has a collision or uncoverage conflict with some neighbors (with the current $\mathcal{M}$ values), perform a vote on the value of $\mathcal{M}_v$ among the voting nodes in its 2-neighborhood as follows:
      i. **Let** $\mathcal{M}_v \leftarrow 0$ if $|\{w \in \mathtt{Vot}_i(v) \mid \mathcal{D}_w^{sav}(v) = 0\}| > \gamma(v)/2$.
      ii. **Let** $\mathcal{M}_v \leftarrow 1$ if $|\{w \in \mathtt{Vot}_i(v) \mid \mathcal{D}_w^{sav}(v) = 1\}| > \gamma(v)/2$.
      iii. **If** neither of the two votes has the majority (* due to nonvoters *)
      v marks itself and its neighbors in $\mathtt{Vot}_i(v)$ "nonparticipating" (and leaves $\mathcal{M}_v$ unchanged).
   (b) Let $\mathtt{NP}_i$ be the set of nonparticipating nodes.
   (c) **for** every participating node $v \in \mathtt{Small}_i \setminus \mathtt{NP}_i$ **do**:
   If v has a collision with a "participating" neighbor $w \in \mathtt{Big}_i$ (namely, $\mathcal{M}_v = \mathcal{M}_w = 1$), then let $\mathcal{M}_v = \mathcal{M}_v^{sav}$.
   (d) **Let** $\mathcal{C}_i = (\mathcal{C}(\mathcal{M}) \cap \mathtt{Small}_i) \setminus \mathtt{NP}_i$.
   (* Participating nodes that are in $\mathtt{Small}_i$ and in the conflict set induced by the current $\mathcal{M}$. *)
   (e) **If** $\mathcal{C}_i \neq \emptyset$ then the nodes of $\mathcal{C}_i$ **invoke** Procedure REL_COMP$(G, V \setminus \mathcal{C}_i, \mathcal{M}, \tau_i)$ for $\tau_i = T_{MIS_D}(\lambda_{i+3})$ in the deterministic case, and $\tau_i = \tilde{c} \cdot T_{MIS_R}(\lambda_{i+3})$ in the randomized case.
   (f) **for** every node v whose $\mathcal{M}_v$ has changed, **let** $\mathcal{D}_u(v) = \mathcal{M}_v$ **for** every $u \in \Gamma_2(v)$.
3. (* Handling the case that all phases failed. *)
   In the randomized case only: if any conflict still exists then invoke a Las Vegas variant of Algorithm MIS_R.

FIG. 4. *Algorithm Mend.*

later, as a function of the specific MIS procedure used. The only requirement imposed on the choice of this sequence is that it must be *quadratically growing*, namely, satisfy

$$\lambda_{i+1} \geq \lambda_i^2$$

(thus the length of the sequence is smaller than $\log \log n$). This choice also determines the value of J_{max}, the number of phases in the algorithm. (J_{max} is taken to be the first i such that $\lambda_i \geq n$.)

For the algorithm and analysis, we use the following definitions. Let

$$\mathtt{NV}_i = \{v \mid \deg_{RI}(v) > \lambda_i\} \cup \mathtt{Detected} \qquad \text{and} \qquad \mathtt{Vot}_i(v) = \Gamma_2(v) \setminus \mathtt{NV}_i$$

(standing, respectively, for "nonvoting" and "voters"). For every node v, let $\gamma(v) = |\Gamma_2(v)|$. Let

$$\mathtt{Big}_i = \{v \in V \mid \gamma(v) > 2\lambda_i\} \qquad \text{and} \qquad \mathtt{Small}_i = \{v \in V \mid \gamma(v) \leq 2\lambda_i\}.$$

6. Correctness and analysis of the main algorithm. We first describe the general structure of the proof. In Lemma 6.5 (using Lemma 6.4) we show that for a sufficiently large i, phase i computes a valid MIS. However, the complexity depends

on the size of the set F_i of "currently faulty" nodes whose output $\mathcal{M}_v$ at the beginning of phase i is different than the original value $\mathcal{M}_v^*$. Note that these could be either faulty nodes or healthy nodes that erroneously computed a new $\mathcal{M}_v$ during previous phases. In Lemmas 6.1 and 6.2 we show that the number of such nodes F_i in phase i is not much larger than the number of faults F. First, in Lemma 6.1 we show that the number of nodes participating in computing an MIS in phase i is not much larger than the number of (low-degree) nodes in F_i. Then in Lemma 6.2 we show that the number of (low-degree) nodes in F_{i+1} is not much larger than the number of nodes that participated in the MIS in phase i.

LEMMA 6.1. *In any phase i, the set $\mathcal{C}_i$ satisfies $|\mathcal{C}_i| = O(|F| + |F_i \cap \texttt{Small}_i| \cdot \lambda_i)$.*

Proof. Consider a phase i. Partition the set $\mathcal{C}_i$ defined in step 2d into $\mathcal{C}_F = (F \cup F_i) \cap \mathcal{C}_i$ and $\mathcal{C}_H = (H \cap H_i) \cap \mathcal{C}_i$. Clearly, $|\mathcal{C}_F|$ is bounded by $|F| + |F_i \cap \texttt{Small}_i|$, since $\mathcal{C}_i \subseteq \texttt{Small}_i$. It is therefore sufficient to bound $|\mathcal{C}_H|$.

The notion of *accusation* is defined as follows. Each node in conflict must accuse some node. In particular, for every $v \in \mathcal{C}_i$, the set of nodes accused by v, denoted $\texttt{Accuse}(v)$, is $\{w \mid (v,w) \in E, \mathcal{M}_v = \mathcal{M}_w = 1\}$ if v is collided, and $\{w \mid \mathcal{D}_v^{sav}(w) = 1\}$ if v is uncovered.

We now argue that $\emptyset \neq \texttt{Accuse}(v) \subseteq F_i$ for every node $v \in \mathcal{C}_H$. To see that $\texttt{Accuse}(v)$ is nonempty, for an uncovered v, note that since v is in both H and H_i, necessarily $\mathcal{M}_v^* = \mathcal{M}_v^{sav} = \mathcal{M}_v = 0$, and v's records $\mathcal{D}_v^{sav}$ are identical to the original $\mathcal{D}_v^*$. Hence these records must show a neighbor w such that $\mathcal{D}_v^{sav}(w) = 1$; hence $w \in \texttt{Accuse}(v)$. The argument for a collided v is similar. To see that each node w accused by v is in F_i, note that since $v \in H \cap H_i$, $\mathcal{D}_v^{sav}(w)$ agrees with $\mathcal{M}_w^*$, and hence $\mathcal{M}_w \neq \mathcal{M}_w^*$; hence $w \in F_i$.

Consequently, partition the set $\mathcal{C}_H$ into two sets, the set ACC_BIG_i of nodes $v \in \mathcal{C}_H$ that accuse some neighbor $w \in F_i \cap \texttt{Big}_i$, and the set ACC_SMALL_i of nodes $v \in \mathcal{C}_H$ for which $\texttt{Accuse}(v) \subseteq F_i \cap \texttt{Small}_i$. To bound $|\mathcal{C}_H|$, we separately bound $|\text{ACC\_BIG}_i|$ and $|\text{ACC\_SMALL}_i|$.

CLAIM A. $|\text{ACC\_BIG}_i| \leq |F|$.

Proof. Consider a node $v \in \text{ACC\_BIG}_i$, and arbitrarily associate with it one particular node $w_v \in F_i \cap \texttt{Big}_i \cap \texttt{Accuse}(v)$. Observe that w_v cannot be a nonparticipating node, since if $w_v \in \texttt{NP}_i$ then v should have been nonparticipating as well. (See step 2(a)iii in the algorithm.) We use this observation later on to show that w_v has many faulty voting neighbors.

Consider a node $w \in \texttt{Big}_i$. Group the neighbors of w into the following sets. Let $\Gamma^F(w)$ denote the set of 2-neighbors x of w such that $x \in F \setminus \texttt{NV}_i$ and $\mathcal{D}_x^{sav}(w) = \mathcal{M}_w$. Note that $\Gamma^F(w) \subseteq \Gamma_2(w) \cap (F \setminus \texttt{NV}_i)$. Let $\Gamma^{AB}(w)$ denote the nodes $v \in \text{ACC\_BIG}_i$ for which $w = w_v$. Note that by the definition of ACC_BIG and of w, $\Gamma^{AB}(w) \subseteq H$. Let $\gamma^Z(w) = |\Gamma^Z(w)|$ for any appropriate Z.

Since the majority vote carried by w was won by the nodes of $\Gamma^F(w)$, the voting rule used to determine $\mathcal{M}_w$ (step 2(a)), combined with the fact that $w \in \texttt{Big}_i$, imply that

$$\gamma^F(w) > \gamma(w)/2 \geq \lambda_i. \tag{1}$$

Observe that each node of $\Gamma^F(w)$ is connected to each node of $\Gamma^{AB}(w)$ in G_{RI}. This follows from the fact that by the definitions of $\Gamma^F(w)$ and $\Gamma^{AB}(w)$, each node $y \in \Gamma^F(w)$ is at distance 4 or less from each node $z \in \Gamma^{AB}(w)$ (since both y and z are 2-neighbors of w), and in addition, y's record regarding w's bit is erroneous while z's record is correct; hence $\mathcal{D}_y(w) \neq \mathcal{M}_w^* = \mathcal{D}_z(w)$, so y and z are connected by an edge in G_{RI}.

Let $c_F(w)$ denote the number of edges connecting the nodes of $\Gamma^F(w)$ to the nodes of $\Gamma^{AB}(w)$ in G_{RI}. As $c_F(w)$ is bounded (for w to be a voting node), we conclude that not too many nodes can be corrupted, since each corruption "costs" distinct λ_i edges in G_{RI}. More precisely,

$$c_F(w) \geq \gamma^F(w) \cdot \gamma^{AB}(w).$$

Combined with (1), it follows that $c_F(w) \geq \lambda_i \cdot \gamma^{AB}(w)$. Hence

$$\begin{aligned} |\textsc{Acc\_Big}_i| &= \sum_{w \in \mathtt{Big}_i} \gamma^{AB}(w) \;\leq\; \frac{1}{\lambda_i} \sum_{w \in \mathtt{Big}_i} c_F(w) \\ &\leq \frac{1}{\lambda_i} \sum_{x \in F \setminus \mathtt{NV}_i} \deg_{RI}(x) \;\leq\; \frac{1}{\lambda_i} \cdot (\lambda_i \cdot |F \setminus \mathtt{NV}_i|) \;\leq\; |F|. \end{aligned}$$

(The second inequality holds since the sets $\Gamma^{AB}(w)$ are distinct for different w's, and hence the numbers $c_F(w)$ count distinct edges. The third inequality relies on the definition of $\mathtt{NV}$.) □

Claim B. $|\textsc{Acc\_Small}_i| \;\leq\; 2\lambda_i |F_i \cap \mathtt{Small}_i|$.

Proof. By definition of $\textsc{Acc\_Small}_i$, each $v \in \textsc{Acc\_Small}_i$ has a neighbor in $F_i \cap \mathtt{Small}_i$ (namely, the node it accuses). It follows that $\textsc{Acc\_Small}_i \subseteq \Gamma(F_i \cap \mathtt{Small}_i)$. But the neighborhood of every $v \in F_i \cap \mathtt{Small}_i$ is of size $|\Gamma(v)| \leq 2\lambda_i$ by definition of $\mathtt{Small}_i$. Hence

$$|\textsc{Acc\_Small}_i| \;\leq\; |\Gamma(F_i \cap \mathtt{Small}_i)| \;\leq\; 2\lambda_i |F_i \cap \mathtt{Small}_i|. \qquad \square$$

The lemma now follows immediately from Claims A and B. □

Lemma 6.2. *In any phase i, the set F_i satisfies $|F_i \cap \mathtt{Small}_i| = O(|F| \cdot \lambda_{i+1})$, and the set $\mathcal{C}_i$ satisfies $|\mathcal{C}_i| = O(|F| \cdot \lambda_i \cdot \lambda_{i+1})$.*

Proof. The second claim follows from the first claim and the previous lemma as

$$|\mathcal{C}_i| \;\leq\; O(|F| + |F_i \cap \mathtt{Small}_i| \cdot \lambda_i) \;\leq\; O(|F| \cdot \lambda_{i+1} \cdot \lambda_i).$$

The first claim is proved by induction on i. The claim is clear for the beginning of phase $i = 1$. Now assume the claim for phase i and look at the beginning of phase $i + 1$. Nodes enter $F_{i+1} \cap \mathtt{Small}_{i+1}$ (if they are not in F_i) either from $\Gamma_2(\mathcal{C}_i)$ (as a result of performing MIS) or as a result of moving from $\mathtt{Big}_i$ to $\mathtt{Small}_{i+1}$. The first of those sources is bounded by noting that $\mathcal{C}_i \subseteq \mathtt{Small}_i$, so

$$|\Gamma_2(\mathcal{C}_i)| \;\leq\; 2\lambda_i |\mathcal{C}_i| \;\leq\; O(\lambda_i \cdot (|F| \cdot \lambda_{i+1} \cdot \lambda_i)) \;\leq\; O(|F| \cdot \lambda_{i+2}).$$

(The last inequality follows from the assumption that the sequence $\{\lambda_i\}$ is quadratically growing.)

As for the second, note that a node $v \in \mathtt{Big}_i$ that was not participating in the previous phase enters phase $i + 1$ with its initial value $\mathcal{M}_v^{sav}$. (Note that it was a nonparticipating node in all the previous phases.) Hence the number of such nodes joining $F_{i+1} \cap \mathtt{Small}_i$ is bounded by $|F|$. Also, the number of nodes in $F \cap \mathtt{Big}_i$ that join F_{i+1} is also bounded by $|F|$.

It remains to consider nodes $v \in H \cap \mathtt{Big}_i$ that have participated in the previous phase i. For such a node v, getting the wrong value (causing it to enter F_{i+1}) necessitates the influence of at least $\gamma(v)/2$ neighbors from $F \setminus \mathtt{NV}_i$, and thus it incurs a contribution of at least λ_i to $\sum_{x \in F \setminus \mathtt{NV}_i} \deg_{RI}(x)$ in G_{RI}. But the above sum is

bounded from above by $\lambda_i \cdot |F \setminus \mathtt{NV}_i|$. Thus the number of nodes in $\mathtt{Big}_i$ that can be influenced in such a way is bounded by $|F \setminus \mathtt{NV}_i| \leq |F|$. ∎

LEMMA 6.3. *Let i be such that $\lambda_i \geq |F|$. Then $\mathtt{NV}_i$ contains only nodes from F.*

Proof. A node $v \in H$ can be adjacent only to nodes of F in G_{RI}; hence its degree in this graph is at most $\deg_{RI}(v) \leq |F|$. Therefore, $\mathtt{NV}_i$ does not contain it by definition. ∎

LEMMA 6.4. *Let i be such that $\lambda_i \geq |F|$. Then*

(a) *every node w in $\mathtt{Big}_i$ is participating; and*

(b) *the voted value for $\mathcal{M}_w$ equals $\mathcal{M}^*_w$.*

Proof. Such a node w has at least $\gamma(w) \geq 2\lambda_i + 1$ neighbors (including itself), out of which at most λ_i are in F. By the previous lemma, none of its healthy neighbors from H can be in $\mathtt{NV}_i$. Thus, at least $\gamma(v) - \lambda_i > \gamma(v)/2$ do vote for a value for $\mathcal{M}_w$ which is equal to $\mathcal{M}^*_w$. The lemma follows. ∎

LEMMA 6.5. *If $|F| \leq \lambda_i$, then at the end of phase i, if the algorithm uses a deterministic MIS procedure MIS_D then $\mathcal{M}$ is a valid MIS assignment, and if it uses a randomized MIS procedure MIS_R then $\mathcal{M}$ is a valid MIS assignment with probability* 3/4.

Proof. Suppose that $|F| \leq \lambda_i$. By Lemma 6.4 all the nodes in $\mathtt{Big}_i$, as well as their neighbors, are participating nodes. Moreover the nodes in $\mathtt{Big}_i$ start then with their values from $\mathcal{M}^*$. As for the nodes in $\mathtt{Small}_i$, since they all are participating, a valid MIS is reached if Procedure REL_COMP computes a valid MIS.

By Lemma 6.1, the set $\mathcal{C}_i$ of nodes involved in MIS conflicts after step 2(a) of the algorithm satisfies $|\mathcal{C}_i| = O(|F| \cdot \lambda_i \cdot \lambda_{i+1})$. Since $\mathcal{C}_i \subseteq \mathtt{Small}_i$ and $Q \subseteq \Gamma_2(\mathcal{C}_i)$ (where Q is the set defined in step 2 of Procedure REL_COMP, namely, the expansion of the set $\mathcal{C}_i$), it follows that

$$|Q| \leq |\mathcal{C}_i| \cdot 2\lambda_i \leq |F| \cdot 2\lambda_i \cdot \lambda_i \cdot \lambda_{i+1} \leq 2\lambda_i^3 \lambda_{i+1} \leq \lambda_{i+3}.$$

Hence the time required to compute MIS on Q by MIS_D is $T_{MIS_D}(|Q|) \leq T_{MIS_D}(\lambda_{i+3})$, and hence the expected time required to compute MIS on Q by MIS_R is $T_{MIS_R}(|Q|) \leq T_{MIS_R}(\lambda_{i+3})$. Given the choice of time limit prescribed for the ith phase in step 2(e), Lemma 4.5 guarantees that a valid MIS assignment is produced with certainty in the deterministic case, and with probability 3/4 in the randomized case. ∎

We are now ready to give the main results.

LEMMA 6.6. *Given a deterministic distributed procedure MIS_D, if $T_{MIS_D}(n)$ is sublogarithmic or superlogarithmic, then for a suitable choice of the sequence $\{\lambda_i\}$, Algorithm $Mend[MIS_D]$ reaches a legal MIS assignment in time $T_{Mend[MIS_D]} = O(T_{MIS_D}(|F|^c))$ for some constant $c \geq 1$ (with $c = 1$ for the sublogarithmic case).*

Proof. Each of the two cases is handled separately. Consider first the case that $T_{MIS_D}(n)$ is superlogarithmic. Then we fix $\lambda_i = 2^{2^i}$. Note that this choice satisfies the quadratic growth requirement.

Let J be the first phase satisfying $\lambda_J \geq |F|$. By the previous lemma, the algorithm will halt on phase J. The total time consumed by phases 1 through J is bounded by

$$T_{Mend[MIS_D]} \;=\; \sum_{j=1}^{J} T_{MIS_D}(\lambda_{j+3}) \;=\; \sum_{j=1}^{J} T_{MIS_D}\left(2^{2^{j+3}}\right).$$

By the superlogarithmic assumption (SL2),

$$T_{Mend[MIS_D]} \;\leq\; T_{MIS_D}\left(\prod_{j=1}^{J} 2^{2^{j+3}}\right) \;=\; T_{MIS_D}\left(2^{\sum_{j=1}^{J} 2^{j+3}}\right) \;\leq\; T_{MIS_D}\left(2^{2^{J+4}}\right).$$

By the choice of J it follows that $|F| \geq 2^{2^{J-1}}$, and hence

$$T_{Mend[MIS_D]} \leq T_{MIS_D}\left(2^{2^{J-1}\cdot 2^5}\right) \leq T_{MIS_D}(|F|^{32}).$$

Next, consider the case that $T_{MIS_D}(n)$ is sublogarithmic. Then we fix $\lambda_i = T^{-1}_{MIS_D}(2^i)$. This choice satisfies the quadratic growth requirement as well. To show this, we rely on the fact that since T_{MIS_D} is sublogarithmic, its inverse $T \equiv T^{-1}_{MIS_D}$ is *superexponential*; i.e., it satisfies

$$T(x+y) \geq T(x)T(y). \tag{SE}$$

Therefore, $\lambda_{i+1} = f(2^{i+1}) \geq f(2^i)^2 = \lambda_i^2$.

Letting J be the first phase satisfying $\lambda_J \geq |F|$, we have that the total time consumed by the algorithm (that again halts on phase J) is bounded by

$$T_{Mend[MIS_D]} = \sum_{j=1}^{J} T_{MIS_D}(\lambda_{j+3}) = \sum_{j=1}^{J} T_{MIS_D}(T^{-1}_{MIS_D}(2^{j+3})) = \sum_{j=1}^{J} 2^{j+3} \leq 2^{J+4}.$$

By the choice of J it follows that $|F| \geq \lambda_{J-1}$, and hence by choice of λ_i and the fact that T_{MIS_D} is nondecreasing, $T_{MIS_D}(|F|) \geq T_{MIS_D}(\lambda_{J-1}) = 2^{J-1}$. Consequently,

$$T_{Mend[MIS_D]} \leq 32 \cdot 2^{J-1} \leq 32 \cdot T_{MIS_D}(|F|). \qquad \square$$

COROLLARY 6.7. *The MIS problem is fault locally $2^{O(\sqrt{\log |F|})}$-mendable.*

Proof. Use Procedure MIS_{PS} of [PS92] in the mending algorithm. By Lemmas 2.2 and 6.6, Algorithm $Mend[PS]$ mends MIS in time

$$T_{Mend[PS]} = O(T_{MIS_{PS}}(|F|^c)) = O(2^{\sqrt{\alpha \log |F|^c}}) = 2^{O(\sqrt{\log |F|})}$$

(using constant c from Lemma 6.6 and for some constant α). $\square$

COROLLARY 6.8. *Let the time complexity of the MIS problem be $T^*_D(n)$. If $T^*_D(n)$ is sublogarithmic, then MIS is tightly locally mendable. If $T^*_D(n)$ is superlogarithmic, then MIS is near-tightly locally mendable.* $\square$

LEMMA 6.9. *Given a randomized distributed procedure MIS_R, if $T_{MIS_R}(n)$ is sublogarithmic, then for a suitable choice of the sequence $\{\lambda_i\}$, Algorithm $Mend[MIS_R]$ reaches a legal MIS assignment in expected time $O(T_{MIS_R}(|F|))$.*

Proof. As in the sublogarithmic case of Lemma 6.6, we fix $\lambda_i = T^{-1}_{MIS_R}(2^i)$. Letting J be the first phase satisfying $\lambda_J \geq |F|$, we have that the time consumed by the first J phases is bounded in the same way as in that proof, yielding a bound of

$$T_1 \leq 32\tilde{c} \cdot T_{MIS_R}(|F|).$$

By Lemma 6.5, the algorithm will halt on any phase $i \geq J$ with probability $3/4$. Hence the algorithm will reach the execution of phase $J+k$ with probability $1/4^k$. The total expected time consumed by phases $J+1$ and on is therefore bounded by

$$T_2 = \sum_{k>0} \frac{1}{4^k} \cdot \tilde{c} \cdot T_{MIS_R}(\lambda_{J+k+3}) = \sum_{k>0} \frac{1}{4^k} \cdot \tilde{c} \cdot 2^{J+k+3} = \tilde{c} \cdot 2^{J+3} \sum_{k>0} \frac{1}{4^k} \cdot 2^k \leq \tilde{c} \cdot 2^{J+3}.$$

Again, by the choice of J it follows that $|F| \geq \lambda_{J-1}$, and hence by choice of λ_i and the fact that T_{MIS_R} is nondecreasing, $T_{MIS_R}(|F|) \geq T_{MIS_R}(\lambda_{J-1}) = 2^{J-1}$. Consequently,

$$T_2 \leq 16\tilde{c} \cdot 2^{J-1} \leq 16\tilde{c} \cdot T_{MIS_R}(|F|).$$

Finally, the algorithm will get to the final step of executing the Las Vegas procedure MIS_R only if all phases failed. This will happen with probability $1/4^k$ for $k = J_{max} - J$, and in that case, the algorithm will require an additional time $T_{MIS_R}(n)$; hence the contribution of this step to the expected time complexity is

$$T_3 \leq \frac{1}{4^k} \cdot T_{MIS_R}(n) \leq \frac{1}{4^k} \cdot T_{MIS_R}(\lambda_{J_{max}}) \leq \frac{1}{4^k} \cdot 2^{J_{max}} = \frac{2^J}{2^k} \leq \frac{T_{MIS_R}(|F|)}{2^{k-1}}.$$

Overall, the expected time complexity of the algorithm is bounded by

$$T_{Mend[MIS_R]} = T_1 + T_2 + T_3 = O(T_{MIS_R}(|F|))$$

as required. □

COROLLARY 6.10. *The MIS problem is randomly locally* $\log |F|$*-mendable.*

Proof. Use Procedure MIS_L of [L86] in the mending algorithm. By Lemmas 2.4 and 6.9, Algorithm $Mend[L]$ mends MIS in time $T_{Mend[L]} = O(T_{MIS_L}(|F|)) = O(\log |F|)$. □

COROLLARY 6.11. *Let the randomized expected time complexity of the MIS problem be* $T^*_R(n)$*. If* $T^*_R(n)$ *is sublogarithmic,*[2] *then MIS is randomly tightly locally mendable.* □

Space and communication complexity. The dominating factor in the size of the storage required is the data structure $\mathcal{D}_v(u) = \mathcal{M}_u$, storing, at each node v, a bit for each node u whose distance from v in the graph is 2 or less. This storage requirement decreases with the decreased density of the graph. In the worst case it implies $O(n)$ bits per node. The dominating factor in the communication is the voting according to $\mathcal{D}_v(u)$, where, in the worst case, each node may receive a vote from every other node. Every vote travels at most over two edges, leading to a total bit communication complexity of $O(n^2)$.

7. Discussion.

7.1. Distributed termination of the algorithm. Lemmas 6.6 and 6.9 discuss the time until a legal MIS assignment is reached. It may seem as if even after reaching a legal state, nodes still need to test in subsequent phases that they are not in conflict. Even if this had been the case, we would still consider it a reasonable requirement, since in many contexts one needs to address the problem of *fault detection* and not just fault *correction*. For instance, in the approach of self-stabilization, correction is said to have been achieved once a legal state is reached, yet all nodes must continue to periodically test the legality of their state.

Still, for completeness, we mention that our algorithm can be implemented in such a way that nodes that do not detect a conflict at the end of some phase and do not need to take any further steps in the algorithm, unless notified of conflicts by other nodes which are still active. Thus if the nodes which take steps at some phase i are not in any conflict, and have no neighbors in any conflict, at the end of phase i, then no node will take steps in any phase larger than i. (One should not confuse the action of taking steps, though, with the status of "nonparticipating" determined in step 2(a)iii in the algorithm; "nonparticipating" nodes nevertheless took some steps to determine that they are "nonparticipating.")

The required modification to the algorithm is described and analyzed in detail in [KP94].

[2] Recall that it is at most logarithmic.

7.2. Summary and open problems. The results presented in this paper are mainly of theoretical nature, since the solution is limited in a number of ways. In particular, it is not asynchronous, and, moreover, nodes are not allowed to start at arbitrary (and different) times. Even more desirable would be to have a self-stabilizing solution. Some initial steps in this direction have been taken in [KP96]. Finally, tight mendability is proved only under the assumption that the complexity of MIS is a function that "behaves nicely." (Nevertheless, even without this assumption it is known that MIS is at least nearly tightly mendable.) It would be interesting to remove that assumption.

Despite these limitations of our specific solution, it is our belief that the proposed general *approach* may be of considerable practical significance. Toward that goal, a number of necessary steps must be taken to remove the limitations discussed above. In addition, although demonstrated only for MIS, questions related to mendability and tight mendability can be asked for other distributed problems. In particular, it is desirable to apply our approach to real-life protocols.

Of course, one can think also of generalizations of our work to other models, defined by different characteristics; e.g., one can study tight fault locality in shared memory with, e.g., atomic registers. Related questions can be asked in the context of interactive tasks (as opposed to input–output tasks). These are tasks that run forever, guaranteeing certain properties (e.g., that each node enters a critical section often).

Finally, questions similar to the one asked in this paper can be raised in *nondistributed* contexts as well. For example, tight mending is meaningful also for sequential data structures. There, it forms a generalization of the area of dynamic data structures [Fre85, T83], in which one assumes that F (rather than 1 or n) changes occurred, and studies the complexity of updating the data structure.

Summary of main notation.

$G = (V, E)$—a graph representing the system, with node set V, $|V| = n$, and link set E.
$Mend[P]$—a mending algorithm for MIS, employing an MIS construction procedure P.
T_{Mend}—the running time of algorithm $Mend$.
$\mathcal{M} = (\mathcal{M}_{v_1}, \ldots, \mathcal{M}_{v_n})$—a given MIS vector.
$\mathcal{S}(\mathcal{M})$—the set induced by $\mathcal{M}$.
$\mathcal{D}_v$—an additional data structure maintained by node v.
$\mathcal{M}^*_v$, $\mathcal{D}^*_v$—the MIS bit and the data structure stored at v before the faults.
$\mathcal{M}_v$, $\mathcal{D}_v$—the MIS bit and the data structure stored at v after the faults.
F, H—the sets of faulty and healthy nodes.
$\mathcal{C}(\mathcal{M})$—the conflict set.
$\Gamma_i(W)$—the i-neighborhood of W.
$Co(U)$—the complement of U (w.r.t. V).
$Border_i(U)$—the i-border of U.
`Big`, `Small`—the sets of nodes v s.t. $|\Gamma_2(v)| > 2|F|$ (respectively, $|\Gamma_2(v)| \leq 2|F|$) for a given $|F|$.
$\hat{C}$—the set of conflicting nodes in `Small`.
F_i—the set of faulty nodes at the beginning of the ith phase.
G_{RI}—the record inconsistency graph.
$\deg_{RI}(v)$—v's degree in G_{RI}.
`Detected`—the set of nodes that detected a local inconsistency in their initial output.

λ_i—a quadratically growing sequence.
$\mathtt{NV}_i$—the nonvoting nodes in phase i.
$\mathtt{Vot}_i(v)$—the phase i voters in $\Gamma_2(v)$.
$\gamma(v) = |\Gamma_2(v)|$.
$\mathtt{Big}_i$, $\mathtt{Small}_i$—the sets of nodes with $\gamma(v) > 2\lambda_i$ (resp., $\gamma(v) \leq 2\lambda_i$).
$\mathcal{C}_F$—the conflicting nodes which are faulty (now or at the beginning).
$\mathcal{C}_H$—the conflicting nodes which are healthy (now and at the beginning).
$\mathtt{Accuse}(v)$—the nodes accused by v.
w_v—the faulty accused node associated with v.
Acc_Big_i—the set of nodes in $\mathcal{C}_H$ that accuse some neighbor in $F_i \cap \mathtt{Big}_i$.
Acc_Small_i—the set of nodes in $\mathcal{C}_H$ that accuse only neighbors from $F_i \cap \mathtt{Small}_i$.
$\Gamma^F(w)$—the set of 2-neighbors x of w such that $x \in F \setminus \mathtt{NV}_i$ and $\mathcal{D}_x^{sav}(w) = \mathcal{M}_w$.
$\Gamma^{AB}(w)$—the nodes $v \in \text{Acc\_Big}_i$ for which $w = w_v$.
$\gamma^Z(w) = |\Gamma^Z(w)|$ (for any appropriate Z).
$c_F(w)$—the number of edges connecting the nodes of $\Gamma^F(w)$ to the nodes of $\Gamma^{AB}(w)$ in G_{RI}.

Acknowledgments. It is a pleasure to thank Hadas Shachnai and Moti Yung for helpful discussions and Esther Jennings for her useful comments on an earlier draft. Thanks are also due to two anonymous referees, whose remarks have helped to improve the presentation of the paper.

REFERENCES

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni, *Applying static network protocols to dynamic networks*, in Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, Toronto, Canada, 1987, pp. 358–370.

[ACS94] B. Awerbuch, L. J. Cowen, and M. A. Smith, *Efficient asynchronous distributed symmetry breaking*, in Proceedings of the 26th ACM Symposium on the Theory of Computing, Montreal, Canada, 1994, pp. 214–223.

[ACK90] B. Awerbuch, I. Cidon, and S. Kutten, *Optimal maintenance of replicated information*, in Proceedings of the 31st IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 492–502.

[AKY90] Y. Afek, S. Kutten, and M. Yung, *Memory-efficient self stabilizing protocols for general networks*, in Proceedings of the Fourth Workshop on Distributed Algorithms, Springer-Verlag, Berlin, New York, 1990, pp. 15–28.

[AGLP] B. Awerbuch, A. Goldberg, M. Luby, and S. Plotkin, *Network decomposition and locality in distributed computation*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, Raleigh, NC, 1989, pp. 364–375.

[AK93] S. Aggarwal and S. Kutten, *Time-optimal self stabilizing spanning tree algorithms*, in Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science, Bombay, India, 1993, pp. 400–410.

[APV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilization by local checking and correction*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1991, pp. 268–277.

[APVD94] B. Awerbuch, B. Patt-Shamir, G. Varghese, and S. Dolev, *Self stabilization by local checking and global reset*, in Proceedings of the Eighth Workshop on Distributed Algorithms, Springer-Verlag, Berlin, New York, 1994, pp. 226–239.

[AV91] B. Awerbuch and G. Varghese, *Distributed program checking: A paradigm for building self-stabilizing distributed protocols*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1991, pp. 258–267.

[CGKK95] I. Cidon, I. Gopal, M. Kaplan, and S. Kutten, *Distributed control for fast networks*, IEEE Trans. Comm., 43 (1995), pp. 1950–1960.

[CK85] I. Chlamtac and S. Kutten, *A Spatial Reuse TDMA/FDMA for Mobile Multihop Radio Networks*, IEEE INFOCOM 85, Washington, DC, 1985.

[CM84] K. Chandy and J. Misra, *The drinking philosophers problem*, ACM TOPLAS, 6 (1984), pp. 632–646.

[CP87] I. Chlamtac, and S. Pinter, *Distributed node organization algorithm for channel access in a multihop dynamic radio network*, IEEE Trans. Comput., C-36 (1987), pp. 728–737.

[DH95] S. Dolev and T. Herman, *Superstabilizing protocols for dynamic distributed systems*, in Second Workshop on Self-Stabilizing Systems, Las Vegas, NV, 1995, pp. 3.1–3.15.

[F79] S. G. Finn, *Resynch procedures and a fail-safe network protocols*, IEEE Trans. Comm., COM-27 (1979), pp. 840–845.

[Fre85] G. N. Frederickson, *Data structures for on-line updating of minimum spanning trees, with applications*, SIAM J. Comput., 14 (1985), pp. 781–798.

[GP93] A. S. Gopal and K. J. Perry, *Unifying self stabilization and fault tolerance*, in Proceedings of the 12th ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 195–206.

[GPS87] A. V. Goldberg, S. Plotkin, and G. Shannon, *Parallel symmetry breaking in sparse graphs*, in Proceedings of the 19th ACM Symposium on Theory of Computing, New York, NY, 1987 pp. 315–324.

[KP94] S. Kutten and D. Peleg, *Tight Fault Locality*, Tech. report CS94-24, The Weizmann Institute of Science, Isreal, 1994.

[KP95] S. Kutten and D. Peleg, *Fault-local distributed mending*, in Proceedings of the 14th ACM Symposium on Principles of Distributed Computing, Ottawa, Canada, 1995, pp. 20–27.

[KP96] S. Kutten and B. Patt-Shamir, *Time-Adaptive Self Stabilization*, manuscript.

[L92] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.

[LPRS93] N. Linial, D. Peleg, Y. Rabinovich, and M. Saks, *Sphere packing and local majorities in graphs*, in Proceedings of the 2nd Israel Symposium on Theory of Computing and Systems, Natanya, Israel, 1993, IEEE Comp. Soc. Press, Los Alamitos, CA, pp. 141–149.

[L86] M. Luby, *A simple parallel algorithm for the maximal independent set problem*, SIAM J. Comput., 15 (1986), pp. 1036–1053.

[L80] N. A. Lynch, *Fast allocation of nearby resources in a distributed system*, in Proceedings of the 12th ACM Symposium on Theory of Computing, Los Angeles, CA, 1980, pp. 70–81.

[MNS95] A. Mayer, M. Naor, and L. Stockmeyer, *Local computations on static and dynamic graphs*, in Proceedings of the 3rd Israel Symposium on Theory of Computing and Systems, IEEE Computer Society, Los Alamitos, CA, 1995, pp. 268–278.

[MRR80] J. M. McQuillan, I. Richer, and E. C. Rosen, *The new routing algorithm for the ARPANET*, IEEE Trans. Comm., COM-28 (1980), pp. 711–719.

[NS93] M. Naor and L. Stockmeyer, *What can be computed locally?*, in Proceedings of the 25th ACM Symposium on the Theory of Computing, San Diego, CA, 1993, pp. 184–193.

[PS92] A. Panconesi and A. Srinivasan, *Improved distributed algorithms for coloring and network decomposition problems*, in Proceedings of the 24th ACM Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 581–592.

[P96] D. Peleg, *Local majority voting, small coalitions and controlling monopolies in graphs: A review*, in Proceedings of the Third Colloquium on Structural Information and Communication Complexity, Siena, Italy, Carleton University Press, Ottawa, Canada, 1996, pp. 170–179.

[R81] E. C. Rosen, *Vulnerability of network control protocols: An example*, Comput. Comm. Rev., July 1981.

[T83] R.E. Tarjan, *Data Structures and Network Algorithms*, CBMS-NSF Regional Conf. Ser. in Appl. Math. 44, SIAM, Philadelphia, PA, 1983.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 269–282

CONVERGENCE CRITERIA FOR GENETIC ALGORITHMS*

DAVID GREENHALGH† AND STEPHEN MARSHALL‡

Abstract. In this paper we discuss convergence properties for genetic algorithms. By looking at the effect of mutation on convergence, we show that by running the genetic algorithm for a sufficiently long time we can guarantee convergence to a global optimum with any specified level of confidence. We obtain an upper bound for the number of iterations necessary to ensure this, which improves previous results. Our upper bound decreases as the population size increases. We produce examples to show that in some cases this upper bound is asymptotically optimal for large population sizes. The final section discusses implications of these results for optimal coding of genetic algorithms.

Key words. genetic algorithms, convergence criteria, upper bounds, probability, optimal coding

AMS subject classifications. 68A10, 68A20, 60G35

PII. S009753979732565X

1. Introduction.

Genetic algorithms are robust search mechanisms based on underlying genetic biological principles. Having been established as a valid approach to problems requiring efficient and effective search, genetic algorithms are increasingly finding widespread application in business, scientific, and engineering circles (Goldberg (1989)). These algorithms are computationally simple yet powerful in their search for improvement. They work on a wide range of discrete search spaces and are very versatile, as few assumptions are needed about these spaces. However, perhaps because of this versatility, relatively little mathematical theory is available concerning the performance of these algorithms.

Genetic algorithms comprise three basic mechanisms: reproduction, crossover, and mutation. Typically the discrete space to be searched is coded as a set of binary strings of length γ. These strings are analogous to chromosomes. An initial population of n strings is chosen. For simplicity n is taken to be even. The goal of the genetic algorithm is to find the maximum of some objective function f (called the fitness function) defined on the search space.

A simple genetic algorithm is described in Goldberg (1989). Starting with the original population, a new population of n strings is selected in three stages. First, a new set of n strings (not necessarily distinct) is chosen from the original n strings by selecting each member of the population with probability proportional to its fitness. This is called reproduction. Second, the new set of strings is mated (paired) at random. For each pair crossover occurs with probability χ. If crossover occurs, then a position between 1 and $\gamma - 1$ is chosen at random, and the first half of the first string in the pair is matched with the second half of the second string, and the second half of the first string is matched with the first half of the second to give a new pair (like chromosomes). Third, each bit of each string in the new population is independently flipped (changed from zero to one or vice versa) with probability μ. This is called uniform mutation. Usually the crossover probability χ and the mutation probability

*Received by the editors August 6, 1997; accepted for publication (in revised form) May 2, 1999; published electronically May 2, 2000.
http://www.siam.org/journals/sicomp/30-1/32565.html
†Department of Statistics and Modeling Science, University of Strathclyde, Livingstone Tower, 26 Richmond St., Glasgow G1 1XH, Scotland (david@stams.strath.ac.uk).
‡Department of Electronic and Electrical Engineering, University of Strathclyde, Royal College Building, 204 George St., Glasgow G1 1XW, Scotland (s.marshall@eee.strath.ac.uk).

μ are small. A typical value of μ is less than 0.05. More complex genetic algorithms may have nonuniform crossover rates between iterations, or nonuniform mutation rates between iterations, or both. Other possibilities include more complex crossover schemes (for example, multiple point crossover) and mutation schemes (for example, mutation of blocks of digits altogether simultaneously).

The process of reproduction probabilistically ensures that individuals of high fitness are preferentially maintained in the population, while crossover and mutation are necessary to introduce the diversity needed to ensure that the entire sample space is reachable and avoid becoming stuck at suboptimal solutions. In fact mutation on its own is sufficient to introduce such diversity. The idea is that after a large number of iterations the population should consist mainly of individuals very close to the optimal solution. When the genetic algorithm has finished, the fitness of each member of the population is evaluated and the member corresponding to the maximum fitness taken as the optimum member of the search space.

However, with this method it is possible for the optimal solution to be seen and then disappear, if it appears in some population and then is lost due to mutation or crossover. A more effective method, which requires very little extra effort and is commonly used, is to save at each stage the population member of maximal fitness which has been seen so far (in either the initial population or one of the end of iteration populations up to now). This is updated each time that the fitnesses for a population are calculated. This is sometimes called a queen bee or elitist approach.

Heuristically genetic algorithms appear in practice to work well, but there is little mathematical analysis underpinning their performance. One key problem area is how to decide when to stop. A simplistic approach (but one that is often taken) is to stop after a (large) arbitrary fixed number of iterations. A slightly more sophisticated approach is described by de Jong (1975), who defines the on-line performance (describing on-going performance) as the average of all fitness functions evaluated up to and including the current trial. The off-line performance (describing convergence) is a running average of the best performance values to a particular time. A more sophisticated alternative to stopping after a fixed number of iterations is to stop after either the on-line or, preferably, the off-line performance appears to have stabilized.

As genetic algorithms are stochastic search algorithms it is never possible to guarantee that an optimal solution (in general there may be more than one) has been seen after a fixed number of iterations. There is always a probability that we have not seen an optimal solution. However, we can examine a sort of convergence in probability. Given a fixed probability δ $(0 < \delta < 1)$ we may ask what is the smallest number of iterations required to guarantee that we have seen an optimal solution with probability δ. We denote this number by $t(\delta)$.

2. Genetic algorithms as Markov chains. Previous work has succeeded in deriving bounds for this quantity $t(\delta)$ by modeling genetic algorithms as Markov chains. Vose and Liepins (1991) show how genetic algorithms can be expected to display punctuated equilibria, often appearing to converge to false suboptimal solutions for relatively long times before moving on to better solutions. This behavior is often seen in practice. Thus convergence criteria such as de Jong's on-line and off-line performance can often be misleading. Nix and Vose (1992) characterize genetic algorithms as Markov chains. They show that if the state variable is taken to be the current population, then the Markov chain is ergodic if and only if the mutation probability μ is strictly positive. Hence a steady state behavior exists and it is not possible for the Markov chain to be trapped at a suboptimal solution indefinitely.

Aytug and Koehler (1996) use this Markov chain formulation to find bounds for $t(\delta)$. For $x \geq 0$ define INT$[x]$ to be the smallest integer greater than or equal to x. The show that $t_2(\delta) \leq t(\delta) \leq t_1(\delta)$ where

$$t_1(\delta) = \text{INT}\left[\frac{\ln(1-\delta)}{\ln[1-\min\{(1-\mu)^{\gamma n}, \mu^{\gamma n}\}]}\right]$$

and

$$t_2(\delta) = \text{INT}\left[\max_j \left\{\frac{\ln(1-\delta)}{\ln[\rho(Q - Qe_j e_j^T)]}\right\}\right].$$

In the definition of $t_2(\delta)$ the maximum is taken over all states of the Markov chain, Q is the Markov chain transition matrix, and e_j is the jth unit vector. $\rho(A)$ denotes the spectral radius of a matrix A. The lower bound is difficult to evaluate, but the upper bound gives an easily computed sufficient condition that after $t_1(\delta)$ iterations we have seen the optimal solution with probability at least δ.

Although simple genetic algorithms use binary coding it is sometimes advantageous to use a nonbinary coding. This time each element of the space to be searched is represented by a vector in $\{0, 1, 2, \ldots, K-1\}^\gamma$, called a K-string of length γ. Reproduction and crossover proceed as before. For mutation, each digit of each population member independently mutates with probability μ. When a digit mutates it changes to one of the other $K-1$ digits, chosen at random. The case $K = 2$ is the binary case. Goldberg (1989) has argued for binary over higher cardinality representations for maximizing implicit parallelism in genetic processing. Notwithstanding this argument, practitioners report better performance with nonbinary representations in many applications (Davis, 1991). Aytug, Bhattacharrya, and Koehler (1996) extend their results from binary representations to alphabets of cardinality $K = 2^x$ (i.e., alphabets whose cardinality is a power of 2) to show that in a cardinality K genetic algorithm the number of iterations necessary to see all populations with probability δ is bounded above by

$$t_1(\delta) = \text{INT}\left[\frac{\ln(1-\delta)}{\ln\left[1-\min\left\{(1-\mu)^{\gamma n}, \left(\frac{\mu}{K-1}\right)^{\gamma n}\right\}\right]}\right].$$

For $K = 2$ this upper bound reduces to the one obtained by Aytug and Koehler (1996). Note that Aytug, Bhattacharrya, and Koehler (1996) incorrectly define INT$[x]$ as the smallest integer greater than (as opposed to greater than or equal to) x. The correct definition is given in Aytug and Koehler (1996).

The bounds of Aytug, Bhattacharrya, and Koehler assume that with specified probability they have seen every population and hence every optimal solution. This is a huge overkill. Aytug and Koehler believe tighter bounds could be derived by establishing a way where all population members (not all populations) are seen at least once with a specified probability. In fact a very simple argument along the suggested lines enables us to improve Aytug, Bhattacharrya, and Koehler's upper bound.

3. Main results.

THEOREM 1.

$$t(\delta) \leq \tilde{t}_1(\delta) = \text{INT}\left[\frac{\ln(1-\delta)}{n \ln\left[1 - \min\left\{(1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right), \left(\frac{\mu}{K-1}\right)^{\gamma}\right\}\right]}\right].$$

Proof. See the appendix.

Note that the only properties of the crossover function used in the proof of Theorem 1 are in (b), where we show that if after crossover we have exactly one copy of the optimal string $\underline{e}^*$ and one nonoptimal string, they agree in at most $\gamma - 2$ digits. The arguments used to show this do not depend on the crossover scheme having a one-point crossover or a uniform crossover rate. Thus our results are equally valid if our crossover scheme is a multiple point crossover or has a nonuniform crossover rate or both. The arguments also hold for any selection process which at each stage selects n strings (possibly including some replicated ones) from an original population of n strings. However, although our results hold for a wide variety of genetic algorithm models, they do not hold for the general case given by Vose, where mutation masks and crossover masks are employed instead of rates. The mixing probability formula was first given in Vose and Wright (1995) and generalized to the general cardinality case in Koehler, Bhattacharrya, and Vose (1997).

It is straightforward to show this bound is always an improvement on $t_1(\delta)$.

LEMMA 1.

$$\tilde{t}_1(\delta) \leq t_1(\delta).$$

Proof. If $\mu \leq \frac{K-1}{K}$, $1-\mu \geq \frac{\mu}{K-1}$. Then for $n = 1$, $t_1(\delta) = \tilde{t}_1(\delta)$, whence the result follows as $\tilde{t}_1(\delta)$ is decreasing in n and $t_1(\delta)$ increasing in n. If $\mu > \frac{K-1}{K}$, $1-\mu < \frac{\mu}{K-1}$ and it is straightforward to show that for $n = 1$, $t_1(\delta) > \tilde{t}_1(\delta)$. The result follows similarly to above.

Thus our bound $\tilde{t}_1(\delta)$ is significantly tighter than the previous one. In addition it decreases as the population size n increases, whereas $t_1(\delta)$ increases with n. This is intuitively correct as increasing the population size means that there is a greater probability of the optimum being present in any given population. On the other hand the total number of possible combinations of populations will be larger, and it will therefore take significantly more iterations to encounter all of them. Moreover our results hold for alphabets of arbitrary cardinality, whereas the results of Aytug, Bhattacharrya, and Koehler require that the cardinality K be a power of 2.

In practice the mutation rate μ is usually small so that μ^γ is often very small. In this case, using the approximation $\ln(1+x) \approx x$ for x small and $\text{INT}[x] \approx x$ for x large, then the original bound may be approximated as

$$t_1(\delta) \approx -\frac{(K-1)^{\gamma n} \ln(1-\delta)}{\mu^{\gamma n}}$$

if $t_1(\delta)$ is large and the new bound as

$$\tilde{t}_1(\delta) \approx -\frac{(K-1)^{\gamma} \ln(1-\delta)}{n\mu^{\gamma}} \tag{1}$$

if $\tilde{t}_1(\delta)$ is large. The ratio is therefore

$$\frac{t_1(\delta)}{\tilde{t}_1(\delta)} \approx n \left(\frac{\mu}{K-1} \right)^{-\gamma(n-1)}.$$

As a numerical example we take $K = 2$, $\gamma = 10$, $\mu = 0.2$, $n = 15$, and $\delta = 0.9$ giving $t_1(\delta)$ as approximately 1.6×10^{105} iterations, whereas for the same figures $\tilde{t}_1(\delta)$ is very much smaller at 1.5×10^6 iterations. Indeed if we double the population size to $n = 30$, the new bound reduces by half to 7.5×10^5 and the old bound almost squares to 1.1×10^{210}.

Aytug, Bhattacharrya, and Koehler (1996) show that the bound $t_1(\delta)$ is minimized at $n = 1$ and $\mu = \frac{K-1}{K}$. It is straightforward to show that for fixed n our new upper bound $\tilde{t}_1(\delta)$ is also minimized at $\mu = \frac{K-1}{K}$ for $\gamma \geq 2$ and at $\mu = 1$ for $\gamma = 1$. This says that as the bound is based only on mutation, it gives the best bound when the process is diffusing most rapidly, which happens when $\mu = \frac{K-1}{K}$ for $\gamma \geq 2$ and when $\mu = 1$ for $\gamma = 1$. This is also true for the upper bound of Aytug, Bhattacharrya, and Koehler for the most practically relevant situation where $\gamma \geq 2$. However, if $\gamma = 1$, then Aytug and Koehler's upper bound is still maximized at $\mu = \frac{K-1}{K}$. It is similarly straightforward to show that $\tilde{t}_1(\delta)$ is monotone increasing in n for μ fixed. This says that as the population size increases, our search naturally becomes more efficient at finding the optimal solution. At $n = 1$, having seen every population is equivalent to having seen that every member of the search space and the two bounds are the same.

Theorem 1 assumes a uniform mutation rate, where at each stage each digit mutates independently. These results may carry over to different mutation schemes by reworking the calculations and redoing some of the proofs. For example, suppose that at iteration i each digit of each population member mutates independently at random with probability μ_i. If μ_i is bounded above away from one and below away from zero, so that $\mu_1 > \mu_i > \mu_2$ for some $\mu_1 < 1$ and $\mu_2 > 0$, then it is straightforward to modify the proof of Theorem 1 to show that

$$t(\delta) \leq \text{INT} \left[\frac{\ln(1-\delta)}{n \ln \left[1 - \min \left\{ (1-\mu_1)^{\gamma-1} \left(\frac{\mu_1}{K-1} \right), \left(\frac{\mu_2}{K-1} \right)^{\gamma} \right\} \right]} \right].$$

We can obtain a better bound (again by modifying the argument of Theorem 1) to show that $t(\delta) \leq T_1$, where T_1 is the smallest integer such that

$$\sum_{i=1}^{T_1} \ln \left[1 - \min \left\{ (1-\mu_i)^{\gamma-1} \left(\frac{\mu_i}{K_1} \right), \left(\frac{\mu_i}{K-1} \right)^{\gamma} \right\} \right] \leq \frac{\ln(1-\delta)}{n}.$$

In most practical situations each μ_i will be small. In such circumstances define $\tilde{T}_1$ to be the smallest integer T such that

$$\sum_{i=1}^{T} \mu_i^{\gamma} \geq -\frac{(K-1)^{\gamma} \ln(1-\delta)}{n}.$$

Then, using similar approximations to the ones made in deriving (1), we deduce that T_1 can be approximated by $\tilde{T}_1$ if each μ_i is sufficiently small.

Theorem 1 gives an upper bound $\tilde{t}_1(\delta)$ for $t(\delta)$, the number of iterations necessary to ensure that we have seen the optimal solution with probability δ. This is an improvement on previous upper bounds. A natural question is whether this bound can be reduced further. Of course for specific objective functions the genetic algorithm will usually perform much better than our upper bound. However, the following examples exhibit circumstances where no further improvement is possible, and our upper bound is asymptotically optimal (for large n). Thus with a completely general objective function and population size it is not possible to improve on our result. The first example shows that no better bound can be obtained for the range $\mu \leq \frac{K-1}{K}$, the second that none can be obtained for the range $\mu \geq \frac{K-1}{K}$.

Example 1. Consider an objective function f defined on all K-strings of length γ to be ε everywhere apart from the origin $\underline{0} = (0, 0, 0, \ldots, 0)$ and the vector $\underline{e} = (K-1, K-1, K-1, \ldots, K-1)$, $0 < \varepsilon < 1$, $f(\underline{0}) = 1$, and $f(\underline{e}) = 2$. There is a nonzero uniform mutation rate μ. For the sake of definiteness we assume a nonzero uniform crossover rate with a single point crossover, although our argument would work equally well with a nonuniform crossover rate or multiple point crossover or both. At the selection phase individuals are selected in proportion to their fitness. We start with an initial population of n $\underline{0}$'s.

RESULT 1. *Consider Example* 1 *for* $\mu \leq \frac{K-1}{K}$. *Then*

(2)
$$\tilde{t}_1(\delta) \geq t(\delta) > \text{INT}\left[\frac{\ln(1-\delta)}{n\ln\left[1-\left(\frac{\mu}{K-1}\right)^{\gamma}\right] + \ln(1-\theta^n) - n\ln(1+(n-1)\varepsilon)}\right] - 1,$$

where

$$\theta = 1 - \frac{(1-\mu)^{\gamma}}{1-\left(\frac{\mu}{K-1}\right)^{\gamma}} \qquad (0 < \theta < 1).$$

The proof of Result 1 is given in the appendix. Result 1 shows that the bound of Theorem 1 is optimal in the following sense. Pick $\varepsilon_n = \frac{1}{n^3}$ (so ε depends on n). Then as $\ln(1+x) \leq x$ for $x \geq 0$, $0 \leq n\ln(1+(n-1)\varepsilon_n) \leq \frac{n(n-1)}{n^3} \leq \frac{1}{n}$ for all n. As $n \to \infty$, $\tilde{t}_1(\delta)$ tends to zero as $\frac{1}{n}$, and θ^n tends to zero exponentially fast. Hence it is straightforward to show that for $\mu \leq \frac{K-1}{K}$ and n large enough

$$\left|\frac{\ln(1-\delta)}{n\ln\left[1-\left(\frac{\mu}{K-1}\right)^{\gamma}\right]} - \frac{\ln(1-\delta)}{n\ln\left[1-\left(\frac{\mu}{K-1}\right)^{\gamma}\right] + \ln(1-\theta^n) - n\ln(1+(n-1)\varepsilon_n)}\right| \leq \tilde{t}_1(\delta)\frac{-2}{n^2\ln\left[1-\left(\frac{\mu}{K-1}\right)^{\gamma}\right]},$$

which tends to zero at a much faster rate than $\tilde{t}_1(\delta)$ does. A consequence of this result is that provided that n is very large then the upper bound in (2) is one plus the lower bound for almost all values of δ; so in this case, $\tilde{t}_1(\delta)$ gives an arbitrarily close approximation to $t(\delta)$.

The idea behind Example 1 is that we have a population initially consisting of all $\underline{0}$'s. At each iteration the optimum solution may appear in the population by direct mutation for some $\underline{0}$ in one step. Otherwise, provided that at least one $\underline{0}$ remains at the end of this iteration and ε is very small, after reproduction we are almost certainly back to the initial population of n $\underline{0}$'s (ignoring small terms in ε). Thus the only way the optimum can appear in the population is by direct mutation of one of the population members. This argument breaks down if, in any iteration, all of the population members simultaneously mutate away from $\underline{0}$ in one iteration, but none of them mutates to the optimum. However, by making n large enough, we can make the probability of this happening arbitrarily small, and provided that we choose $\varepsilon_n = \frac{1}{n^3}$, the small terms in ε can still be neglected. Thus if n is large enough, $\tilde{t}_1(\delta)$ can be made as close as we like to $t(\delta)$.

Our second example shows a similar result when $\mu \geq \frac{K-1}{K}$.

Example 2. Consider an objective function f defined on all K-strings on length γ to be ε everywhere apart from $\underline{e}_0 = (0, K-1, K-1, \ldots, K-1)$ and $\underline{e} = (K-1, K-1, K-1, \ldots, K-1)$, $0 < \varepsilon < 1$, $f(\underline{e}_0) = 1$, and $f(\underline{e}) = 2$. There is a nonzero uniform mutation rate μ. Again for the sake of definiteness we assume a nonzero uniform crossover rate with a single crossover, although our argument would work equally well with a nonuniform crossover rate or multiple crossover or both. The selection phase and the initial population are the same as in Example 1.

RESULT 2. *Consider Example* 2 *for* $\mu \geq \frac{K-1}{K}$. *Then if* n *is large enough*

(3)

$$\tilde{t}_1(\delta) \geq t(\delta) > \text{INT}\left[\frac{\ln(1-\delta)}{n\ln\left[1-(1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right)\right] + \ln(1-\theta^n) - n\ln(1+(n-1)\varepsilon)}\right] - 1,$$

where

$$\theta = 1 - \frac{(1-\mu)^{\gamma}}{1-(1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right)} \qquad (0 < \theta < 1).$$

The proof of Result 2 is similar to that of Result 1. Similarly to Result 1 this can be used to show that if $\varepsilon_n = \frac{1}{n^3}$, then for $\mu > \frac{K-1}{K}$, as n becomes very large $\tilde{t}_1(\delta)$ again gives a very close approximation to $t(\delta)$ for almost all values of δ and so is asymptotically optimal for large n.

4. Implications for coding of genetic algorithms. Aytug, Bhattacharrya, and Koehler (1996) discuss the implications of their upper bound for the practical problem of coding genetic algorithms. They ask: given a problem whose domain can be represented with a coding of cardinality K, what is the best choice of coding? In other words, should one use binary or a higher cardinality coding for a given problem? The answer to this question is given by the following theorem (Aytug, Bhattacharrya, and Koehler, 1996).

THEOREM 2. *If $t_1(\delta)$ iterations guarantee that an optimal solution has been found with probability δ by using binary coding, to guarantee the same worst-case level of confidence in $t_1(\delta)$ or fewer iterations by using $K = 2^x$ cardinality coding, the mutation rate of the higher cardinality genetic algorithm (μ_K) has to satisfy*

$$\mu_K \geq \mu_2^x(2^x - 1),$$

where $\mu_2 \leq 0.5$ is the mutation rate used in binary coding.

This theorem shows that when an optimal mutation rate $\mu_2 = 0.5$ is used for a binary genetic algorithm, if the same problem is re-encoded using a higher cardinality alphabet (and thus a correspondingly shorter string length) in the worst case, the re-encoded genetic algorithm cannot perform better than the binary genetic algorithm and can just match the worst case performance of the binary genetic algorithm by operating with mutation rate $\mu_K = \frac{K-1}{K}$. However, Aytug, Bhattacharrya, and Koehler show that for a lower binary genetic algorithm mutation rate we can achieve a better upper bound, and hence a better worst-case performance with the re-encoded higher cardinality genetic algorithm, even if the mutation rate for the re-encoded genetic algorithm is lower than that of the binary genetic algorithm. In other words, the re-encoded genetic algorithm not only can just match the performance of the binary genetic algorithm at the optimal binary mutation rate but can also outperform it even with a mutation rate lower than that of the binary genetic algorithm if the binary genetic algorithm mutation rate is nonoptimal. Because the functional forms of the worst-case upper bounds obtained by Aytug, Bhattacharrya, and Koehler (1996) are very similar to our own, provided that $\mu_2 \leq 0.5$ and $\gamma \geq 2$, it is straightforward to show that the same results hold using the new improved upper bounds derived in this paper (provided $\gamma \geq 2$, as will be the case for all practical applications). In fact it is also straightforward to show that similar results hold when $\gamma = 1$ with two minor modifications. First, for $\gamma = 1$ using our new bounds, Theorem 2 holds without any restriction on the value of μ_2. Second, if the optimal mutation rate $\mu_2 = 1$ is used for the binary genetic algorithm, in the worst case the re-encoded genetic algorithm always performs worse than the binary genetic algorithm, even when operating with the optimal re-encoded genetic algorithm mutation rate $\mu_K = 1$.

5. Summary and conclusions. Genetic algorithms are a robust search method for finding global optima of functions which mimic the biological principles of sexual reproduction. They have increasing practical applications in many areas (Goldberg (1989), Harvey and Marshall (1996)). Although they appear to perform well in practice, there is not a great deal of mathematical work underpinning their performance. One reason for this is they make few assumptions about the underlying state space. While this gives them great flexibility it makes general results hard to obtain. The NFL (no free lunch) work of Wolpert and Macready (Wolpert and Macready (1995), Wolpert (1996), Salomon (1997)) is a framework that addresses the core aspects of search, focusing on the connection between fitness functions and effective search algorithms. The key result is the No Free Lunch Theorem, which states that averaged over all problems, all search algorithms perform equally. This result implies that if we are comparing a genetic algorithm to some other algorithm (e.g., simulated annealing or even random search) and the genetic algorithm performs better on some class of problems, then the other algorithm necessarily performs better on problems outside the class. Thus it is essential to incorporate knowledge of the problem into the search algorithm.

In this paper we have examined questions of convergence. We ask how long must

a genetic algorithm run for it to be reasonably certain that we have seen the optimal solution. A naive approach is just to run the genetic algorithm for a large, fixed number of iterations, but there is no guarantee that this will be successful. A better approach is to look at the on-line and off-line performances, but as the work of Vose and Liepins (1991) shows that genetic algorithms can often appear to converge to false optima, this method may also give a misleading result.

Previous work of Aytug, Bhattacharrya, and Koehler has examined this question using the theory of Markov chains. Taking the state variable to be the current genetic algorithm population they show that the Markov chain is ergodic. They also show that given any probability $\delta < 1$, by running the genetic algorithm for sufficiently long, it is possible to guarantee that we have seen the optimal solution with probability δ and obtain an upper bound for the time necessary to do this. We have examined this problem using the simpler approach of looking at the effect of mutation on the convergence of the genetic algorithm. We also showed that by running the genetic algorithm for long enough we can guarantee convergence to the global optimum with any prespecified level of confidence and obtained a much lower bound for the number of iterations necessary to do this. Moreover our bound has the desirable property that it decreases as the population size increases (and the genetic algorithm gets more effective), whereas the bounds of Aytug, Bhattacharrya, and Koehler increase with increasing population size. The upper bounds we obtain are much smaller than those of Aytug, Bhattacharrya, and Koehler for realistic parameter values. Nevertheless they are still large, and for most practical problems the effect of reproduction and crossover, combined with some smoothness of the objective function, will mean that in practice the genetic algorithm converges much faster, although as discussed above it is difficult to say how much faster. The question therefore arises of whether it is possible to improve on these results further. We have produced examples to show that in general it is not possible to improve further on our upper bound, at least if the population size is sufficiently large. In the final section we showed that the results of Aytug, Bhattacharrya, and Koehler on optimal coding of genetic algorithms carry over with our new improved upper bounds.

Note that the proof of Theorem 1 assumed that the objective or fitness function had a unique global maximum attained for exactly one string. In general we will not know whether this is true. If there are several distinct strings where the objective function attains its maximum value, then the proof of Theorem 1 is still valid, but in this case we expect the genetic algorithm to converge faster than our bound. If we know in advance the number of strings where the fitness function attains its maximum value and this number of strings is greater than one, then it is possible to improve further on our upper bound (with this restricted class of objective functions). This is a possible area for future research. In general we will not know this information and, as we have shown in our paper, in the general case the bound which we have found will be asymptotically optimal.

Note added in proof. During the review of this manuscript we were informed by a referee that Aytug and Koehler submitted a paper in December 1996 to the *INFORMs Journal on Computing* (Aytug and Koehler (1999)) which gives similar results to this paper for the binary case and the more general results are contained in a working paper by Aytug, Koehler, and Bhattacharrya. We have seen Aytug and Koehler's paper but have not yet seen this working paper. The results are similar, but the proofs in our manuscript were simpler. These results were derived independently, to the best of our knowledge; as yet, neither of these papers has been published.

Appendix. *Proof of Theorem* 1. We assume that the objective function has a unique global optimum value corresponding to a unique string $\underline{e}^*$ and work out the probability that we have seen this value $\underline{e}^*$ by the end of the tth iteration. There is no loss of generality in this assumption. If the objective function attains its global optimum value at several distinct strings, then we arbitrarily choose one as $\underline{e}^*$ and our proof will still be valid (although in this case we would expect the genetic algorithm to actually converge faster).

(i) $1 - \mu \geq \frac{\mu}{K-1}$.

Given a member of the search space which agrees with the optimal string $\underline{e}^*$ in y digits, the probability that it mutates to this optimal string under mutation alone in one step is

$$(1-\mu)^y \left(\frac{\mu}{K-1}\right)^{\gamma-y} \geq \left(\frac{\mu}{K-1}\right)^{\gamma}.$$

Hence the probability that it does not mutate to $\underline{e}^*$ under mutation alone in one step is at most

$$\left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right).$$

Although this bound involves only the mutation operator it holds whatever happens in the reproduction and crossover phases. To see this, note that reproduction and crossover happen before mutation. Thus whatever happens in the reproduction and crossover phases immediately before mutation, we end up with a population of n strings and the probability that any specified one does not mutate to $\underline{e}^*$ in the mutation stage is at most

$$\left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right).$$

In t iterations of the genetic algorithm there are nt independent mutations. Hence if P_t is the probability that the optimum has been seen after t iterations,

$$P_t \geq 1 - \left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right)^{nt}$$

(whatever happens in the reproduction and crossover phases).

Choosing t_1 such that

$$1 - \left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right)^{nt_1} \geq \delta \tag{A.1}$$

guarantees that the optimum has been seen after t_1 steps with probability at least δ (therefore $t_1 \geq t(\delta)$). (A.1) can be rewritten

$$t_1 \geq \frac{\ln(1-\delta)}{n \ln\left[1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right]}, \tag{A.2}$$

and therefore (A.2) implies $t_1 \geq t(\delta)$. Thus

$$t(\delta) \leq \tilde{t}_1(\delta) = \mathrm{INT}\left[\frac{\ln(1-\delta)}{n \ln\left[1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right]}\right],$$

and the result of Theorem 1 follows in this case.

(ii) $\frac{\mu}{K-1} \geq 1-\mu$.

In t iterations of the genetic algorithm the mutation operator is applied in total nt times to the population members occurring after reproduction and crossover. If the optimal solution has not yet been seen in either the initial population or the populations at the end of each iteration, consider the population members immediately before mutation; that is, after reproduction and crossover. These can be divided naturally into pairs according to which crossover they come from. For each pair either of the following cases occurs.

(a) Neither string in the pair is $\underline{e}^*$. Suppose that the strings in the pair agree with $\underline{e}^*$ in y_1 and y_2 digits, respectively, where $0 \leq y_1 \leq \gamma - 1$ and $0 \leq y_2 \leq \gamma - 1$. For $i = 1,\ 2$ the probability that the ith member of the pair mutates to $\underline{e}^*$ in one step is

$$(1-\mu)^{y_i}\left(\frac{\mu}{K-1}\right)^{\gamma-y_i} \geq (1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right).$$

Hence the probability that neither string in the pair mutates to the optimal string is at most

$$\left(1-(1-\mu)^{\gamma-1}\frac{\mu}{K-1}\right)^2.$$

(b) Exactly one member of the pair is $\underline{e}^*$. The pair must have been obtained by crossover from two distinct population members, neither of which was $\underline{e}^*$. Suppose that the pair before crossover agreed with $\underline{e}^*$ in z_1 and z_2 digits, respectively, where $0 \leq z_1 \leq \gamma - 1$ and $0 \leq z_2 \leq \gamma - 1$. In this case $\gamma \geq 2$ and the other member of the pair after crossover must agree with $\underline{e}^*$ in $z_1 + z_2 - \gamma \leq \gamma - 2$ digits. Hence the probability that neither member of the pair after crossover mutates to $\underline{e}^*$ is

$$\begin{aligned}&(1-(1-\mu)^{\gamma})\left(1-(1-\mu)^{z_1+z_2-\gamma}\left(\frac{\mu}{K-1}\right)^{2\gamma-z_1-z_2}\right)\\ &\quad\leq (1-(1-\mu)^{\gamma})\left(1-(1-\mu)^{\gamma-2}\left(\frac{\mu}{K-1}\right)^2\right)\end{aligned}$$

as

$$\begin{aligned}(1-\mu)^{\gamma-2}\left(\frac{\mu}{K-1}\right)^2 &\leq (1-\mu)^{y_1+y_2-\gamma}\left(\frac{\mu}{K-1}\right)^{2y-\gamma_1-\gamma_2}\\ &\leq \left(1-(1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right)\right)^2,\end{aligned}$$

using the inequality

$$2(1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right) \leq (1-\mu)^{\gamma} + (1-\mu)^{\gamma-2}\left(\frac{\mu}{K-1}\right)^2.$$

As the pair of strings immediately before mutation is obtained from reproduction followed by crossover, it is not possible for both strings in this pair to be $\underline{e}^*$. This is because we have supposed that $\underline{e}^*$ has not yet appeared in the initial population or any of the end of iteration populations.

Combining the results of (a) and (b) we deduce that $1 - P_t$, the probability the optimal string has not been seen in the end of iteration populations by t iterations, is less than

$$\left(1 - (1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right)\right)^{nt}.$$

Arguing as in the case where $1 - \mu > \frac{\mu}{K-1}$ we deduce that

$$t(\delta) \leq \tilde{t}_1(\delta) = \text{INT}\left[\frac{\ln(1-\delta)}{n\ln\left[1 - (1-\mu)^{\gamma-1}\left(\frac{\mu}{K-1}\right)\right]}\right].$$

This completes the proof of Theorem 1.

Proof of Result 1. Consider what happens at the first iteration of the genetic algorithm. Reproduction and crossover have no effect on the initial population. After mutation either of the following cases occur.

(i) The optimal string has been seen with probability

$$1 - \left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right)^{n}.$$

(ii) The optimal string has not been seen and no $\underline{0}$'s remain. This is equivalent to saying that all n population members neither mutate to the optimal string nor stay unchanged and occurs with probability

$$\left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma} - (1-\mu)^{\gamma}\right)^{n}.$$

(iii) The optimal string has not been seen and at least one $\underline{0}$ remains. This occurs with probability

$$\left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right)^{n} - \left(1 - \left(\frac{\mu}{K-1}\right)^{\gamma} - (1-\mu)^{\gamma}\right)^{n}.$$

Provided that at least one $\underline{0}$ remains and no $\underline{e}$'s are present at the end of iteration k, then in the next iteration of the genetic algorithm after reproduction the population again consists entirely of $\underline{0}$'s with probability at least $(\frac{1}{1+(n-1)\varepsilon})^n$. Hence as

Prob(Optimal string not seen after t iterations)

$\geq$ Prob(Optimal string not seen, and at least one $\underline{0}$ remains in all t iterations),

$$\begin{aligned}
&1 - P_t \\
&\geq \left[\left(\left\{1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right\}^{n} - \left\{1 - \left(\frac{\mu}{K-1}\right)^{\gamma} - (1-\mu)^{\gamma}\right\}^{n}\right)\right]^{t}\left(\frac{1}{1+(n-1)\varepsilon}\right)^{n(t-1)} \\
&\geq \left[\left(\left\{1 - \left(\frac{\mu}{K-1}\right)^{\gamma}\right\}^{n} - \left\{1 - \left(\frac{\mu}{K-1}\right)^{\gamma} - (1-\mu)^{\gamma}\right\}^{n}\right)\left(\frac{1}{1+(n-1)\varepsilon}\right)^{n}\right]^{t}.
\end{aligned}$$

Choosing t_1 so that

(A.3)

$$\delta > 1 - \left[\left(\left\{ 1 - \left(\frac{\mu}{K-1} \right)^{\gamma} \right\}^n - \left\{ 1 - \left(\frac{\mu}{K-1} \right)^{\gamma} - (1-\mu)^{\gamma} \right\}^n \right) \left(\frac{1}{1+(n-1)\varepsilon} \right)^n \right]^{t_1}$$

guarantees that the optimum has been seen after t_1 steps with probability strictly less than δ (so $t(\delta) > t_1$). (A.3) can be rewritten

(A.4)

$$t_1 < \frac{\ln(1-\delta)}{\ln\left[\left\{ 1 - \left(\frac{\mu}{K-1} \right)^{\gamma} \right\}^n - \left\{ 1 - \left(\frac{\mu}{K-1} \right)^{\gamma} - (1-\mu)^{\gamma} \right\}^n \right] - n \ln(1+(n-1)\varepsilon)}$$

and therefore (A.4) implies $t_1 < t(\delta)$. Thus $t(\delta)$ is greater than

$$\mathrm{INT}\left[\frac{\ln(1-\delta)}{\ln\left[\left\{ 1 - \left(\frac{\mu}{K-1} \right)^{\gamma} \right\}^n - \left\{ 1 - \left(\frac{\mu}{K-1} \right)^{\gamma} - (1-\mu)^{\gamma} \right\}^n \right] - n \ln(1+(n-1)\varepsilon)} \right] - 1,$$

$$= \mathrm{INT}\left[\frac{\ln(1-\delta)}{n \ln\left[1 - \left(\frac{\mu}{K-1} \right)^{\gamma} \right] + \ln(1-\theta^n) - n \ln(1+(n-1)\varepsilon)} \right] - 1,$$

where

$$\theta = 1 - \frac{(1-\mu)^{\gamma}}{1 - \left(\frac{\mu}{K-1} \right)^{\gamma}} \qquad (0 < \theta < 1).$$

REFERENCES

H. Aytug, S. Bhattacharrya, and G. J. Koehler (1996), *A Markov Chain analysis of genetic algorithms with power of* 2 *cardinality alphabets*, European J. Oper. Res., 96, pp. 195–201.

H. Aytug and G. J. Koehler (1996), *Stopping criteria for finite length genetic algorithms*, ORSA J. Comp., 8, pp. 183–191.

H. Aytug and G. J. Koehler (1999), *A new stopping criterion for genetic algorithms*, INFORMs J. Comput., submitted; also available online from http://www.cba.ufl.edu/dis/research.html.

L. Davis (1991), *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York.

K. A. de Jong (1975), *An Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, Ph.D. thesis, University of Michigan, Ann Arbor, MI.

D. E. Goldberg (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison–Wesley, Reading, MA.

N. R. Harvey and S. Marshall (1996), *The Use of Genetic Algorithms in Morphological Filter Design*, Signal Processing: Image Communication, 8, pp. 55–71.

G. J. Koehler, S. Bhattacharrya, and M. D. Vose (1997), *General cardinality genetic algorithms*, Evolutionary Computation, 5, pp. 439–459.

A. Nix and M. D. Vose (1992), *Modelling genetic algorithms with Markov Chains*, Ann. Math. Artificial Intelligence, 5, pp. 79–88.

R. Salomon (1997), *Improving the performance of genetic algorithms through derandomization*, Software—Concepts and Tools, 18, pp. 175–184.

M. D. Vose and G. E. Liepins (1991), *Punctuated equilibria in genetic search*, Complex Systems, 5, pp. 31–44.

M. D. Vose and A. H. Wright (1995), *Simple genetic algorithms with linear fitness*, Evolutionary Computation, 2, pp. 347–368.

D. H. Wolpert and W. G Macready (1995), *No Free Lunch Theorems for Search*, Tech. Rep. SFI-TR-95-02-010, Santa Fe Institute, Santa Fe, NM.

D. H. Wolpert (1996), *The lack of a priori distinctions between learning algorithms*, Neural Computation, 8, pp. 1341–1390.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 283–299

A MORE EFFICIENT APPROXIMATION SCHEME FOR TREE ALIGNMENT*

LUSHENG WANG†, TAO JIANG‡, AND DAN GUSFIELD§

Abstract. We present a new polynomial time approximation scheme (PTAS) for tree alignment, which is an important variant of multiple sequence alignment. As in the existing PTASs in the literature, the basic approach of our algorithm is to partition the given tree into overlapping components of a constant size and then apply local optimization on each such component. But the new algorithm uses a clever partitioning strategy and achieves a better efficiency for the same performance ratio. For example, to achieve approximation ratios 1.6 and 1.5, the best existing PTAS has to spend time $O(kdn^5)$ and $O(kdn^9)$, respectively, where n is the length of each leaf sequence and d, k are the depth and number of leaves of the tree, while the new PTAS only has to spend time $O(kdn^4)$ and $O(kdn^5)$. Moreover, the performance of the PTAS is more sensitive to the size of the components, which basically determines the running time, and we obtain an improved approximation ratio for each size. Some experiments of the algorithm on simulated and real data are also given.

Key words. approximation algorithm, computational biology, evolutionary tree, multiple sequence alignment, phylogeny

AMS subject classifications. 05C05, 68Q25, 05C90, 81T30, 92B05

PII. S0097539796313507

1. Introduction.

Multiple sequence alignment is one of the fundamental and most challenging problems in computational molecular biology [1, 2, 5, 13]. It plays an essential role in the solution of many problems such as searching for highly conserved subregions among a set of biological sequences and inferring the evolutionary history of a family of species from their molecular sequences. For example, most methods for phylogeny reconstruction based on sequence data assume a given multiple sequence alignment.

An important approach to multiple sequence alignment is the *tree alignment* method. Suppose that we are given k sequences and a *rooted phylogenetic tree* containing k leaves, each of which is labeled with a unique given sequence. The goal is to construct a sequence for each internal node of the tree such that the *cost* of the resulting *fully labeled* tree is minimized. Here, the cost of the fully labeled tree is the total cost of its edges, and the cost of an edge is the *mutational distance* (or *weighted edit distance*) between the two sequences associated with both ends of the edge.

The biological interpretation of the model is that the given tree represents the evolutionary history (known from means other than sequence analysis or postulated

*Received by the editors December 13, 1996; accepted for publication (in revised form) August 3, 1999; published electronically May 2, 2000.

http://www.siam.org/journals/sicomp/30-1/31350.html

†Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong (lwang@cs.cityu.edu.hk). The work of this author was supported in part by Department of Energy grant DE-FG03-90ER60999 and HK CERG grants 9040444, 9040297, and 9040352. Most of this author's work was completed while the author was at UC Davis.

‡Department of Computer Science, University of California, Riverside, CA 92521 (jiang@cs.ucr.edu). The work of this author was supported in part by NSERC research grant OGP0046613, a Canadian Genome Analysis and Technology (CGAT) grant, and a CITO grant. This author's work was completed while the author was visiting the University of Washington and on leave from McMaster University.

§Department of Computer Science, University of California, Davis, CA 95616 (gusfield@cs.ucdavis.edu). The work of this author was supported in part by Department of Energy grant DE-FG03-90ER60999.

in a phylogeny reconstruction experiment) which has created the molecular (DNA, RNA, or amino acid) sequence written at the leaves of the tree. The leaf sequences are ones found in organisms existing today, and the sequences to be determined for the internal nodes of the tree represent inferred sequences that may have existed in the ancestral organisms. It should be emphasized that the tree is almost always binary in biological applications.

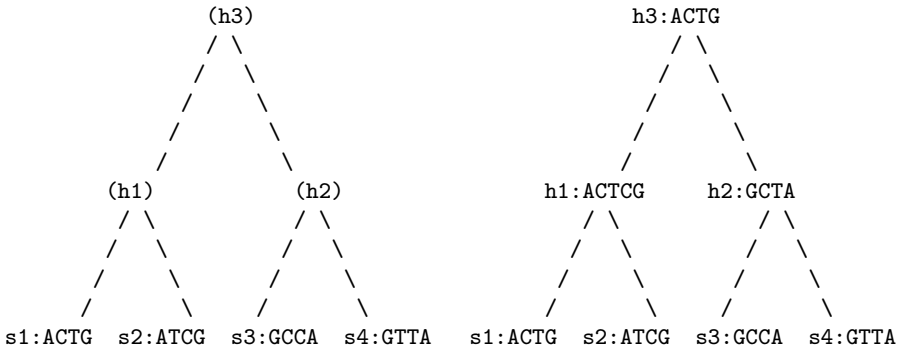
To see how the above is related to multiple sequence alignment, we demonstrate by an example that the sequences inferred for the internal nodes actually induce a mutiple alignment of the given k sequence. Such a multiple sequence alignment is believed to expose evolutionarily significant relationships among the sequences, according to the maximum parsimony principle.

Example 1. Consider four sequences ACTG, ATCG, GCCA, and GTTA, and the tree shown in Figure 1.1(a) connecting these sequences. Suppose that the internal sequences are reconstructed as in (b). For each edge of the tree in (b) we can construct a pairwise alignment of the two sequences associated with the edge, with a cost equal to the mutational distance of the sequences. Then we can induce a multiple alignment that is *consistent* with all six pairwise alignments by "merging" the pairwise alignments incrementally as in [3], taking the spaces into consideration. For example, from the given pairwise alignments of sequences s_1, h_1 and of sequences s_2, h_1, we can obtain a multiple alignment of s_1, s_2, h_1 as shown in (d). The final induced multiple alignment of the four leaf sequences is shown in (e) (if we ignore everything below the dashed line).

Tree alignment is also a key step in a particular phylogeny reconstruction method called *generalized parsimony* [10, 12]. The basic strategy of generalized parsimony is to start with a tree consisting of two leaves and grow the tree by gradually inserting the rest of the leaves (labeled with sequences) at the edges. To add a leaf, we consider insertion at all possible edges of the current tree and use tree alignment to test which edge will result in a most parsimonious tree (i.e., one with the smallest cost).

Tree alignment is known to be NP-hard [14]. Many heuristic algorithms have been proposed in the literature [1, 7, 8, 10, 11], and some approximation algorithms with guaranteed relative error bounds have been reported recently. In particular, a polynomial time approximation scheme (PTAS) is presented in [15], and an improved version is given in [16]. Both PTASs partition the tree into overlapping constant size components, label the leaves of each such component in some way, and then apply local optimization on each component, i.e., compute an optimal tree alignment for each component. The PTAS in [16] achieves an approximation ratio $1 + \frac{2}{t} - \frac{2}{t \times 2^t}$, i.e., it produces a fully labeled tree with cost at most $1 + \frac{2}{t} - \frac{2}{t \times 2^t}$ times the optimal cost, when the running time is $O(kdn^{2^{t-1}+1})$, where k is the number of the given sequences, n is the length of each given sequence, d is the depth of the given phylogeny, and t is a parameter to control the number of sequences involved in a local optimization as well as the performance ratio. For any fixed t, a local optimization aligns a tree with $2^{t-1}+1$ leaves (i.e., sequences), which takes $O(n^{2^{t-1}+1})$ time [9]. Thus the more accurate the algorithm is, the more time it consumes. Since in practice n is at least 100, the bottleneck of the time efficiency is the time spent on local optimization. At present, we can expect to optimally align up to eight sequences of length 200 at a time, as demonstrated by the software package MSA for a similar multiple alignment problem [4]. Thus the above PTASs are still far from being practical.

In this paper, we further improve the PTAS in [16]. The new approximation scheme adopts a more clever partitioning strategy and has a better time efficiency for



(a) The input sequences and tree. (b) The fully labeled tree.

```
s1: ACT G   s2: A TCG   s3: GCCA   s4: GTTA   h1: ACTCG   h2: GCTA
h1: ACTCG   h1: ACTCG   h2: GCTA   h2: GCTA   h3: ACT G   h3: ACTG
```

(c) The pairwise alignments.

```
s1: ACT G
h1: ACTCG
s2: A TCG
```

(d) The alignment of s1, s2, and h1.

```
s1               ACT G
s2               A TCG
s3               GCC A
s4               GTT A
----------------------
h1               ACTCG
h2               GCT A
h3               ACT G
```

(e) The induced multiple alignment.

FIG. 1.1. *Tree alignment induces a multiple alignment.*

the same performance ratio. For any fixed r, where $r = 2^{t-1} + 1 - q$ and $0 \leq q \leq 2^{t-2} - 1$, the new PTAS runs in time $O(kdn^r)$ and achieves an approximation ratio of $1 + \frac{2^{t-1}}{2^{t-2}(t+1)-q}$. Here the parameter r represents the "size" of local optimization. In particular, when $r = 2^{t-1}+1$, its approximation ratio is simply $1+\frac{2}{t+1}$. A comparison of the performance of the new PTAS and the PTAS in [16] for small values of t and r is given in Table 1.1. Note that the new PTAS yields an improved approximation ratio for *every* size of local optimization, whereas the previous PTAS does not. This is because the previous partitioning and analysis method works only when the size of local optimization has the form of one plus some power of two, and the new method works for any size. Hence to achieve a ratio 1.5, the new PTAS requires running time $O(kdn^5)$, while the old PTAS would require running time $O(kdn^9)$.

Although the new PTAS is only a small improvement to the previous PTASs in terms of the feasible approximation ratios it can provide, it still represents a concrete

TABLE 1.1
A comparison of the new PTAS and the best previous PTAS.

r	3	4	5	6	7	8	9
t	2		3				4
Running time	$O(kdn^3)$	$O(kdn^4)$	$O(kdn^5)$	$O(kdn^6)$	$O(kdn^7)$	$O(kdn^8)$	$O(kdn^9)$
Old ratio	1.75		1.58				1.47
New ratio	1.67	1.57	1.50	1.47	1.44	1.42	1.40

step towards constructing a practical approximation algorithm for tree alignment with provable performance bounds, which could be extremely useful in sequence analysis. The new partitioning strategy is also much more flexible and has the potential of leading to even better analytical bounds than what is reported here. We observe that the bounds listed in Table 1.1 do not appear to be tight for either PTAS, and a better analysis technique may also reduce the approximation ratio by a significant amount.

The paper is organized as follows. We review the uniform lifting technique developed in [16] in section 2. Sections 3 and 4 present the new approximation scheme and its analysis, respectively. Finally we describe an implementation of the PTAS and perform some tests on simulated and real data in section 5.

Remark 1.1. There are many ways to define the mutational distance $D(s, s')$ between sequences s and s'. In this paper, we need only assume that the distance is a *metric*. That is, it satisfies

1. *positive definiteness*: $D(s, s') \geq 0$ and $D(s, s) = 0$;
2. *symmetry*: $D(s, s') = D(s', s)$;
3. *triangle inequality*: $D(s, s') \leq D(s, s'') + D(s'', s')$ for any sequence s''.

2. Uniformly lifted trees and the general approach. Before we discuss the new algorithm, let us introduce some useful concepts and results. Let T be a binary (phylogenetic) tree such that each of its leaves is labeled with a given sequence. For convenience, convert T to an *ordered* tree by specifying the children of each internal as left and right children, arbitrarily. A *loaded* tree for T is a tree in which each internal node is also assigned a sequence label (not necessarily a given sequence). A loaded tree is called a *lifted* tree if the sequence label of every internal node v equals the sequence label of some child of v. Figure 2.1(a) exhibits a lifted tree. A lifted tree is called a *uniformly* lifted tree if, for each level, either every internal node at that level receives its sequence label from its left child or every internal node at that level receives its sequence label from its right child. In other words, the lifting choices for the internal nodes at the same level are uniform. Figure 2.1(b) exhibits a uniformly lifted tree.

Let d denote the depth of the tree T. Then there are 2^d different uniformly lifted trees for T. We use a vector $V = (x_1, \ldots, x_d)$ to denote the uniform lifting choices for all levels, where $x_i = R$ or L corresponds to lifting the sequence of the left child or the right child at ith level. For any vector V, $T(V)$ denotes the uniformly lifted tree for T specified by V. For each internal node v, the path from v to the leaf whose sequence label is lifted to v is called the *lifting path* of v. Observe that, every node on the lifting path of v (including v) is assigned the identical sequence.

The following results are proven in [16]. Let $T^{\min}$ denote an optimal loaded tree for T.

THEOREM 2.1. *The average cost of the 2^d uniformly lifted trees for T is at most twice the cost of $T^{\min}$.*

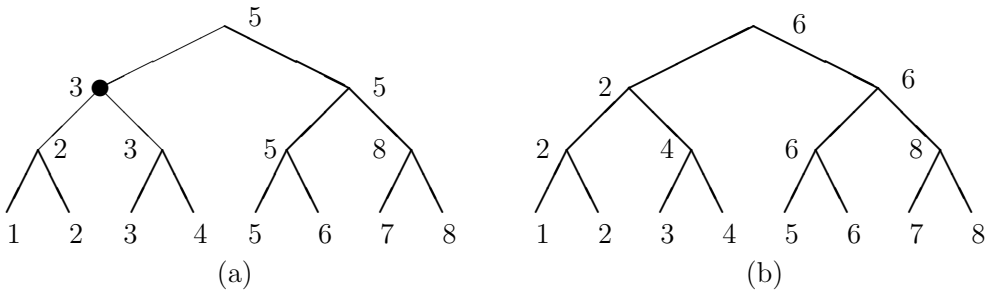


FIG. 2.1. (a) *A lifted tree.* (b) *A uniform lifted tree.*

COROLLARY 2.2. *There exists a uniformly lifted tree for T with a cost at most twice the cost of $T^{\min}$.*

An optimal uniformly lifted tree can be computed in $O(kd + kdn^2)$ time by a straightforward bottom-up dynamic programming, where k is the number of leaves in T and n is the length of each given sequence [16].

Observe that given any lifted tree, we may further reduce its cost by keeping the lifted sequences on some nodes and reconstructing the sequences on the other (internal) nodes to minimize the cost of the edges incident upon these (latter) nodes. For example, based on the lifted sequences 2, 3, 5, we can compute a sequence for the dark circled node in Figure 2.1(a) such that the total cost of the three thin edges is minimized. The new sequence should in general yield a loaded tree with a smaller cost. This suggests the idea of partitioning a (uniformly) lifted tree into a collection of overlapping components, keeping the lifted sequences at the leaves of these components intact, and optimally reconstructing the sequences for the internal nodes in each component, i.e., doing a local optimization on each component. The computation can be done in polynomial time as long as each component has a constant size.

Both PTASs in [15, 16] are based on this idea and partition the tree simply by cutting at levels separated by a fixed constant distance. Our new algorithm also follows the same general approach, but we use a more sophisticated and flexible partitioning strategy.

3. The new partitioning strategy and algorithm. Our algorithm involves partitioning a uniformly lifted tree into many overlapping components, each of which is a binary tree by itself. Let us first describe the structure of the components used in a partition.

A degree-1 node in a tree is called a *terminal*. For a fixed constant r, we consider components of r terminals. If $2^{t-2} + 1 \leq r < 2^{t-1} + 1$ for some positive integer t, a component contains t levels of nodes, where the top level contains only one terminal, called the *head*, the other $t - 1$ levels form a complete binary tree, and the bottom two levels contain $r - 1$ leaves. (See Figure 3.1.) Such a component will be referred to as a *basic component* of a partition, to avoid ambiguity. The terminals (i.e., the head and leaves) of a basic component are called its *boundary nodes*. Note that the child of the head could be either the left or right child. A basic component is of *L-type* (or *R-type*) if it uses the left (or right, respectively) child of the head.

Let T be a phylogeny, V a uniform lifting choice vector, and $T(V)$ the corresponding uniformly lifted tree. Suppose that r is an integer such that $2^{t-2}+1 \leq r \leq 2^{t-1}+1$.

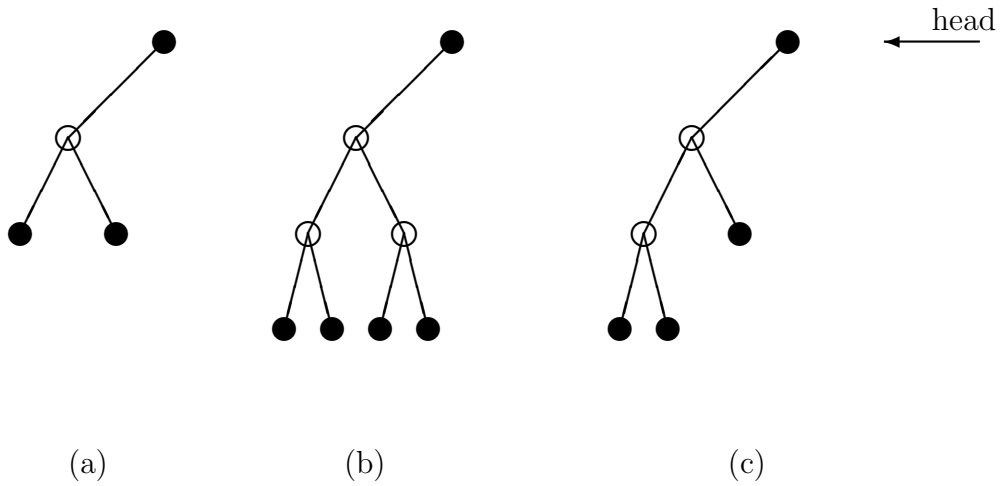


FIG. 3.1. *The L-type basic components for* (a) $r = 3$, (b) $r = 5$, *and* (c) $r = 4$. *The boundary nodes of a basic component are dark circled.*

For every uniform lifting choice vector V, we can obtain t partitions $P_0, P_1, \ldots, P_{t-1}$ of $T(V)$ as follows. Observe that to define a partition, it suffices to specify the heads of the basic components involved and the type of each basic component. First consider $i = 0, 1, \ldots, t-2$.

1. P_i has a (deformed) basic component at the top of $T(V)$, which consists of 2^i leaves at the ith level. (The root of the tree is at level 0, and the ith level is below the $(i-1)$th level. See Figures 3.2 and 3.3.)
2. All nodes on the lifting path of each leaf and each head of every basic component are heads, and the leaves of the basic components are heads.
3. The type of each basic component is identical to the lifting choice at its head as given in V. In other words, the basic component is of L-type (R-type) iff its head receives its sequence from the left child.

The partition P_{t-1} is defined similarly except that its top basic component part is a complete binary tree with $r-1$ leaves instead. Figures 3.2, 3.3, and 3.4 give an example of the partitions P_0, P_1, P_2 when $r = 4$.

Given a partition P_i, if we preserve the sequences uniformly lifted to all the boundary nodes and optimally reconstruct the sequences on rest of the nodes (i.e., the internal nodes of each basic component) to minimize the cost of each basic component, we obtain a loaded tree, which will simply be called an (r, i)-*tree*. We use $T(V)_{r,i}$ to denote the (r, i)-tree obtained from the uniformly lifted tree $T(V)$. An *optimal* r-*tree* is some (r, i)-tree $T(V)_{r,i}$ with the smallest cost among all possible i and V. For any loaded tree T_1, $C(T_1)$ denotes its cost. For any tree T_1, denote the set of internal nodes of T_1 as $I(T_1)$ and the set of leaves of T_1 as $L(T_1)$, and for each $a \in L(T_1)$, T_1^a denotes the unique $(r, 0)$-tree for T_1 such that the sequence label of leaf a is lifted to the root of T_1.

Now let us begin to describe our algorithm. We will first assume that T is a full binary tree and then extend the construction to arbitrary binary trees. Let v be a node of T. The subtree of T rooted at v consisting of all descendants of v is denoted as T_v. Note that, since we assume that T is a full binary tree, lifting the sequence of a, for any $a \in L(T_v)$, to v uniquely determines a uniform lifting choice for T_v.

Let T' be the top basic component of T_v^a and v' the child of v that is on the

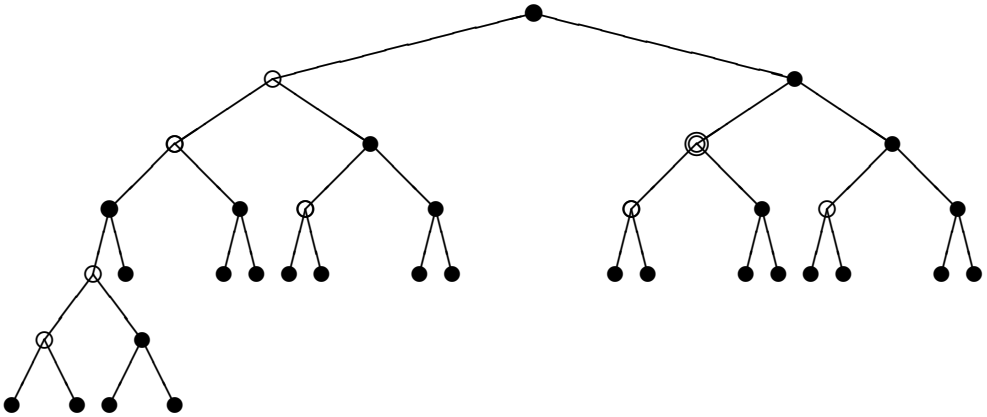


FIG. 3.2. *The partition P_0 when $r = 4$. The uniform lifting choice is $(R, R, \ldots, R)$. The dark circled nodes are boundary nodes of the basic components.*

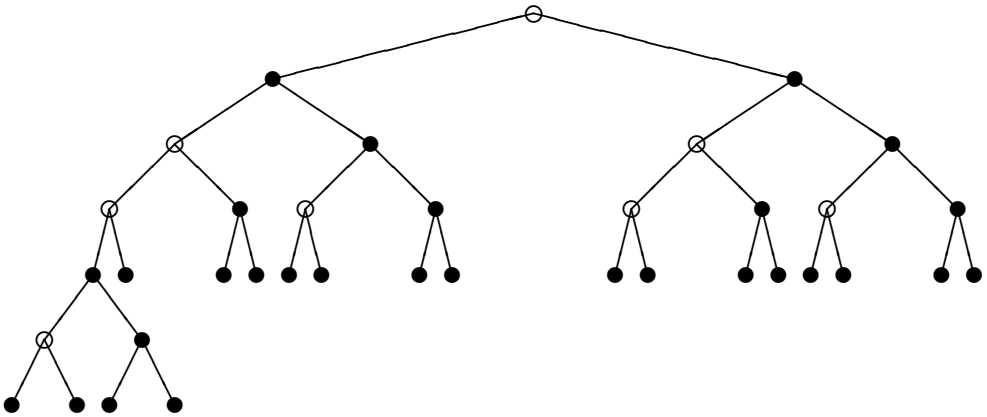


FIG. 3.3. *The partition P_1 when $r = 4$. The uniform lifting choice is $(R, R, \ldots, R)$.*

lifting path of v in T_v^a. Observe that a uniquely determines the sequences lifted to the boundary nodes of T'. The cost of the $(r, 0)$-tree for T_v^a can be computed using the recurrence equation

$$C(T_v^a) = C(T') + \sum_{u \in L(T')} C(T_u^{a(u)}) + C(T_{v'}^a), \tag{3.1}$$

where $a(u)$ is the leaf whose sequence is lifted to u and $a(u)$ is uniquely determined by a. (See Figure 3.5.)

Hence we can compute the values $C(T_v^a)$ for all v and a inductively by traversing the tree bottom-up. Note that, for each pair v and a, computing (3.1) requires $O(r + n^r)$ time, where n is the length of the sequences, given the value of $C(T_u^{a(u)})$ for each $u \in L(T')$ and the value of $C(T_{v'}^a)$. Thus, computing $C(T_v^a)$ for all pairs v and a requires $O((r + n^r) \cdot \sum_{v \in T} |L(T_v)|)$ time. Since each leaf a appears in at most d different $L(T_v)$s, where d is the depth of T, totally we need $O(rkd + kdn^r)$ time. Similarly, the cost of an (r, i)-tree $T_{r,i}^a$ obtained by lifting the sequence label of leaf a

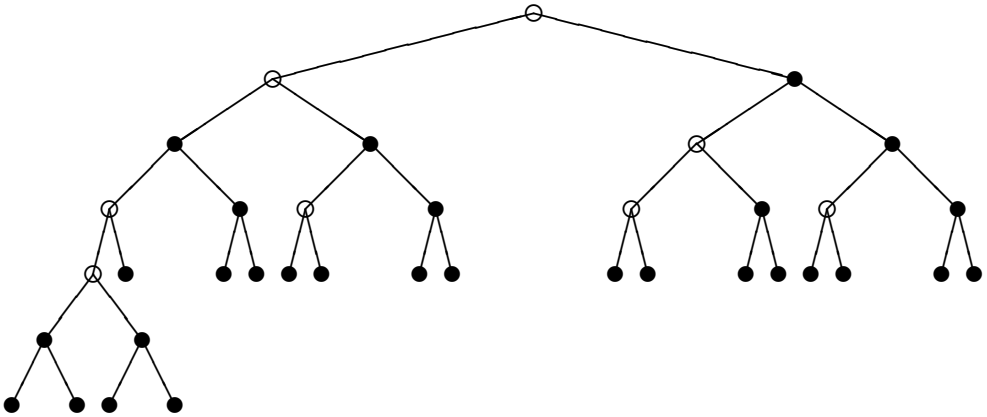


FIG. 3.4. *The partition P_2 when $r = 4$. The uniform lifting choice is $(R, R, \ldots, R)$.*

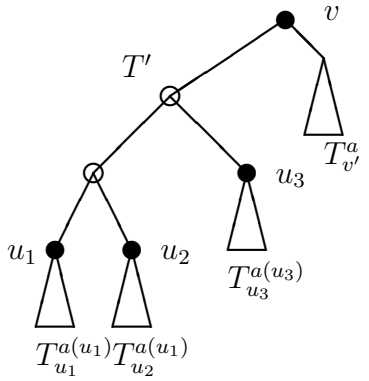


FIG. 3.5. *An illustration for the recurrence equation* (3.1).

to the root of T can be computed as follows:

$$C(T^a_{r,i}) = C(T'_i) + \sum_{v \in L(T'_i)} C(T^{a(v)}_v), \tag{3.2}$$

where T'_i is the top basic component of partition P_i and $a(v)$ is the leaf whose sequence is lifted to v as determined by the choice of a.

The above algorithm only works for full binary trees since, in general, lifting from a leaf a to a node v does not completely determine the uniform lifting choice for the subtree T_v if a is not at the bottom level of T_v. In particular, in arbitrary binary trees, lifting from leaf a to node v does not uniquely determine the leaf $a(u)$ being lifted to the node u for any internal node u that is lower than v, hence invalidating the recurrence equation (3.1).

To extend the algorithm to arbitrary binary trees without losing any time efficiency, we need the notion of an *extension* of a phylogeny. An unlabeled (ordered) tree T_1 *extends* another unlabeled (ordered) tree T_2 if T_2 can be obtained from T_1 by contracting some subtrees to single leaves. Clearly, if T_1 extends T_2, then each leaf of T_2 uniquely corresponds to a subtree of T_1. The *minimal extension* of a set of unlabeled trees is the (unique) unlabeled tree with the fewest edges that extends every tree in

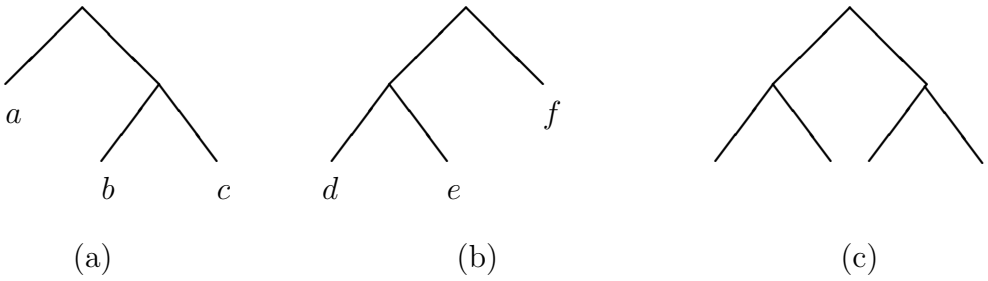


FIG. 3.6. *The minimal extension of the structures of the trees in* (a) *and* (b) *is shown in* (c).

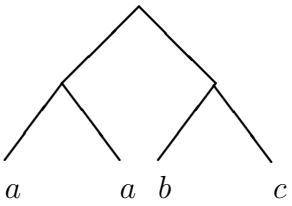


FIG. 3.7. *An extension of the tree in Figure* 3.6(a).

the set. A (partially) leaf-labeled tree T_1 extends another (partially) leaf-labeled tree T_2 if (i) the structure of T_1 extends the structure of T_2; and (ii) for each leaf v of T_2, the subtree of T_1 corresponding to v has all its leaves assigned the same label as that of v. Figure 3.6 gives an example of the minimal extension of two unlabeled trees, and Figure 3.7 shows an extension of the leaf-labeled tree in given Figure 3.6(a).

Now we are ready to generalize the algorithm to an arbitrary binary tree T. Let v be an internal node of T. Denote $T_{v,t-1}$ as the tree consisting of the top t levels of the subtree T_v. To compute $C(T_v^a)$ for each $a \in L(T_v)$, we first extend the tree $T_{v,t-1}$, which is in general a partially leaf-labeled tree, to a full binary (partially leaf-labeled) tree $T'_{v,t-1}$ with 2^{t-1} leaves. Let $v_1, \ldots, v_{2^{t-1}}$ be the leaves of this extended tree. For each v_i, $i = 1, \ldots, 2^{t-1}$, we extend T_{v_i} to obtain a leaf-labeled tree T'_{v_i} such that the structure of T'_{v_i} is the minimal extension of the structures of all $T_{v_1}, \ldots, T_{v_{2^{t-1}}}$. Note that $T'_{v,t-1}$ has 2^{t-1} leaves $v_1, \ldots, v_{2^{t-1}}$ that are the roots of subtrees $T'_{v_1}, \ldots, T'_{v_{2^{t-1}}}$. Denote the tree containing $T_{v,t-1}$ at the top and $T'_{v_1}, \ldots, T'_{v_{2^{t-1}}}$ at the bottom as $ET(v)$. We compute the cost $C(ET(v)^a)$ for each $a \in L(ET(v))$ first.

Suppose that $C(T_u^a)$ has been computed for each $u \in T_{v,t-1}$, where $u \neq v$ and $a \in L(T_u)$. We can easily compute the cost $C(ET(v)_u^a)$ of the $(r,0)$-tree $ET(v)_u^a$ for each $u \in T_{v,t-1}$, where $u \neq v$ and $a \in L(ET(v)_u)$ as follows:

$$C(ET(v)_u^a) = C(T_u^b), \qquad \text{where } a \in L(ET(v)_b). \tag{3.3}$$

Observe that the subtrees $T'_{v_1}, \ldots, T'_{v_{2^{t-1}}}$ of $ET(v)$ all have the same structure. Thus, lifting the sequence of a, for any $a \in L(ET(v))$, to v uniquely determines from where each node of $ET(v)_{v,t-1}$ receives its lifted sequence. Since $ET(v)_{v,t-1}$ extends the top basic component of $ET(v)^a$, lifting the sequence of leaf a, $a \in L(ET(v))$, to v uniquely determines from where each leaf of the top basic component of $ET(v)^a$ receives its lifted sequence. Therefore, we can compute the cost $C(ET(v)^a)$ for each $a \in L(ET(v))$ using the recurrence equation (3.1).

```
1. begin
2. for  each level i, with the bottom level first
3.    for each node v at level i
4.    begin
5.       Construct the extended tree ET(v).
6.       for each u ∈ T_{v,t-1}
7.          Compute C(ET(v)^a_u) for each a ∈ L(ET(v)_u) using equation 3.3.
8.       for each leaf a ∈ L(ET(v))
9.          Compute C(ET^a_v) using equation 3.1.
10.      for each leaf a ∈ L(T(v))
11.         Compute C(T^a_v) using equation 3.4.
12.   end
13. for each i = 0, 1, ..., t − 1
14.    Compute C(T^a_{r,i}) for every a ∈ L(T).
15. Find i, a so that C(T^a_{r,i}) is minimized.
16. Compute a loaded tree from C(T^a_{r,i}) by back-tracing.
17. end.
```

FIG. 3.8. *The algorithm to compute an optimal r-tree.*

Once we have the cost $C(ET(v)^a)$ for each $a \in L(ET(v))$, we can compute $C(T_v^a)$ easily for each $a \in L(T_v)$ by "reversing" (3.3) as follows:

$$C(T_v^a) = \min_{b \in L(ET(v)_a)} C(ET(v)^b). \tag{3.4}$$

The time complexity of the algorithm can be analyzed as follows. Since the structure of T_v extends the structure of all subtrees $T_{v_1}, \ldots, T_{v_{2^{t-1}}}$, $|L(T'_{v_i})| \leq |L(T_v)|$ for each i, $1 \leq i \leq 2^{t-1}$. Thus, the extended tree $ET(v)$ can be computed in $O(r|L(T_v)|)$ time for each $v \in T$. Hence totally it requires at most $O(\sum_{v \in T} r|L(T_v)|)$ time to construct all the $ET(v)$s. Since each leaf of T appears in at most d $L(T_v)$s, $O(\sum_{v \in T} r|L(T_v)|) = O(rd|L(T)|) = O(rkd)$. Computing (3.3) and (3.4) takes merely a traversal of $ET(v)$ each. Hence the algorithm still runs in time $O(rkd + kdn^r)$ on arbitrary binary trees. The complete algorithm is summarized in Figure 3.8.

4. The analysis of the algorithm. Given an arbitrary binary phylogeny T, we can extend T into a full binary tree $\hat{T}$. Obviously, any loaded tree for T can be extended to a loaded tree with the same cost for $\hat{T}$. Conversely, given any loaded tree for $\hat{T}$, we can obtain a loaded tree for T with equal or smaller cost by pruning appropriate subtrees and contracting nodes with only one child. The last operation will not increase the cost by the triangle inequality. Thus for the analysis we may assume that the given tree T is a full binary tree. For convenience, we number the levels from top to bottom. That is, the root is at level 0, and the level below level i is level $i + 1$.

First, let us find a good upper bound for the cost of an (r, i)-tree $T(V)_{r,i}$ for an arbitrary uniform lifting choice V. Again, let $T^{\min}$ be an optimal loaded tree for T. For each node $v \in T$, let $s(v)$ denote the sequence label of v in $T^{\min}$. We can modify $T(V)_{r,i}$ by replacing the sequence label of each nonterminal node with the sequence $s(v)$ to obtain a loaded tree $T(V)'_{r,i}$. Clearly, the cost of $T(V)'_{r,i}$ is at least that of $T(V)_{r,i}$. So, it suffices to upper bound the cost of $T(V)'_{r,i}$.

There are four types of edges in the loaded tree $T(V)'_{r,i}$:

1. the edges whose both ends are assigned an identical lifted sequence;
2. the edges with ends assigned distinct lifted sequences;
3. the edges whose both ends are assigned sequences from $T^{\min}$;
4. the edges with one end assigned a sequence from $T^{\min}$ and the other assigned a lifted sequence.

(See again Figure 3.2.) Obviously each type 1 edge costs zero and each type 3 edge (u, v) costs $D(s(u), s(v))$. Let (u, v) be a type 4 edge, where u is assigned a sequence from $T^{\min}$ and v is assigned a lifted sequence according to V. The cost of the edge (u, v) is upper bounded by

$$D(s(u), s(v)) + C(P_{v,V}),$$

where $P_{v,V}$ denotes the lifting path of node v in the uniformly lifted tree $T(V)$ and $C(P_{v,V})$ is the cost of the path $P_{v,V}$ in the optimal tree $T^{\min}$. Observe that type 2 edges may only appear at the bottom level of the tree. Hence a type 2 edge can be viewed as a degenerate type 4 edge, with the lower end (which is a leaf) of the edge being the node assigned a sequence from $T^{\min}$, and can be treated similarly to the type 4 edges. For convenience, call $D(s(u), s(v))$ and $C(P_{v,V})$ the *first* and *second* parts of the cost of edge (u, v) in $T(V)$, respectively. For each internal node v in $T(V)_{r,i}$, the path from v to the leaf whose sequence label is lifted to v is called the *lifting path* of v in $T(V)_{r,i}$. Call each internal node on a lifting path in $T(V)_{r,i}$ of some node v a *lifted* node. A lifted node is *heavy* if it is involved in two type 4 edges, and is *light* if it is involved in one type 4 edge. For a light node v, the cost of the lifted path $P_{v,V}$ is charged once in the above upper bound as the second part, whereas for a heavy node v, the cost of the lifted path $P_{v,V}$ is charged twice. A *maximal* lifting path is a lifting path in $T(V)_{r,i}$ which cannot be further extended. Notice that each maximal lifting path contains at most one heavy node at the upper end of the path. Since every edge in a maximal lifting path has type 1 and thus zero cost, we can charge one of the $C(P_{v,V})$s for a heavy node v to $C(T^{\min})$, obtaining the following upper bound for the cost $T(V)_{r,i}$:

$$C(T(V)_{r,i}) \le C(T^{\min}) + \sum_{v \text{ is a lifted node}} C(P_{v,V}). \tag{4.1}$$

To further bound the total cost of the lifting paths, we need the following lemmas.

LEMMA 4.1 (see [15, 16]). *Let T be a binary tree such that every internal node has exactly two children. For each leaf $l \in L(T)$, there exists a mapping π_l from the internal nodes of T to its leaves such that* (i) *for each internal node v, $\pi(v)$ is a descendant leaf of v,* (ii) *the paths from nodes v to their images $\pi(v)$ are edge-disjoint, and* (iii) *moreover, there is an* unused *path from the root of T to the leaf l that is edge-disjoint from all the paths in* (ii).

In other words, Lemma 4.1 says that a binary tree can be decomposed into a set of edge-disjoint paths from internal nodes to leaves, with one path for each nonroot internal node and two paths for the root. Let $C(\pi)$ denote the cost of the paths induced by mapping π in the tree $T^{\min}$.

LEMMA 4.2. *There exist mappings $\{\pi_l | l \in L(T)\}$ such that*

$$\sum_{V} \sum_{v \in I(T)} C(P_{v,V}) \le \sum_{l \in L(T)} C(\pi_l) \le 2^d C(T^{\min}),$$

where d is the depth of T.

Proof. We prove it by induction on d, the depth of the tree. Recall that T is a full binary tree. The lemma obviously holds for $d = 1$. Suppose that it holds for $d \leq q$. Consider a tree T with $d = q + 1$. Let v_0 be the root of T and v_L and v_R the left and right children of v_0. Recall that T_{v_L} and T_{v_R} denote the subtrees rooted at v_L and v_R, respectively. Then the depth of T_{v_j} $(j = L, R)$ is q. Let $\mathcal{V}$ be the set of lifting choices for T, and $\mathcal{V}_j$ the set of lifting choices for T whose first component is j. Clearly $\mathcal{V} = \mathcal{V}_L \cup \mathcal{V}_R$. By the inductive hypothesis, for each $j = L, R$, there exist mappings $\{\pi_l | l \in L(T_{v_j})\}$ for the subtree T_{v_j} such that

$$\sum_{V \in \mathcal{V}_j} \sum_{v \in I(T_{v_j})} C(P_{v,V}) \leq \sum_{l \in L(T_{v_j})} C(\pi_l).$$

For each mapping π_l, $l \in L(T_{v_j})$, there is an unused path from v_j to the leaf l of T_{v_j}. For any leaf l, let P_l be the path from the root v_0 of T to the leaf l. Number the leaves of T from left to right as $l_1, \ldots, l_{2^{q+1}}$. It is easy to see that for any $h = 1, \ldots, 2^q$, the mappings π_{l_h} and $\pi_{l_{2^q+h}}$ and the path P_{l_h} form a mapping for tree T, denoted $\pi^{h,1}$, with an unused path from v_0 to l_{2^q+h}, and symmetrically the mappings π_{l_h} and $\pi_{l_{2^q+h}}$ and the path $P_{l_{2^q+h}}$ form a mapping for tree T, denoted $\pi_{h,2}$, with an unused path from v_0 to l_h. Thus, we have

$$\begin{aligned}
\sum_{V \in \mathcal{V}} \sum_{v \in I(T)} C(P_{v,V}) &= 2 \cdot \left[\sum_{j=L}^{R} \sum_{V \in \mathcal{V}_j} \sum_{v \in I(T_{v_j})} C(P_{v,V}) \right] + \sum_{V \in \mathcal{V}} C(P_{v_0,V}) \\
&\leq \sum_{h=1}^{2^{q+1}} 2 \cdot C(\pi_{l_h}) + C(P_{l_h}) \\
&\leq \sum_{h=1}^{2^q} \sum_{j=1}^{2} C(\pi_{l_h}) + C(\pi_{l_{2^q+h}}) + C(P_{l_{(j-1)2^q+h}}) \\
&\leq \sum_{h=1}^{2^q} \sum_{j=1}^{2} C(\pi_{h,j}) \\
&\leq 2^{q+1} C(T^{\min}). \qquad \square
\end{aligned}$$

Since, there are 2^d distinct uniform lifting choices, the average cost of each $\sum_{v \in I(T)} C(P_{v,V})$ is at most $C(T^{\min})$. Observe that Theorem 2.1 follows immediately from inequality (4.1) and Lemma 4.2, since in a uniformly lifted tree $T(V)$, every internal node is a lifted node. Now we are ready to derive the required upper bound for an optimal r-tree and hence the performance ratio of our approximation algorithm. To simplify the presentation, consider first the case when $r = 2^{t-1} + 1$ for some $t \geq 1$.

4.1. The performance ratio when $r = 2^{t-1}+1$. Let $y_i(j)$ denote the number of boundary nodes at level j of an (r, i)-tree, where $0 \leq i \leq t-1$ and $0 \leq j \leq d$. It is easy to see that $y_0(j)$ can be computed by the recurrence equation

$$y_0(j) = y_0(j-1) + 2^{t-1} y_0(j-t), \tag{4.2}$$

where $t \leq j \leq d$. The initial values are $y_0(0) = \cdots = y_0(t-1) = 1$.

Let $x_i(j) = \frac{y_i(j)}{2^j}$ be the fraction of the nodes on level j of an (r,i)-tree that are boundary nodes, where $0 \le i \le t-1$ and $0 \le j \le d$. From (4.2), we have

$$x_0(j) = \frac{x_0(j-1) + x_0(j-t)}{2}, \tag{4.3}$$

where $t \le j \le d$. The initial values are $x_0(j) = 2^{-j}$ for each $j = 0, 1, \ldots, t-1$. Moreover, it is easy to see that

$$x_i(j) = x_{i-1}(j-1) = \cdots = x_0(j-i) \tag{4.4}$$

for any $i \le j \le d$ and $x_i(j) = 0$ for any $0 \le j < i$. The next lemma is a key to our bound. This gives a complete recurrence relation for all $x_i(j)$.

LEMMA 4.3.

$$2x_0(j) + \sum_{i=1}^{t-1} x_i(j) \le 2.$$

Proof. Define $f(j) = 2x_0(j) + \sum_{i=1}^{t-1} x_i(j)$. From (4.3) and (4.4), we obtain

$$f(j+1) - f(j) = 2x_0(j+1) - x_0(j) - x_0(j-t+1) = 0$$

for $j \ge t-1$. Based on the given initial values, we have

$$f(j) = f(t-1) = 2x_0(t-1) + x_0(t-2) + \cdots + x_0(0) = 2$$

for $j \ge t-1$. It is also easy to see that $f(j) < f(t-1)$ for any $j < t-1$. ∎

Combining inequality (4.1) and Lemmas 4.2 and 4.3, we have the following lemma.

LEMMA 4.4.

$$2 \cdot \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,0}) + \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,1}) + \cdots + \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,t-1}) \le (t+1)2^d C(T^{\min}) + 2 \cdot 2^d C(T^{\min}).$$

The left-hand side of the above inequality consists of $(t+1)2^d$ distinct (r,i)-trees. That is, the average cost of each of these (r,i)-trees is $1 + \frac{2}{t+1} C(T^{\min})$. Thus we can conclude that there exists an (r,i)-tree with cost at most $1 + \frac{2}{t+1} C(T^{\min})$.

THEOREM 4.5. *When $r = 2^{t-1} + 1$, the performance ratio of the algorithm in Figure 3.8 is $1 + \frac{2}{t+1}$.*

4.2. The performance ratio for an arbitrary *r*. Assume that $r = 2^{t-1} + 1 - q$ for some integer q, $0 \le q < 2^{t-2}$. Define variables $y_i(j)$ and $x_i(j)$ as before, and consider $y_0(j)$ and $x_0(j), j \ge t$ first. There are three types of boundary nodes at level j. Recall that each basic component consists of a head and a complete binary tree with $r-1$ leaves.

1. There are $(2^{t-1} - 2q)y_0(j-t)$ boundary nodes which are at the lowest level of some basic components.
2. There are $qy_0(j-t+1)$ boundary nodes which are at the second lowest level of some basic components.
3. There are $y_0(j-1)$ remaining boundary nodes which are on some lifting paths.

The above three types of boundary nodes are disjoint. (Note that each node in classes 1 and 2 is always shared by two adjacent basic components, whereas each node in class 3 is only involved in one basic component.) Therefore, we have recurrence equation

$$y_0(j) = y_0(j-1) + qy_0(j-t+1) + (2^{t-1} - 2q)y_0(j-t)$$

for $j \geq t$. Hence

$$x_0(j) = \frac{1}{2}x_0(j-1) + \frac{2q}{2^t}x_0(j-t+1) + \frac{2^{t-1}-2q}{2^t}x_0(j-t) \tag{4.5}$$

for $j \geq t$. The initial values are

$$x_0(j) = \frac{1}{2^j} \quad \text{for} \quad j = 0, 1, \ldots, t-2,$$
$$x_0(t-1) = \frac{q+1}{2^{t-1}}.$$

Again, it is easy to see that

$$x_i(j) = x_{i-1}(j-1) \tag{4.6}$$

for any $i = 1, \ldots, t-2$, and

$$y_{t-1}(j) = (2^{t-1} - 2q)y_0(j-t+1) + qy_0(j-t+2).$$

From the last equation, we have

$$x_{t-1}(j) = \frac{2^{t-1}-2q}{2^{t-1}}x_0(j-t+1) + \frac{2q}{2^{t-1}}x_0(j-t+2). \tag{4.7}$$

Observing that $x_i(j) = 0$ for any $j < i$, the above gives a complete recursive definition of all $x_i(j)$.

LEMMA 4.6.

$$2^t x_0(j) + 2^{t-1}x_1(j) + \cdots + 2^{t-1}x_{t-3}(j) + (2^{t-1} - 2q)x_{t-2}(j) + 2^{t-1}x_{t-1}(j) \leq 2^t.$$

Proof. Define

$$f(j) = 2^t x_0(j) + 2^{t-1}x_1(j) + \cdots + 2^{t-1}x_{t-3}(j) + (2^{t-1} - 2q)x_{t-2}(j) + 2^{t-1}x_{t-1}(j).$$

From (4.6) and (4.7), we have, for $j \geq t-1$,

$$f(j) = 2^t x_0(j) + 2^{t-1}x_0(j-1) + \cdots + 2^{t-1}x_0(j-t+2) + (2^{t-1} - 2q)x_0(j-t+1). \tag{4.8}$$

From (4.5) and (4.8), we obtain $f(j+1) - f(j) = 0$. Plugging the initial values of $x_0(j)$ in (4.8), we have

$$\begin{aligned} f(j) &= f(t-1) \\ &= 2^t x_0(t-1) + 2^{t-2}x_0(j-1) + \cdots + 2^{t-1}x_0(1) + (2^{t-1} - 2q)x_0(0) \\ &= 2(q+1) + 2 + \cdots + 2^{t-2} + (2^{t-1} - 2q) = 2^t \end{aligned}$$

for any $j \geq t-1$. It is easy to verify that $f(j) < f(t-1)$ for any $j < t-1$. ∎

Similar to Lemma 4.4, we have the following lemma.

LEMMA 4.7.

$$\begin{aligned} 2^t \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,0}) + 2^{t-1} \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,1}) + \cdots + 2^{t-1} \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,t-3}) \\ + \quad (2^{t-1} - 2q) \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,t-2}) + 2^{t-1} \sum_{V=(L,\ldots,L)}^{(R,\ldots,R)} C(T(V)_{r,t-1}) \\ \leq \quad (2^{t-1}(t+1) - 2q) 2^d C(T^{\min}) + 2^t \cdot 2^d C(T^{\min}). \end{aligned}$$

Noting that the left-hand side of the above inequality consists of $(2^{t-1}(t+1)-2q)2^d$ distinct (r,i)-trees, we have the main theorem of this section.

THEOREM 4.8. *Suppose that* $r = 2^{t-1} + 1 - q$, *where* $0 \leq q < 2^{t-2}$. *The performance ratio of the algorithm in Figure* 3.8 *is* $1 + \frac{2^{t-1}}{2^{t-2}(t+1)-q}$.

5. Discussions.

5.1. Further improvement. Note that an approximate solution produced by the algorithm in Figure 3.8 still uses sequences lifted from the leaves as sequence labels for some internal nodes. Such a solution can be further improved with the iterative method proposed by Sankoff, Cedergren, and Lapalme [10].

To illustrate the iterative method in [10], consider the phylogeny in Figure 5.1, which contains nine given species on its nine leaves. To improve the cost of a loaded tree, we divide the phylogeny into seven 3-components as shown in Figure 5.1, each consisting of a center and three terminals. Local optimization is done for every 3-component based on the labels of its three terminals. The new center label can then be used to update the center label of an overlapping 3-component. The algorithm converges since each local optimization reduces the cost of the tree by at least one. Thus if the process is repeated often enough, every 3-component will become optimal. Empirical results show that the algorithm produces a reasonably good loaded tree within five iterations [10].

If we are able to handle local optimization of size r, $r \geq 3$, we may extend the above iterative method to components with r terminals. Augmenting the algorithm in Figure 3.8 with the iterative method should result in an improved performance. We do not have an error bound analysis for the combined method at this point that is better than the bound given in section 4. However, there is a good way to estimate the error bound of this combined method for a particular input.

5.2. Estimating the error bound for an instance. Theorem 2.1 says that the average cost of the uniformly lifted trees is at most twice that of an optimal solution. Thus, a half of the average cost is a lower bound of the optimal cost. Let $C_{avg}(I)$ be the cost of the average cost of the uniformly lifted trees for some instance I, and let $X(I)$ be the cost of the solution obtained by the combination of the approximation algorithm in Figure 3.8 and the iterative method above for the input I. Then $X(I)$ is at most $\frac{2X(I)}{C_{avg}(I)}$ times the optimum. This suggests a way to estimate the error bound for a particular instance. In [6], an algorithm to compute the average cost of the uniformly lifted trees is proposed. The running time of the algorithm is $O(kdn^2)$. It is demonstrated that this lower bound is reasonably tight.

5.3. Some experimental results. We have implemented our algorithm in C and run the program on a small example from [10]. The given tree topology is shown in Figure 5.1. Each terminal is labeled with an RNA S_5 sequence. The score scheme

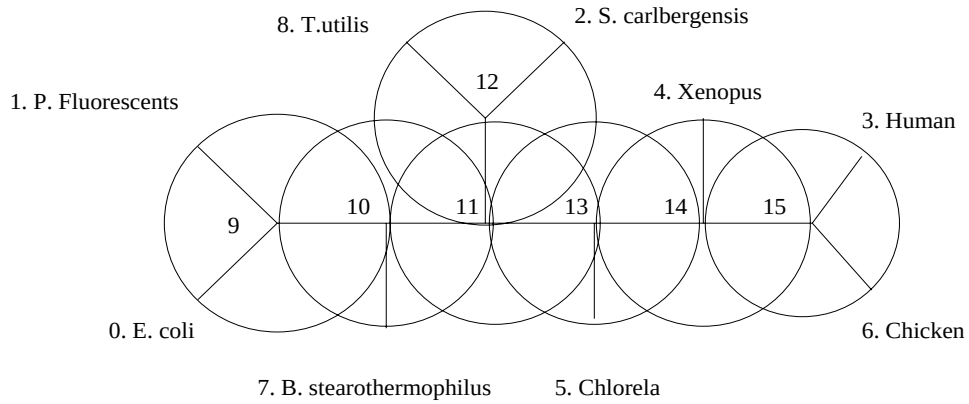


FIG. 5.1. *A phylogeny with nine species, which is divided into seven* 3*-components.*

is the same as in [10]. For $r = 3$, we select the root in the middle of the edge (13,14) and the cost of the corresponding loaded tree is 332.75. For $r = 4$, the same root gives a loaded tree with cost 322.5. Such a cost changes a bit if we try other roots. The average cost of the uniformly lifted trees is 461.6 [6]. Thus we can conclude that the costs of the solutions for $r = 3$ and 4 are at most 1.442 and 1.397 times the optimal for this example. We then use the iterative method of [10] to further improve the solution. The costs are further reduced from 332.75 and 322.5 to 321.25 and 298.25, respectively.

REFERENCES

[1] S. F. ALTSCHUL AND D. J. LIPMAN, *Trees, stars, and multiple biological sequence alignment*, SIAM J. Appl. Math., 49 (1989), pp. 197–209.

[2] S. CHAN, A. WONG, AND D. CHIU, *A survey of multiple sequence comparison methods*, Bull. Math. Biol., 54 (1992), pp. 563–598.

[3] D. FENG AND R. DOOLITTLE, *Progressive sequence alignment as a prerequisite to correct phylogenetic trees*, J. Mol. Evol., 25 (1987), pp. 351–360.

[4] S. GUPTA, J. KECECIOGLU, AND A. SCHAFFER, *Making the shortest-paths approach to sum-of-pairs multiple sequence alignment more space efficient in practice*, in Proceedings of the 6th Combinatorial Pattern Matching Conference, 1995, Springer-Verlag, Berlin, pp. 128–143.

[5] D. GUSFIELD, *Efficient methods for multiple sequence alignment with guaranteed error bounds*, Bull. Math. Biol., 55 (1993), pp. 141–154.

[6] D. GUSFIELD AND L. WANG, *New uses for uniform lifted alignment*, in Mathematical Supports for Molecular Biology, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 47, AMS, Providence, RI, 1999, pp. 33–51.

[7] J. HEIN, *A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given*, Mol. Biol. Evol., 6 (1989), pp. 649–668.

[8] R. RAVI AND J. KECECIOGLU, *Approximation algorithms for multiple sequence alignment under a fixed evolutionary tree*, in Proceedings of the 6th Combinatorial Pattern Matching Conference, 1995, Springer-Verlag, Berlin, pp. 330–339.

[9] D. SANKOFF, *Minimal mutation trees of sequences*, SIAM J. Appl. Math., 28 (1975), pp. 35–42.

[10] D. SANKOFF, R. J. CEDERGREN, AND G. LAPALME, *Frequency of insertion-deletion, transversion, and transition in the evolution of* 5*S ribosomal RNA*, J. Mol. Evol., 7 (1976), pp. 133–149.

[11] D. SANKOFF AND R. CEDERGREN, *Simultaneous comparisons of three or more sequences related by a tree*, in Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison, D. Sankoff and J. Kruskal, eds., Addison-Wesley, Reading, MA, 1983, pp. 253–264.

[12] D. Swofford and G. Olson, *Phylogenetic reconstruction*, in Molecular Systemtics, D. Hillis and C. Moritz, eds., Sinauer Associates, Sunderland, MA, 1990.
[13] M. S. Waterman and M. D. Perlwitz, *Line geometries for sequence comparisons*, Bull. Math. Biol., 46 (1984), pp. 567–577.
[14] L. Wang and T. Jiang, *On the complexity of multiple sequence alignment*, J. Comput. Biol., 1 (1994), pp. 337–348.
[15] L. Wang, T. Jiang, and E. L. Lawler, *Approximation algorithms for tree alignment with a given phylogeny*, Algorithmica, 16 (1996), pp. 302–315.
[16] L. Wang and D. Gusfield, *Improved approximation algorithms for tree alignment*, J. Algorithms, 25 (1997), pp. 255–273.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 300–317

BEYOND COMPETITIVE ANALYSIS*

ELIAS KOUTSOUPIAS† AND CHRISTOS H. PAPADIMITRIOU‡

Abstract. The competitive analysis of online algorithms has been criticized as being too crude and unrealistic. We propose refinements of competitive analysis in two directions: The first restricts the power of the adversary by allowing only certain input distributions, while the other allows for comparisons between information regimes for online decision-making. We illustrate the first with an application to the paging problem; as a byproduct we characterize completely the work functions of this important special case of the k-server problem. We use the second refinement to explore the power of lookahead in server and task systems.

Key words. online algorithms, competitive analysis, paging problem, metrical task systems

AMS subject classifications. 68Q05, 68Q25

PII. S0097539796299540

1. Introduction.

The area of *online algorithms* [14, 10] shares with complexity theory the following characteristic: Although its importance cannot be reasonably denied (an algorithmic theory of decision-making under uncertainty is of obvious foundational significance and practical relevance), certain aspects of its basic premises, modeling assumptions, and results have been widely criticized with respect to their realism and relation to computational practice. In this work we revisit some of the most often voiced criticisms of *competitive analysis* (the basic framework within which online algorithms have been heretofore studied and analyzed) and propose and explore some better-motivated alternatives.

In competitive analysis, the performance of an online algorithm is compared against an all-powerful adversary on a worst-case input. The *competitive ratio* of a problem—the analogue of worst-case asymptotic complexity for this area—is defined as

$$R = \min_A \max_x \frac{A(x)}{\mathrm{opt}(x)}. \tag{1}$$

Here A ranges over all online algorithms, x ranges over all "inputs," opt denotes the optimum offline algorithm, while $A(x)$ is the cost of algorithm A when presented with input x. This clever definition is both the weakness and strength of competitive analysis. It is a strength because the setting is clear, the problems are crisp and sometimes deep, and the results are often elegant and striking. But it is a weakness for several reasons. First, in the face of the devastating comparison against an all-powerful offline algorithm, a wide range of online algorithms (good, bad, and mediocre) fare equally badly; the competitive ratio is thus not very informative and fails to discriminate or to suggest good approaches. Another aspect of the same problem is that, since a worst-case input decides the performance of the algorithm, the optimal algorithms are often unnatural and impractical and the bounds too pessimistic to be informative

*Received by the editors February 23, 1996; accepted for publication (in revised form) July 20, 1999; published electronically May 15, 2000.
http://www.siam.org/journals/sicomp/30-1/29954.html
†Computer Science Department, University of California, Los Angeles, CA 90095 (elias@cs.ucla.edu). This research was supported in part by NSF grant CCR-9521606.
‡Computer Science Division, University of California, Berkeley, CA 94720 (christos@cs.berkeley.edu). This research was supported in part by the National Science Foundation.

in practice. Even enhancing the capabilities of the online algorithm in obviously desirable ways (such as a limited lookahead capability) brings no improvement to the ratio (this is discussed more extensively later). The main argument for competitive analysis over the classical approach (minimize the expectation) is that *the distribution is usually not known.* However, competitive analysis takes this argument way too far: It assumes that *absolutely nothing* is known about the distribution, that *any distribution* of the inputs is in principle possible; the worst-case "distribution" prevailing in competitive analysis is, of course, a worst-case input with probability one. Such complete powerlessness seems unrealistic to both the practitioner (we always know, or can learn, *something* about the distribution of the inputs) and the theoretician of another persuasion (the absence of a prior distribution, or some information about it, seems very unrealistic to a probabilist or mathematical economist).

The *paging problem,* perhaps the most simple, fundamental, and practically important online problem, is a good illustration of all these points. An unreasonably wide range of deterministic algorithms (both the good-in-practice LRU and the empirically mediocre FIFO) have the same competitive ratio—k, the amount of available memory. Even algorithms within more powerful *information regimes*—for example, any algorithm with lookahead $\ell > 0$ pages—provably can fare no better. Admittedly, there have been several interesting variants of the framework that were at least partially successful in addressing some of these concerns. *Randomized* paging algorithms have more realistic performance [5, 11, 13]. Some alternative approaches to evaluating online algorithms were proposed in [1, 12] for the general case and in [2, 6, 7, 15] specifically for the paging problem.

In this paper we propose and study two refinements of competitive analysis which seem to go a long way toward addressing the concerns expressed above. Perhaps more importantly, we show that these ideas give rise to interesting algorithmic and analytical problems (which we have only begun to solve in this paper).

Our first refinement, *the diffuse adversary model*, removes the assumption that we know nothing about the distribution—without resorting to the equally unrealistic classical assumption that we know all about it. *We assume that the actual distribution D of the inputs is a member of a known class Δ of possible distributions.* That is, we seek to determine, for a given class of distributions Δ, the performance ratio

$$R(\Delta) = \min_A \max_{D \in \Delta} \frac{\mathcal{E}_D(A(x))}{\mathcal{E}_D(\mathrm{opt}(x))}. \tag{2}$$

That is, the adversary picks a distribution D among those in Δ, so that the expected, under D, performance of the algorithm and the offline optimum algorithm are as far apart as possible. Notice that, if Δ is the class of all possible distributions, (1) and (2) coincide since the worst possible distribution is the one that assigns probability one to the worst-case input and probability zero everywhere else. Hence the diffuse adversary model is indeed a refinement of competitive analysis.

In the paging problem, for example, the input distribution specifies, for each page a and sequence of page requests ρ, $\mathrm{Prob}(a|\rho)$—the probability that the next page fault is a, given that the sequence so far is ρ. It is unlikely that an operating system knows this distribution precisely. On the other hand, it seems unrealistic to assume that any distribution at all is possible. For example, suppose that the next page request *is not predictable with absolute certainty:* $\mathrm{Prob}(a|\rho) \leq \epsilon$, for all a and ρ, where ϵ is a real number between 0 and 1 capturing the inherent uncertainty of the request sequence. This is a simple, natural, and quite well-motivated assumption; call

the class of distributions obeying this inequality Δ_ϵ. An immediate question is, What online algorithm achieves the optimal competitive ratio $R(\Delta_\epsilon)$?

As it turns out, the answer is quite interesting. The optimum online algorithm is *robust*—that is, the same for all ϵ's—and turns out to be a familiar algorithm that is also very good in practice: LRU. It is noteworthy that LRU emerges from the analysis as the unique "natural" optimal algorithm, although there are other algorithms that may also be optimal. An important byproduct of our analysis is that, extending the work in [9], we completely characterize the work functions of the paging special case of the k-server problem.

In a preliminary version of this work that appeared in [8], we incorrectly stated that the competitive ratio $R(\Delta_\epsilon)$ is given by a simple Markov chain of $(k+\frac{1}{\epsilon})^{k-1}$ states. In fact, the answer is more complicated. The competitive ratio is given by the "optimal" Markov chain from a family of $k^{(k+\frac{1}{\epsilon})^{k-1}}$ Markov chains of $(k+\frac{1}{\epsilon})^{k-1}$ states each.

The second refinement of competitive analysis that we are proposing deals with the following line of criticism: In traditional competitive analysis, the all-powerful adversary frustrates not only interesting algorithms, but also powerful *information regimes.* The classical example is again from paging: In paging, the best competitive ratio of any online algorithm is k. But what if we have an online algorithm *with a lookahead of ℓ steps*, that is, an algorithm that knows the immediate future? It is easy to see that any such algorithm must fare equally badly as algorithms without lookahead. In proof, consider a worst-case request sequence, $abdc\ldots$, and take its $(\ell+1)$*-stuttered version*, $a^{\ell+1}b^{\ell+1}d^{\ell+1}c^{\ell+1}\ldots$. It is easy to see that an algorithm with lookahead ℓ is as powerless in the face of such a sequence as one without a lookahead. Once more, the absolute power of the adversary blurs practically important distinctions. Still, lookahead is obviously a valuable feature of paging algorithms. How can we use competitive analysis to evaluate its power? Notice that this is not a question about the effectiveness of a single algorithm, but about *classes of algorithms*, about the power of *information regimes*—ultimately, about the value of information.

To formulate and answer this and similar questions we introduce our second refinement of competitive analysis, which we call *comparative analysis.* Suppose that $\mathcal{A}$ and $\mathcal{B}$ are classes of algorithms—typically but not necessarily $\mathcal{A}\subseteq\mathcal{B}$; that is, $\mathcal{B}$ is usually a broader class of algorithms, a more powerful information regime. The *comparative ratio* $R(\mathcal{A},\mathcal{B})$ is defined as follows:

$$R(\mathcal{A},\mathcal{B})=\max_{B\in\mathcal{B}}\min_{A\in\mathcal{A}}\max_{x}\frac{A(x)}{B(x)}. \tag{3}$$

This definition is best understood in terms of a game-theoretic interpretation: $\mathcal{B}$ wants to demonstrate to $\mathcal{A}$ that it is a more powerful class of algorithms. To this end, $\mathcal{B}$ proposes an algorithm B among its own. In response, $\mathcal{A}$ comes up with an algorithm A. Then $\mathcal{B}$ chooses an input x. Finally, $\mathcal{A}$ pays $\mathcal{B}$ the ratio $A(x)/B(x)$. The larger this ratio, the more powerful $\mathcal{B}$ is in comparison to $\mathcal{A}$. Notice that if we let $\mathcal{A}$ be the class of online algorithms and $\mathcal{B}$ the class of all algorithms—online or offline—then (1) and (3) coincide, and $R(\mathcal{A},\mathcal{B})=R$. Hence comparative analysis is indeed a refinement of competitive analysis.

We illustrate the use of comparative analysis by attacking the question of the power of lookahead in online problems of the "server" type: If $\mathcal{L}_\ell$ is the class of all algorithms with lookahead ℓ, and $\mathcal{L}_0$ is the class of online algorithms, then we show

that, in the very general context of *metrical task systems* [3], we have

$$R(\mathcal{L}_0, \mathcal{L}_\ell) = 2\ell + 1,$$

(that is, the ratio is at most $2\ell + 1$ for all metrical task systems, and it is exactly $2\ell + 1$ for some), while in the more restricted context of paging,

$$R(\mathcal{L}_0, \mathcal{L}_\ell) = \min\{\ell + 1, k\}.$$

2. Diffuse adversaries. The competitive ratio for a diffuse adversary[1] is given in (2). In order to make the analysis independent of the initial conditions, we shall allow an additive constant in the numerator. More precisely, a deterministic online algorithm A is c-competitive against a class Δ of input distributions if there exists a constant d such that for all distributions $D \in \Delta$,

$$\mathcal{E}_D(A(x)) \leq c \cdot \mathcal{E}_D(\mathrm{opt}(x)) + d. \tag{4}$$

The competitive ratio of the algorithm A is the infimum of all such c's. Finally, the competitive ratio $R(\Delta)$ of the class of distributions is the minimum competitive ratio achievable by an online algorithm. It is important to observe that Δ is a class of acceptable conditional probability distributions; each $D \in \Delta$ is the distribution of the relevant part of the world conditioned on the currently available information. One can easily extend the definition to randomized online algorithms. However, in this work, we deal only with deterministic online algorithms.

In the case of the paging problem with a set of pages M, Δ is a set of probability distributions on page sequences M^*. An equivalent and perhaps more natural way to describe Δ is by a set of conditional probability distributions, that is, functions of the form $D : M^* \times M \mapsto [0, 1]$, where for all $\rho \in M^*$ $\sum_{a \in M} D(a|\rho) \leq 1$; the sum may be less than 1 because the adversary may choose to end the sequence. In the game-theoretic interpretation, as the sequence of requests ρ develops, the adversary chooses the values of $D(a|\rho)$ from those available in Δ to maximize the ratio. Since we deal with deterministic algorithms, the adversary knows precisely the past decisions of A, but the adversary's choices may be severely constrained by Δ. It is indicative of the power of the diffuse adversary model that most of the proposals for a more realistic competitive analysis are simply special cases of it. For example, the locality of reference in the paging problem [2, 6] is captured by the diffuse adversary model where Δ consists of the following conditional probability distributions: $D(a|\rho b) = 0$ if there is *no* edge from b to a in the *access graph* and $D(a|\rho b) = 0$ or 1 otherwise. Similarly, the Markov paging model [7] and the statistical adversary model [12] are also special cases of the diffuse adversary model. In the first case, the class Δ of distributions contains only one conditional distribution D with $D(\cdot|\rho b) = D(\cdot|b)$, and in the latter case $D \in \Delta$ assigns probability one to some request sequence that satisfies the statistical adversary restriction.

In this section we apply the diffuse adversary model to the paging problem. We shall focus on the class of distributions Δ_ϵ, which contains all functions $D : M^* \times M \mapsto [0, \epsilon]$—that is to say, all conditional distributions with no value exceeding ϵ. But first, we diverge in order to give a useful characterization of work functions for the paging problem.

[1] Diffuse adversaries are not related to the diffusion processes in probability theory which are continuous path Markov processes.

2.1. The structure of paging work functions. Since the paging problem is the k-server problem on uniform metric spaces, certain key concepts from the k-server theory will be very useful (see [9, 4] for a more detailed exposition).

DEFINITION 2.1. *Let k denote the number of page slots in fast memory, and let M be the set of pages. A* configuration *is a k-subset of M; we denote the set of all configurations by C. The* work function *associated with a sequence $\rho \in M^*$, w_ρ (or simply w when ρ is not important or understood from context) is a function $w_\rho : C \mapsto \Re^+$, defined as follows: $w_\rho(X)$ is the optimum number of page replacements when the sequence of requests is ρ and the final memory configuration is X.*

Notice that in some cases, for example, when a configuration X does not contain the last request, some page replacements may occur not because of page faults but because we insist that the final configuration is X.

Henceforth we use the symbols $+$ and $-$ to denote set union and set difference, respectively. Also, we represent unary sets with their element, e.g., we write a instead of $\{a\}$. Finally, we denote the cardinality of a set S by $|S|$.

DEFINITION 2.2. *If w is a work function, define the* support *of w to be all configurations $X \in C$ such that there is no $Y \in C$, different from X, with $w(X) = w(Y) + |X - Y|$.*

Intuitively, the values of w on its support completely determine w: if S is the support, then $w(X) = \min_{Y \in S}\{w(Y) + |X - Y|\}$. Furthermore, we can safely assume that the optimal offline algorithm (or adversary) is in a configuration of the support. To see this consider two configurations X and Y with $w_\rho(X) = w_\rho(Y) + |X - Y|$. Then the optimal offline algorithm has no advantage being in configuration X instead of configuration Y after servicing ρ; it can move from Y to X without exceeding the cost of being at X. In other words, an optimal offline algorithm that replaces a page only when it is necessary is always in a configuration of the support.

The following lemmas, specific to the paging problem and not true in general for the k-server problem, characterize all possible work functions by determining the structure of their support. A similar, but more complicated, characterization is implicit in the work of [11]. The first lemma states that all configurations in the support have the same value, and hence what matters is the support itself, not the values of w on it.

LEMMA 2.3. *The support of a work function w contains only configurations on which w achieves its minimum value.*

Proof. Toward a contradiction assume that there is a configuration A in the support of w such that $w(A) > \min_X w(X)$. Choose now a configuration B with $w(A) > w(B)$ that minimizes $|B - A|$. By the quasi-convexity condition in [9], there are $a \in A - B$ and $b \in B - A$ such that

$$w(A) + w(B) \geq w(A - a + b) + w(B - b + a).$$

Since $w(A) > w(B)$, this holds only if either $w(A) > w(A - a + b)$ or $w(A) > w(B - b + a)$. In the first case, we get that $w(A) = w(A - a + b) + 1$ and this contradicts the assumption that A is in the support of w. The second case also leads to a contradiction since it violates the choice of B with minimum $|B - A|$, because $|(B - b + a) - A| = |B - A| - 1$. □

Since the configuration of the optimal offline algorithm is always in the support, Lemma 2.3 shows that the offline cost to service a request is simply the increase (0 or 1) of the minimum value of a work function. As a result, it can be determined online when the adversary has a page fault.

The next lemma determines the structure of the support.

LEMMA 2.4. *For each work function w there is an increasing sequence of sets $S_1 \subsetneq S_2 \subsetneq \cdots \subsetneq S_k$, with $S_1 = \{r\}$ the most recent request, such that the support of w is precisely*

$$\{X : |X \cap S_j| \geq j \text{ for all } j\}.$$

We postpone the proof of the lemma to introduce some new definitions. We will call a k-tuple $S = (S_1, S_2, \ldots, S_k)$ of the lemma a *signature* of w. A signature of w completely determines its support. But it doesn't have to be unique. For example, a work function whose support consists of only one configuration $\{a_1, \ldots, a_k\}$ has $k!$ signatures (for each permutation σ let $S_j = \{a_{\sigma_1}, \ldots, a_{\sigma_j}\}$). We define the *type* of w to be the k-tuple $\mathbb{P} = (|S_1| = 1, |S_2|, \ldots, |S_k|)$. Although a work function may have many signatures, its type is unique; it is easy to see that a work function has more than one signature only when there is a $j > 1$ such that $|S_i| = i$ for $1 \leq i \leq j$.

We now prove Lemma 2.4.

Proof of Lemma 2.4. The proof is by induction on the length of the request sequence. The basis case is obvious: Let $S_j = \{a_1, \ldots, a_j\}$, where $\{a_1, \ldots, a_k\}$ is the initial configuration. For the induction step assume that we have a signature $(S_1, S_2, \ldots, S_k)$ of w, and let w' be the resulting work function after request r.

Consider first the case that r belongs to S_t and not to S_{t-1} for some $t \in \{2, 3, \ldots, k\}$. Since there is at least one configuration in the support of w that contains the request r, the minimum value of w' is the same as the minimum value of w. Therefore, the support of w' is a subset of the support of w. It consists of the configurations that belong to the support of w and contain the most recent request r. It is easy now to verify that w' has the following signature:

$$\begin{aligned} S'_1 &= \{r\}; \\ S'_i &= S_{i-1} + r, \qquad 2 \leq i \leq t; \\ S'_i &= S_i, \qquad t < i \leq k. \end{aligned}$$

If, on the other hand, the request r does not belong to S_k, the minimum value of w' is one more than the minimum value of w. In this case, the support of w' consists of all configurations in the support of w where one point has been replaced by the request r, i.e., a server has been moved to service r. Consequently, a signature of w' is given by

$$\begin{aligned} S'_1 &= \{r\}; \\ S'_i &= S_i + r, \qquad 1 < i \leq k. \end{aligned}$$

The induction step now follows. □

Note that the converse of the lemma, not needed in what follows, also holds: Any such tower of S_j's defines a reachable work function, that is, there exists w_ρ for some ρ with signature S.

We define the *canonical ordering* to be a permutation of all pages such that page a precedes page b when a has been requested more recently than b. Furthermore, pages that have not been requested are ordered according to a fixed ordering (for example, the lexicographic ordering). For example, if the canonical ordering after request sequence ρ is $a_1 \ldots a_m$, then ρ has the form $\ldots a_4(a_3|a_2|a_1)^* a_3(a_2|a_1)^* a_2(a_1)^* a_1$. An obvious property of the canonical ordering is that a new request a_j doesn't change the ordering except from the movement of a_j to the front.

It follows from the proof of Lemma 2.4 that if we know the canonical ordering $a_1 a_2 \dots$ and the type $(p_1, \dots, p_k)$ of a work function w, we can determine a signature $(S_1, \dots, S_k)$ of the work function: S_i consists of the first p_i pages of the canonical ordering, i.e., $S_i = \{a_1, \dots, a_{p_i}\}$. We will call this signature the *canonical signature* of w.

For a type $\mathbb{P} = (p_1 = 1, p_2, \dots, p_k)$ that corresponds to a canonical signature $S = (S_1, S_2, \dots, S_k)$, we will denote by $\mathbb{P}_j$ the type that results from a request in $S_{j+1} - S_j$. From the proof of Lemma 2.4, we get

$$\begin{aligned}
&\mathbb{P}_0 = \mathbb{P},\\
&\mathbb{P}_1 = (1, p_1 + 1, p_3, p_4, \dots, p_k),\\
&\mathbb{P}_2 = (1, p_1 + 1, p_2 + 1, p_4, \dots, p_k),\\
&\quad\vdots\\
&\mathbb{P}_{k-1} = (1, p_1 + 1, p_2 + 1, p_3 + 1, \dots, p_{k-1} + 1), \text{ and}\\
&\mathbb{P}_k = (1, p_2 + 1, p_3 + 1, \dots, p_k + 1).
\end{aligned}$$

The first type $\mathbb{P}_0$ corresponds to a request in S_1 and the last type to a request not in S_k. We will also use the following notation: For types $\mathbb{P} = (p_1, p_2, \dots, p_k)$ and $\mathbb{Q} = (q_1, q_2, \dots, q_k)$ we write $\mathbb{P} \le \mathbb{Q}$ (or $\mathbb{Q} \ge \mathbb{P}$) if for all $j = 1, 2, \dots, k$: $p_j \le q_j$. We also write $\mathbb{P} \trianglelefteq \mathbb{Q}$ (or $\mathbb{Q} \trianglerighteq \mathbb{P}$) if for all $j = 1, 2, \dots, k-1$: $p_j \le q_{j+1}$. The motivation for this notation is the relation between $\mathbb{P}_{k-1}$ and $\mathbb{P}_k$: $\mathbb{P}_{k-1} \trianglerighteq \mathbb{P}_k$. In addition, $\mathbb{P}_0 \ge \mathbb{P}_1 \ge \mathbb{P}_2 \ge \cdots \ge \mathbb{P}_{k-1} \trianglerighteq \mathbb{P}_k$.

We can now state the following crucial property of work functions.

LEMMA 2.5. *Let ρ and ρ' be two request sequences that result in the same canonical ordering. Let $\mathbb{P}$ and $\mathbb{Q}$ be the types of the two work functions $w = w_\rho$ and $w' = w_{\rho'}$, respectively. If $\mathbb{P} \le \mathbb{Q}$, then any configuration in the support of w is also in the support of w'. Furthermore, if $\mathbb{P} \trianglelefteq \mathbb{Q}$, then for any configuration in the support of w there is configuration in the support of w' that differs in at most one position.*

Proof. Let $S = (S_1, S_2, \dots, S_k)$ and $S' = (S'_1, S'_2, \dots, S'_k)$ be the canonical signatures of w and w'. The first case of the lemma, when $\mathbb{P} \le \mathbb{Q}$, is trivial since $S_j \subset S'_j$, $j = 1, 2, \dots, k$. For the second case, when $\mathbb{P} \trianglelefteq \mathbb{Q}$, it suffices to handle the "largest" possible $\mathbb{P}$, so we may assume that $S_{j-1} = S'_j$, for $j = 3, \dots, k-1$. Consider now a configuration X in the support of w. We will show that there is a configuration Y in the support of w' such that $|Y - X| \le 1$. Let x_k be the last page of X in the canonical ordering. Also, let b be the first page of the canonical ordering not in $X - x_k$. We claim that $Y = X - x_k + b$, which differs in at most one position from X, is in the support of w'. Notice first that Y contains the page in S_1. It also contains the second page of the canonical ordering because either this page is in $X - x_k$ or it is equal to b. It remains to show that $|Y \cap S'_j| \ge j$, for $j = 3, \dots, k$. There are two cases to consider: The first case, when $|X \cap S_{j-1}| \ge j$, follows from the fact that x_k is in S'_j $(= S_{j-1})$ only if b is in S'_j. For the second case, when $|X \cap S_{j-1}| = j - 1$, it suffices to note that $b \in S'_j$ but $x_k \notin S'_j$. □

2.2. Optimality of LRU. We now turn to the Δ_ϵ diffuse adversary. Our goal in this section is to show that LRU has optimal competitive ratio against a diffuse adversary. The usual approach to show that an online algorithm has optimal competitive ratio is to compute its ratio and then show that every other algorithm has no smaller competitive ratio. The difficulty here is that we cannot compute the competitive ratio of LRU. Thus we have to compare LRU directly with any other online algorithm. More precisely, we have to show that for any online algorithm A and any adversary

(conditional probability distribution) $D \in \Delta_\epsilon$, there is an adversary $D' \in \Delta_\epsilon$ such that the competitive ratio of A against D' is at least the competitive ratio of LRU against D. We have to face the problem that D may be very complicated and A may be erratic.

We attack the problem in two steps. First, we "standardize" the class of adversaries, by showing that it suffices to consider only a class of adversaries which we call *conservative* (to be defined shortly). In particular, we will show that for any $D \in \Delta_\epsilon$ there is a better conservative adversary $\hat{D} \in \Delta_\epsilon$ against LRU, i.e., LRU has a competitive ratio against $\hat{D}$ that is no less than that against D. Second, we show that for any conservative $D \in \Delta_\epsilon$ against LRU, there is a conservative adversary $D' \in \Delta_\epsilon$ against A, such that the competitive ratio of LRU against D is at most the competitive ratio of A against D'.

To understand why LRU is optimal against diffuse adversaries and to motivate the notion of conservatives adversaries, we start by asking what property an optimal online algorithm must have. Intuitively, if $(S_1, \ldots, S_k)$ is a signature of the current work function w, an optimal online algorithm should prefer to have in its fast memory pages from S_i instead of pages not in S_i. The intuition is that pages in S_i are more valuable to the offline algorithm than pages not in S_i, because a configuration from the support of w remains in the support when we replace any page not in S_i with a page in S_i; the converse does not hold in general. LRU does exactly this. In fact, LRU keeps in its fast memory the first k pages of the canonical ordering (with appropriate initialization).

The same intuition suggests that the adversary should prefer as next request a page $a \in S_i$ to a page $b \notin S_i$; the only exception being when a is in the online cache and b is not. In this case, it is unclear which page (a or b) the adversary should choose because of the following trade-off: On the one hand, b is more favorable because it increases the online cost, whereas a does not. On the other hand, a is more favorable since either the resulting support is larger or b increases the offline cost. The above intuition suggests the notion of conservative adversary. A conservative adversary assigns probability that favors pages with smaller rank in the canonical ordering among the pages in the online cache and similarly for the pages not in the online cache. To be more precise, fix a request sequence ρ that results in canonical ordering $a_1 a_2 \ldots$ and configuration C of an online algorithm A. A conservative adversary against A assigns probabilities with the property that for every $a_i, a_j \in C$ with $i < j$, a_j receives positive probability, $D(a_j|\rho) > 0$, only if a_i receives maximum probability, $D(a_i|\rho) = \epsilon$. Similarly for pages $a_i, a_j \notin C$. Thus the probabilities $D(\cdot|\rho)$ are completely determined by the total probability z assigned to pages in C. We add an additional constraint, although the proofs do not call for it: At most one page receives probability that is not zero or ϵ (equivalently either z or $1 - z$ is an integral multiple of ϵ). For example, if $\epsilon = 1/5$, the online cache C consists of pages $1, 3, 4, 5$ of the canonical ordering, and $z = 2/5$, then pages $1, 2, 3, 6, 7$ receive probability ϵ.

It seems reasonable that an optimal adversary against LRU is conservative and indeed we are going to prove shortly that this is the case. For other online algorithms, however, conservative adversaries may not be optimal, especially for "unreasonable" and highly suboptimal online algorithms. A central idea of our proof is to disregard this problem. Even if conservative adversaries are not optimal against an online algorithm A, it suffices to show that there is a conservative adversary that forces a competitive ratio no less than the ratio of LRU. We now show that an optimal adversary against LRU is conservative.

LEMMA 2.6. *For any $D \in \Delta_\epsilon$, there is a conservative adversary $\hat{D} \in \Delta_\epsilon$ such that the competitive ratio of LRU against $\hat{D}$ is at least the competitive ratio of LRU against D.*

Proof. The proof is by induction on the number of requests. To facilitate induction, we use a strong inductive hypothesis. First, we allow any initial work function (instead of the work function whose support contains only one configuration). Let $\mathrm{opt}(\rho, \mathbb{P}, C)$ be the optimal cost to service ρ when the initial work function has type $\mathbb{P}$ and canonical ordering C. For simplicity, in our notation we use only the type of work functions and we write $\mathrm{opt}(\rho, \mathbb{P})$ instead of $\mathrm{opt}(\rho, \mathbb{P}, C)$. Notice also that the initial type does not affect the behavior of LRU at all, so we can simply write $\mathrm{LRU}(\rho)$ to denote the cost of *LRU* for the sequence request ρ.

Second, since we don't know the competitive ratio of LRU, we simply use the fact that the competitive ratio is positive (although we could safely assume a competitive ratio at least 1). More precisely, the inductive hypothesis is that for any $D \in \Delta_\epsilon$, any n, and any $c > 0$ there is a conservative $\hat{D}$ such that $\mathcal{E}_D(LRU(x_n) - c \cdot \mathrm{opt}(x_n, \mathbb{P})) \leq \mathcal{E}_{\hat{D}}(LRU(x_n) - c \cdot \mathrm{opt}(x_n, \mathbb{P}))$, where $\mathbb{P}$ is the type of the initial work function and x_n denotes a sequence of length n drawn from the conditional probability distribution D or $\hat{D}$. Equivalently, we will show that there is a conservative D that maximizes

$$\psi(D, n, \mathbb{P}) = \mathcal{E}_D(LRU(x_n) - c \cdot \mathrm{opt}(x_n, \mathbb{P})). \tag{5}$$

We use induction on the number of requests n. For $n = 0$, there is nothing to prove. Assume now that the induction hypothesis holds for $n-1$ and let D be a distribution that maximizes $\mathcal{E}_D(LRU(x_n) - c \cdot \mathrm{opt}(x_n, \mathbb{P}))$. If D is not conservative, we will show how to alter it to get a conservative adversary $\hat{D}$ that also maximizes $\psi(\hat{D}, n, \mathbb{P})$.

The proof proceeds as follows: Although D may not be conservative, by induction it becomes conservative after the first request. So, we want to show that the first request is also "conservative." Denote by D_j the resulting conditional distribution when the first request is the jth request a_j of the canonical ordering, i.e., $D_j(\cdot|\rho) = D(\cdot|a_j\rho)$. Let also $\mathbb{P}^{(j)}$ denote the type of the work function that results after request a_j. If $(S_1, \ldots, S_k)$ is the initial canonical signature, then

$$\psi(D, n, \mathbb{P}) = \sum_{j \geq 1} D(a_j) \cdot \left[\psi(D_j, n-1, \mathbb{P}^{(j)}) + I(j > k) - c \cdot I(a_j \notin S_k) \right], \tag{6}$$

where $D(a_j) = D(a_j|\varepsilon)$ is the probability assigned to page a_j conditioned on the empty sequence ε, and $I(\phi)$ is the indicator function that takes value 1 when ϕ is true and 0 otherwise. The last two terms inside the sum of the right-hand side correspond to the online and the offline cost for the first request. By induction, we can replace D_j with a conservative $\hat{D}_j$ without decreasing the right-hand side of (6). So, without loss of generality, we assume that the distributions D_j are conservative.

If we fix the conditional distributions D_j, the optimal probabilities for the first request can be computed as follows: order the pages in decreasing $\psi(D_j, n-1, \mathbb{P}^{(j)}) + I(j > k) - c \cdot I(a_j \notin S_k)$ value, assign probability ϵ to the first $\lfloor 1/\epsilon \rfloor$ pages, and assign the remaining probability (which of course is less than ϵ) to the next page.

We simply want to guarantee that the probabilities are assigned in a conservative manner. Equivalently, if $i < j \leq k$ (both a_i and a_j are in LRU's cache) or when

$k < i < j$ (neither a_i nor a_j are in LRU's cache), it suffices to show that

$$\psi(D_i, n-1, \mathbb{P}^{(i)}) + I(i > k) - c \cdot I(a_i \notin S_k) \geq \psi(D_j, n-1, \mathbb{P}^{(j)}) + I(j > k) - c \cdot I(a_j \notin S_k). \tag{7}$$

We first consider the case of $i < j \leq k$. The crucial point for establishing (7) is that we can assume

$$\psi(D_i, n-1, \mathbb{P}^{(i)}) \geq \psi(D_j, n-1, \mathbb{P}^{(j)}). \tag{8}$$

To see this, let us consider the adversary $\hat{D}$ that is identical to D but when the first request is a_i, it continues like D_j instead of D_i (taking into account the difference in the canonical orderings). Formally, if $b_1 b_2 \ldots$ is the canonical ordering that results when the first request is a_j and $b'_1 b'_2 \ldots$ when the first request is a_i, then $\hat{D}(b'_l | a_i b'_{l_1} \ldots b'_{l_m}) = D(b_l | a_j b_{l_1} \ldots b_{l_m})$. We then have $\psi(\hat{D}, n, \mathbb{P}) = \psi(D, n, \mathbb{P}) + D(a_i)(\psi(D_j, n-1, \mathbb{P}^{(i)}) - \psi(D_i, n-1, \mathbb{P}^{(i)}))$.

But since $\mathbb{P}^{(i)} \geq \mathbb{P}^{(j)}$, we can apply Lemma 2.5 and get $\mathrm{opt}(x_{n-1}, \mathbb{P}^{(i)}) \leq \mathrm{opt}(x_{n-1}, \mathbb{P}^{(j)})$, which in turn implies that $\psi(D_j, n-1, \mathbb{P}^{(i)}) \geq \psi(D_j, n-1, \mathbb{P}^{(j)})$. Thus $\psi(\hat{D}, n, \mathbb{P}) \geq \psi(D, n, \mathbb{P}) + D(a_i)(\psi(D_j, n-1, \mathbb{P}^{(j)}) - \psi(D_i, n-1, \mathbb{P}^{(i)}))$. If (8) does not hold, then $\psi(\hat{D}, n, \mathbb{P}) \geq \psi(D, n, \mathbb{P})$; hence, $\hat{D}$ maximizes ψ and of course has the desired property (8).

Combining (8) and the fact that $I(i > k) = I(j > k) = I(a_i \notin S_k) = I(a_j \notin S_k) = 0$, we get that (7) holds for $i < j \leq k$.

Inequality (7) holds for $k < i < j$. An identical argument as above establishes it for the case of $a_i, a_j \in S_k$ or $a_i, a_j \notin S_k$. The remaining case, when $a_i \in S_k$ and $a_j \notin S_k$, is treated similarly. In this case, we have that $\mathbb{P}^{(i)} \trianglerighteq \mathbb{P}^{(j)}$, which implies $\mathrm{opt}(x_{n-1}, \mathbb{P}^{(i)}) \leq \mathrm{opt}(x_{n-1}, \mathbb{P}^{(j)}) + 1$. The last inequality implies $\psi(D_i, n-1, \mathbb{P}^{(i)}) \geq \psi(D_j, n-1, \mathbb{P}^{(j)}) - c$, and together with the facts that $I(i > k) = I(j > k) = 1$, $I(a_i \notin S_k) = 0$, and $I(a_j \notin S_k) = 1$, we get (7). □

The above lemma establishes that the best adversary against LRU is conservative. We can now proceed to the second part of the proof that LRU has optimal competitive ratio. We will show that for any online algorithm A and any conservative adversary $D \in \Delta_\epsilon$ against LRU, there is an adversary $D' \in \Delta_\epsilon$ against A such that the competitive ratio of LRU against D is at most the competitive ratio of A against D'. The main idea for constructing a D' is by enforcing the online cost of LRU against D to be equal to the online cost of A against D'. This allows us to compare the competitive ratios of LRU and A, simply by comparing the corresponding offline costs.

Given D, how can we design D' so that cost of LRU against D is equal to the cost of A against D'? For the first request it is obvious: the probability that D' assigns to pages in the cache of A should be equal to the probability that D assigns to pages in LRU's cache. But for the second and subsequent requests, the situation depends on previous requests (the outcome of random experiments). In general, a conservative adversary against an algorithm B corresponds to an (infinite) rooted tree T with outdegree $\lceil 1/\epsilon \rceil$ such that each node v has weight $z(v)$ in $[0, \min\{k\epsilon, 1\}]$; furthermore, either $z(v)$ or $1 - z(v)$ is an integral multiple of ϵ. Paths on this tree correspond to request sequences and the values $z(v)$ are the total probability assigned by the adversary to pages in the online cache. More precisely, a request sequence is produced by starting at the root and descending down the tree. At a node v the adversary assigns probabilities in a conservative manner so that the total probability assigned to pages of the current cache of B is $z(v)$. Since the adversary is conservative, at most $l = \lceil 1/\epsilon \rceil$

pages receive nonzero probability. Let $r_1, \ldots, r_l$ be these pages (first the pages in the current cache of B and then the pages outside the cache in the canonical order). If the next request (the outcome of the random experiment) is r_j, the adversary moves to the jth child of v and repeats the process to produce the next request. We will call this conservative adversary the *adversary based on T against B* and we will denote it by $D_{T(B)}$. Notice that the conditional probability distribution $D_{T(B)}$ depends on the online algorithm B (for another algorithm B' it may be different). It also depends on the initial configuration of B, although for simplicity we omit this dependence in our notation.

It is now simple to design D' so that the expected cost of A against D' is equal to the expected cost of LRU against D: if T is the tree such that $D = D_{T(\mathrm{LRU})}$, then $D' = D_{T(A)}$. It suffices to show that the expected optimum cost against D is no less than the expected optimum cost against D'. More precisely, we will show that the expected offline cost to service a request sequence of length n produced by $D_{T(\mathrm{LRU})}$ is no less than the offline cost to service a request sequence of length n produced by $D_{T(A)}$.

We will use induction on the length n of the request sequence. In order to facilitate induction, we generalize the problem by assuming any initial work function. As in the proof of Lemma 2.6, our notation will include only the type of the work function. More precisely, let $h_{T(A)}(n, \mathbb{P})$ denote the expected offline cost to service a request sequence of length n produced by the conditional distribution $D_{T(A)}$ when the initial work function has type $\mathbb{P}$. We will show that for every $\mathbb{P}$: $h_{T(\mathrm{LRU})}(n, \mathbb{P}) \geq h_{T(A)}(n, \mathbb{P})$. But first, we need the following simple lemma.

LEMMA 2.7. *For all conservative adversaries T, if $\mathbb{P} \leq \mathbb{Q}$, then $h_{T(LRU)}(n, \mathbb{P}) \geq h_{T(LRU)}(n, \mathbb{Q})$, and if $\mathbb{P} \trianglelefteq \mathbb{Q}$, then $h_{T(LRU)}(n, \mathbb{P}) + 1 \geq h_{T(LRU)}(n, \mathbb{Q})$.*

Proof. The crucial observation is that LRU does not depend on the initial type of the work function, but only on the initial canonical ordering. An immediate consequence is that the conditional probability distribution $T(\mathrm{LRU})$ is independent of the initial work function. Consider first the case of $\mathbb{P} \leq \mathbb{Q}$. By Lemma 2.5, the offline cost to service a request sequence starting with a work function w of type $\mathbb{P}$ cannot be less than the offline cost to service the same request sequence starting with a work function w' of type $\mathbb{Q}$ and the same canonical ordering with w. Therefore, $h_{T(\mathrm{LRU})}(n, \mathbb{P}) \geq h_{T(\mathrm{LRU})}(n, \mathbb{Q})$. The other case, $\mathbb{P} \trianglelefteq \mathbb{Q}$, is handled similarly. □

We are now ready to show that the expected offline cost of a request sequence produced by a tree T is maximized when the online algorithm is the LRU algorithm.

LEMMA 2.8. *For every tree T and every online algorithm A, $h_{T(LRU)}(n, \mathbb{P}) \geq h_{T(A)}(n, \mathbb{P})$.*

Proof. By induction on n. For $n = 0$, the lemma is trivially true. Assume that the lemma holds for $n-1$. Denote by T_j the subtree rooted at the jth child of the root of T. Let r_A and r_{LRU} be the request that is produced when the adversary descends to T_j for algorithms A and LRU, respectively. Let also $\mathbb{P}_{i_A}$, $\mathbb{P}_{i_{\mathrm{LRU}}}$ denote the resulting type after requests r_A and r_{LRU}. It is important to notice that $i_A \leq i_{\mathrm{LRU}}$; this is the only property of algorithm A used in the proof.

Child j is chosen with probability that depends on the value $z(v)$, where v is the root of T. This probability is ϵ for all values of j except of one child when $1/\epsilon$ is not an integer. Obviously, $h_{T(A)}(n, \mathbb{P})$ is equal to the expected value of $h_{T_j(A)}(n-1, \mathbb{P}_{i_A})$ plus the offline cost for the first request; the offline cost is 0 when $i_A < k$ and it is 1 when $i_A = k$. In other words $h_{T(A)}(n, \mathbb{P}) = E[h_{T_j(A)}(n-1, \mathbb{P}_{i_A}) + I(i_A = k)]$. A similar expression holds also for $h_{T(LRU)}(n, \mathbb{P})$. It suffices therefore to show that for

all j: $h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) + I(i_A = k) \le h_{T_j(LRU)}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) + I(i_{\mathrm{LRU}} = k)$.

We consider three cases according to the values of i_A and i_{LRU}. In the first case, $i_A \le i_{\mathrm{LRU}} < k$, we get

$$\begin{aligned} h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) + I(i_A = k) &= h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) \\ &\le h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_A}) \\ &\le h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) \\ &= h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) + I(i_{\mathrm{LRU}} = k), \end{aligned}$$

where the first inequality follows from the induction hypothesis and the second one from Lemma 2.7, because $\mathbb{P}_{i_A} \ge \mathbb{P}_{i_{\mathrm{LRU}}}$. In the second case, $i_A < i_{\mathrm{LRU}} = k$, we get $h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) + I(i_A = k) = h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) \le h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_A}) \le h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) + 1 = h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) + I(i_{\mathrm{LRU}} = k)$. Again, the first inequality follows from the induction hypothesis and the second one from Lemma 2.7, because $\mathbb{P}_{i_A} \trianglerighteq \mathbb{P}_{i_{\mathrm{LRU}}}$. Finally, the third case, $i_A = i_{\mathrm{LRU}} = k$, is handled similarly: $h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) + I(i_A = k) = h_{T_j(A)}(n-1,\mathbb{P}_{i_A}) + 1 \le h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) + 1 = h_{T_j(\mathrm{LRU})}(n-1,\mathbb{P}_{i_{\mathrm{LRU}}}) + I(i_{\mathrm{LRU}} = k)$. □

The above lemmas establish the main result of this section.

THEOREM 2.9. *For any ϵ, LRU has optimal competitive ratio $R(\Delta_\epsilon)$ for the paging diffuse adversary model.*

The above lemmas, however, do not provide any efficient way to estimate the competitive ratio $R(\Delta_\epsilon)$. The approach suggested by the lemmas is to nondeterministically guess the optimal conservative adversary and then compute the competitive ratio of LRU against this adversary. A conservative adversary is determined by the values $z(v)$ that are multiples of ϵ when $1/\epsilon$ is an integer; it is slightly more complicated when $1/\epsilon$ is not an integer. For a given conservative adversary, the competitive ratio of LRU is given by a finite Markov chain; the states of the Markov chain are all reachable types and there are at most $(k+1/\epsilon)^{k-1}$ such types. This approach provides a doubly exponential algorithm (approximately $k^{(k+1/\epsilon)^{k-1}}$) to compute $R(\Delta_\epsilon)$. It is an interesting open problem to determine $R(\Delta_\epsilon)$ as a function of ϵ. For the extreme values of ϵ, we know that $R(\Delta_1) = k$ and $\lim_{\epsilon\to 0} R(\Delta_\epsilon) = 1$. In the first case, the adversary has complete power and in the second case, it suffers a page fault in almost every step. Recently, Young [16] estimated $R(\Delta_\epsilon)$ within (almost) a factor of two; $R(\Delta_\epsilon)$ is between $\Phi(\epsilon,k) - 1$ and $2\Phi(\epsilon,k)$, where $\Phi(\epsilon,k) = 1 + \sum_{i=1}^{k-1} 1/\max\{1/\epsilon - i, 1\}$ (this is approximately $\ln \frac{1}{1-(k-1)\epsilon}$ for $\epsilon > 1/k$, and $k + 1 - 1/\epsilon$ otherwise).

Even for the simplest case of $k = 2$, determining the competitive ratio is not trivial. Here we give a lower bound which we believe is exact. This is the competitive ratio against the conservative adversary that always assigns nonzero probability to exactly one page from the pages in the online fast memory.

PROPOSITION 2.10. *For $k = 2$, if $n = 1/\epsilon - 1$ is an integer, then*

$$R(\Delta_\epsilon) \ge \frac{\sum_{i=0}^{n} n^i/i!}{\sum_{i=0}^{n-1} n^i/i!} \in [1 + \sqrt{\epsilon}/2, 1 + 2\sqrt{\epsilon}].$$

Proof. Consider the conservative adversary that always assigns probability ϵ to pages of the current canonical ordering with rank $1, 3, 4, \ldots, 1/\epsilon + 1$. Notice that page 2 is assigned zero probability. The request on the page of rank 1 is identical to the previous request and does not change anything. Thus, this adversary is equivalent to the adversary that assigns probability $\delta = \epsilon/(1-\epsilon) = 1/n$ to pages with rank $3, 4, \ldots, 1/\epsilon + 1$.

Let $(1, p_2)$ be the type of the current work function. The following table summarizes the possibilities of the next request, together with the probability, the resulting type, and the associated online and offline cost for servicing the request.

Request	Probability	Type	Online cost	Offline cost
$3 \dots p_2$	$(p_2 - 2)\delta$	$(1, 2)$	1	0
$p_2 + 1 \dots 1/\delta + 2$	$1 - (p_2 - 2)\delta$	$(1, p_2 + 1)$	1	1

In summary, the competitive ratio is given by a Markov chain M_δ with states the types $(1, p_2)$ for $p_2 = 2, 3, \dots, 1/\delta + 2$. The transition probabilities from state $(1, p_2)$ are given in the above table. It is not difficult to see that the Markov process is identical to the following random process: In each phase repeatedly choose uniformly a number from $\{1, 2, \dots, n\}$, where $n = 1/\delta$; a phase ends when a number is chosen twice. The state $(1, p_2)$ of M_δ corresponds to the case that $p_2 - 2$ numbers have been drawn. This random process is a generalization of the well-known *birthday* problem in probability theory. A phase corresponds to a cycle in the Markov chain that starts (and ends) at state with type $(1, 2)$. The expected offline cost per phase is equal to the length of a phase minus one (all transitions in the cycle have offline cost one except the last one). Similarly, the expected online cost per phase is equal to the length of a phase (all transitions have online cost one). It is not hard now to verify the expression for $R(\Delta_\epsilon)$.

For the purpose of bounding the expected length of a phase, notice that each of the first $\sqrt{n}$ numbers has probability at most $1/\sqrt{n}$ to end the phase. In contrast, each of the next $\sqrt{n}$ numbers has probability at least $1/\sqrt{n}$ to end the phase. Elaborating on this observation we get that R_ϵ is in the interval $[1 + \sqrt{\epsilon}/2, 1 + 2\sqrt{\epsilon}]$.

Numerical evaluations suggest that the value of R_ϵ is approximately $1 + 0.8\sqrt{\epsilon}$, when $\epsilon \to 0$. □

A preliminary version of this work [8] had an incorrect proof of optimality of LRU. The proof was based on the unjustified assumption that a conservative adversary that achieves optimal competitive ratio against LRU assigns probability ϵ to exactly one page in the online fast memory. As was pointed out to us by Neal Young (see also [16]), this assumption does not hold in general.

An important open problem is to determine the competitive ratio of known paging algorithms against a diffuse adversary. The most important direction is to estimate the competitive ratio of FIFO. The recent work of Young [16] estimates the competitive ratio of marking algorithms—both LRU and FIFO are marking algorithms—almost within a factor of 2. He gives similar bounds for randomized algorithms. Our proof of the optimality of LRU seems to suggest that for certain values of ϵ, the competitive ratio of FIFO is not optimal. In particular, FIFO does not always keep in its fast memory the first k pages of the canonical ordering or an equivalent set of pages. This then is an indication that a conservative adversary may force FIFO to have larger cost than LRU. We conjecture that FIFO is suboptimal for some values of ϵ. If indeed this is the case, it will add some extra validity to the paging diffuse adversary model, in the sense that the model can actually distinguish between LRU and FIFO.

3. Comparative analysis. Online algorithms deal with the relations between *information regimes*. Formally but briefly, an information regime is the class of all functions from a domain D to a range R that are constant within a fixed partition of D. Refining this partition results in a richer regime. Traditionally, the literature on online algorithms has been preoccupied with comparisons between two basic information regimes: the *online* and the *offline* regime (the offline regime corresponds to the fully

refined partition). As we argued in the introduction, this has left unexplored several intricate comparisons between other important information regimes.

Comparative analysis is a generalization of competitive analysis allowing comparisons between arbitrary information regimes, via the comparative ratio defined in (3). Naturally, such comparisons make sense only if the corresponding regimes are rich in algorithms—single algorithms do not lend themselves to useful comparisons. As in the case of the competitive ratio for the diffuse adversary model, we usually allow an additive constant in the numerator of (3).

We apply comparative analysis in order to evaluate the power of *lookahead* in task systems. An online algorithm for a metrical task system has lookahead ℓ if it can base its decision not only on the past but also on the next ℓ requests. All online algorithms with lookahead ℓ comprise the information regime $\mathcal{L}_\ell$. Thus, $\mathcal{L}_0$ is the class of all traditional online algorithms.

Metrical task systems [3] are defined on some metric space $\mathcal{M}$; a server resides on some point of the metric space and can move from point to point. Its goal is to process online a sequence of tasks $T_1, T_2, \ldots$. The server is free to move to any position before processimg a task, although it has to pay the distance. The cost $c(T_j, a_j)$ for processing a task T_j is determined by the task T_j and the position a_j of the server while processing the task. The total cost for processing the sequence is the sum of the distance moved by the server plus the cost of servicing each task T_j, $j = 1, 2, \ldots$.

THEOREM 3.1. *For any metrical task system, $R(\mathcal{L}_0, \mathcal{L}_\ell) \leq 2\ell + 1$. Furthermore, there are metrical task systems for which $R(\mathcal{L}_0, \mathcal{L}_\ell) = 2\ell + 1$.*

Proof. Trivially the theorem holds for $\ell = 0$. Assume that $\ell > 0$ and fix an algorithm B in $\mathcal{L}_\ell$. We shall define an online algorithm A without lookahead whose cost on any sequence of tasks is at most $2\ell + 1$ times the cost of B. Algorithm A is a typical online algorithm in comparative analysis: it tries to efficiently "simulate" the more powerful algorithm B. In particular, A knows the position of B ℓ steps ago. In order to process the next task, A moves first to B's last known position, and then processes the task greedily, that is, with the minimum possible cost.

Let $T_1, T_2, \ldots$ be a sequence of tasks and let $b_1, b_2, \ldots$ be the points where algorithm B processes each task and $a_1, a_2, \ldots$ the corresponding points for algorithm A. For simplicity, we define also points $b_j = a_j = a_0$ for negative j's.

Then the cost of algorithm B is

$$\sum_{j \geq 1} \left(d(b_{j-1}, b_j) + c(T_j, b_j) \right)$$

and the cost of algorithm A is

$$\sum_{j \geq 1} \left(d(a_{j-1}, b_{j-\ell}) + d(b_{j-\ell}, a_j) + c(T_j, a_j) \right). \tag{9}$$

Recall that in order to process the jth task, algorithm A moves to B's last known position $b_{j-\ell}$ and then processes the task greedily, that is, $d(b_{j-\ell}, a_j) + c(T_j, a_j)$ is the minimum. In particular,

$$d(b_{j-\ell}, a_j) + c(T_j, a_j) \leq d(b_{j-\ell}, b_j) + c(T_j, b_j).$$

From this, the fact that costs are nonnegative, and the triangle inequality we get

$$\begin{aligned} d(a_{j-1}, b_{j-\ell}) &\leq d(a_{j-1}, b_{j-\ell-1}) + d(b_{j-\ell-1}, b_{j-\ell}) \\ &\leq d(b_{j-1}, b_{j-\ell-1}) + c(T_{j-1}, b_{j-1}) + d(b_{j-\ell-1}, b_{j-\ell}). \end{aligned}$$

We can now bound the cost of algorithm A in (9) by

$$\sum_{j\geq 1}\left(d(b_{j-1},b_{j-\ell-1})+c(T_{j-1},b_{j-1})+d(b_{j-\ell-1},b_{j-\ell})+d(b_{j-\ell},b_j)+c(T_j,b_j)\right). \tag{10}$$

Using the triangle inequalities of the form

$$d(b_i,b_{i+2})\leq d(b_i,b_{i+1})+d(b_{i+1},b_{i+2}),$$

we can expand $d(b_{j-\ell-1},b_{j-1})\leq d(b_{j-\ell-1},b_{j-\ell})+\cdots+d(b_{j-2},b_{j-1})$ and similarly we can expand $d(b_{j-\ell},b_j)$. Observe now that each term $d(b_{i-1},b_i)$ appears in (10) $2\ell+1$ times and each term $c(T_i,b_i)$ appears twice. We can therefore conclude that the cost of algorithm A is at most

$$\sum_{j\geq 1}\left((2\ell+1)d(b_{j-1},b_j)+2c(T_j,b_j)\right)\leq(2\ell+1)\sum_{j\geq 1}\left(d(b_{j-1},b_j)+c(T_j,b_j)\right).$$

The last expression is $(2\ell+1)$ times the cost of algorithm B.

To show the converse, we consider a task system with metric space $\mathcal{M}$ a (rooted) binary tree, where the distance between adjacent vertices is 1. Let B be the "greedy" algorithm with lookahead ℓ. In other words, B services the next task in such a way that minimizes the total cost to service the next ℓ tasks. Consider now an algorithm A with no lookahead. We will describe a sequence of tasks $T_1,T_2,\ldots$ that force a comparative ratio $2\ell+1$ for A against B. For this, let a_{j-1} be the position where A services the task T_{j-1}. With appropriate initialization, assume that a_{j-1} is at depth $j+\ell$. The next task T_j has infinite cost on all vertices except for the 2^ℓ vertices that are on depth $j+\ell+1$ and on distance $2\ell+1$ from the current position a_{j-1}. (To move to one of these vertices A must move up to level j and then down to level $j+\ell+1$.) The cost of T_j on these vertices is 0. Thus, the cost for A to service each such task is $2\ell+1$, while the cost for B is 1. (Using its lookahead power, it simply walks down the tree.)

We remark here that although the above lower bound uses an infinite metric space, a finite metric space that looks locally like a binary tree can also be used. In particular, consider a butterfly (the FFT graph) of $2\ell+2$ levels and identify the first and last levels. Then we can embed the infinite binary tree into this graph in such a way that every subtree of $\ell+1$ levels is embedded isometrically (in a distance-preserving manner). We conclude that there are task systems with metric spaces of $2^{O(\ell)}$ points and comparative ratio $2\ell+1$. We leave it as an interesting open problem to determine the comparative ratio for smaller metric spaces. □

Of course, for certain task systems the comparative ratio may be less than $2\ell+1$. For the paging problem, it is $\min\{\ell+1,k\}$.

THEOREM 3.2. *For the paging problem*

$$R(\mathcal{L}_0,\mathcal{L}_\ell)=\min\{\ell+1,k\}.$$

Proof. Let $n=\min\{\ell,k-1\}$ and let B be an algorithm for the paging problem in the class $\mathcal{L}_\ell$, that is, with lookahead ℓ. Without loss of generality we assume that B moves pages only to service requests. Consider the followingonline algorithm A which

is a generalization of LRU:

> To service a request r not in its fast memory, A evicts a page that is not one of the n most recent distinct requests (including r). Among the remaining pages, A chooses to evict a page such that the resulting configuration is as close as possible to the last known configuration of B. A does nothing for requests in its fast memory.

To show that the comparative ratio of A is $n+1$, it suffices to show that for every $n+1$ consecutive page faults of A, B suffers at least one page fault. This can be achieved by showing that after each subsequence of requests that causes n page faults to A and no page fault to B, both algorithms are in the same configuration. To do this, we show by induction on the number of requests the stronger claim that after a subsequence of requests that cause c page faults to A and no fault to B, the configurations of A and B differ by at most $n - c$ pages.

Fix a request sequence $\rho = r_1 r_2 \dots$ and let $A_0, A_1, \dots$ and $B_0 = A_0, B_1, \dots$ be the configurations of A and B that service ρ. The base of the induction is trivial. Assume that the induction hypothesis holds for $t-1$. We have to deal with a few cases. First of all, when $r_t \in A_{t-1}$, A suffers no page fault and the inductive step follows from the fact that $|A_t - B_t| \leq |A_{t-1} - B_{t-1}|$. Similarly, when the request $r_t \notin B_{t-1}$, B suffers a page fault and $|A_t - B_t| \leq |A_{t-1} - B_{t-1}|$, which is at most n by the induction hypothesis. Assume now that $r_t \in B_{t-1} - A_{t-1}$. Let x_t be the page evicted by A to service r_t. It is easy to see that if $x_t \notin B_{t-1}$, then $|A_t - B_t| = |A_{t-1} - B_{t-1}| - 1$. The final and more complicated case is when $x_t \in B_{t-1}$. By the definition of algorithm A, x_t is not one of the n most recent requests and consequently x_t is also in B_{t-n}. It follows that $A_{t-1} \subseteq B_{t-n} + \{r_{t-n+1}, \dots, r_{t-1}\}$ (otherwise A would not choose $x_t \in B_{t-n}$ to evict) and

$$A_t \subseteq B_{t-n} + \{r_{t-n+1}, \dots, r_{t-1}, r_t\}.$$

We also have the obvious relation

$$B_t \subseteq B_{t-n} + \{r_{t-n+1}, \dots, r_{t-1}, r_t\}.$$

(Recall that we assumed that B moves pages only to service requests.) If B suffered no page fault during the last c requests, i.e., $B_t = B_{t-1} = \cdots B_{t-c}$, the set $B_{t-n} + \{r_{t-n+1}, \dots, r_{t-1}, r_t\}$ has cardinality at most $k + n - c$. We conclude that $|A_t \cup B_t| \leq k + n - c$, which implies the desired $|A_t - B_t| \leq n - c$.

To show that $\min\{\ell + 1, k\}$ is a lower bound of the comparative ratio, let B be a variant of the optimal offline algorithm adapted to lookahead ℓ. More precisely, B never evicts one of the next n requests. Fix a set of $k+1$ pages. Clearly, for every request sequence ρ, B suffers at most one page fault for every $n+1$ consecutive requests. The lower bound follows because for any algorithm A, there is a request sequence ρ such that A suffers a page fault for every request. □

4. Open problems. We introduced two refinements of competitive analysis, the diffuse adversary model and comparative analysis. Both restrict the power of the adversary: the first by allowing only certain input distributions and the second by restricting the refinement of the adversary's information regime. In general, we believe that the only natural way to deal with uncertainty is by designing algorithms that perform well in the worst world which is compatible with the algorithm's knowledge.

There are a lot of interesting open problems suggested by this approach. The most important open problem is to determine the competitive ratio of FIFO for the paging

diffuse adversary model. As mentioned above, we conjecture that FIFO is in general suboptimal. There are numerous other applications of these two frameworks for evaluating online algorithms. We simply mention here two challenging open problems.

The Markov diffuse adversary. Consider again the paging problem. Suppose that the request sequence is the output sequence of an unknown *Markov chain* (intuitively, the program generating the page requests) with at most s states, which we can only partially observe via its output. That is, the output $f(q)$ of a state q of the unknown Markov process is a page in M. The allowed distributions Δ are now all output distributions of s-state Markov processes with output. We may want to restrict our online algorithms to ones that do not attempt to exhaustively learn the Markov process. One way to do this would be to bound the length of the request sequence to $O(s)$. A better way, however, is to require the additive constant d in (4) to be independent of s. We believe that this is a useful model of paging whose study and solution may enhance our understanding of the performance of actual paging systems.

The power of vision. Consider two robots, one with vision α (its visual sensors can detect objects in distance α) and the other with vision β, $\beta > \alpha$. We want to measure the disadvantage of the first robot in navigating or exploring a terrain against the second robot. Naturally, comparative analysis seems the most appropriate framework for this type of problem. Different restrictions on the terrain and the objective of the robot result in different problems but we find the following simple problem particularly challenging: On the plane, there are n objects (points). The objective of the robot is to construct a map of the plane, i.e., to find the position of all n objects. We ask what the comparative ratio $R(\mathcal{V}_\alpha, \mathcal{V}_\beta)$ for this problem is, where $\mathcal{V}_\alpha$ and $\mathcal{V}_\beta$ denote the information regimes of vision α and β, respectively.

REFERENCES

[1] S. Ben-David and A. Borodin, *A new measure for the study of on-line algorithms*, Algorithmica, 11 (1994), pp. 73–91.

[2] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, in Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing, New Orleans, LA, 1991, pp. 249–59.

[3] A. Borodin, N. Linial, and M. E. Saks, *An optimal on-line algorithm for metrical task systems*, in Proceedings of the 19th Annual ACM Symposium on the Theory of Computing, New York, 1987, pp. 373–82.

[4] M. Chrobak and L. L. Larmore, *The server problem and on-line games*, in On-Line Algorithms: Proceedings of a DIMACS Workshop, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 7 (1992), pp. 11–64.

[5] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–99.

[6] S. Irani, A. R. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, in Proceedings of the Third Annual ACM–SIAM Symposium on Discrete Algorithms, Orlando, FL, 1992, pp. 228–36.

[7] A. R. Karlin, S. J. Phillips, and P. Raghavan, *Markov paging*, in Proceedings of the 33rd Annual Symposium on Foundations of Computer Science, Pittsburgh, PA, 1992, pp. 208–17.

[8] E. Koutsoupias and C. H. Papadimitriou, *Beyond competitive analysis*, in Proceedings of the 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 394–400.

[9] E. Koutsoupias and C. H. Papadimitriou, *On the k-server conjecture*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Montreal, Canada, 1994, pp. 507–11.

[10] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for on-line problems*, in Proceedings of the 20th Annual ACM Symposium on the Theory of Computing, Chicago, IL, 1988, pp. 322–33.

[11] L. A. McGeoch and D. D. Sleator, *A strongly competitive randomized paging algorithm*, Algorithmica, 6 (1991), pp. 816–25.

[12] P. Raghavan, *A statistical adversary for on-line algorithms*, in On-Line Algorithms: Proceedings of a DIMACS Workshop, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., 7 (1992), pp. 79–83.

[13] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, in Proceedings of the 16th International Colloquium on Automata, Languages and Programming, Stresa, Italy, 1989, pp. 687–703.

[14] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[15] N. Young, *The k-server dual and loose competitiveness for paging*, Algorithmica, 11 (1994), pp. 525–41.

[16] N. E. Young, *Bounding the diffuse adversary*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, 1998, pp. 420–425.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 318–323

ON THE DIFFICULTY OF DESIGNING GOOD CLASSIFIERS*

MICHELANGELO GRIGNI†, VINCENT MIRELLI‡, AND CHRISTOS H. PAPADIMITRIOU§

Abstract. We consider the problem of designing a near-optimal linear decision tree to classify two given point sets B and W in $\Re^n$. A linear decision tree defines a polyhedral subdivision of space; it is a classifier if no leaf region contains points from both sets. We show hardness results for computing such a classifier with approximately optimal depth or size in polynomial time. In particular, we show that unless NP = ZPP, the depth of a classifier cannot be approximated within any constant factor, and that the total number of nodes cannot be approximated within any fixed polynomial. Our proof uses a simple connection between this problem and graph coloring and uses the result of Feige and Kilian on the inapproximability of the chromatic number. We also study the problem of designing a classifier with a single inequality that involves as few variables as possible and point out certain aspects of the difficulty of this problem.

Key words. linear decision tree, hardness of approximation, parameterized complexity

AMS subject classifications. 68Q17, 62H30

PII. S009753979630814X

1. Introduction.

Classifying point sets in $\Re^n$ by linear decision trees is of great interest in pattern analysis and many other applications [4, 5, 14]. Typically, in such a problem we are given a set W of *white* points and a set B of *black* points in $\Re^n$, and we must produce a linear decision tree which classifies them. That is, the tree defines a linear decision at each internal node, such that for each leaf ℓ of this tree, either only white or only black points lead the algorithm to ℓ. We call such a linear decision tree a *classifier*. In many situations W and B are not given explicitly but implicitly in terms of concepts, images of objects, etc.

The problem is already well studied. Constructing a size-optimal classifier is NP-complete even in three dimensions [10]; in high dimensions it is NP-complete even for constant size trees [2, 16]. There is much algorithmic work toward computing classifiers that meet various local optimality conditions [4, 17], but very little is known about how well such local optima approximate the optimal solution. An exception is the use of random sampling to find near-optimal splitting planes in low dimensions [10].

In this paper we prove some very strong negative results on high-dimensional classifying trees (the important case in practice). We point out a simple connection between the problem of designing optimal linear classifying trees and the classical problem of *coloring a graph*. Given a graph G, we construct its geometric realization; roughly speaking, the white points are the vertices of the graph arranged at the corners of a simplex, and the black points correspond to the edges of the graph, with each black point placed at the midpoint between the two endpoints of its edge. It is not

*Received by the editors August 15, 1996; accepted for publication (in revised form) September 16, 1999; published electronically May 15, 2000. This work was supported by the Army Research Laboratory.

http://www.siam.org/journals/sicomp/30-1/30814.html

†Department of Mathematics and Computer Science, Emory University, Atlanta, GA (mic@mathcs.emory.edu). The work of this author was performed at UCSD, supported in part by NSF grant DMS-9206251.

‡U.S. Army Research Laboratory, Fort Belvoir, VA (vmirelli@arl.mil).

§Computer Science Division, University of California at Berkeley, Berkeley, CA (christos@cs.berkeley.edu).

hard to prove then that the optimum size of any classifier is the chromatic number of the graph $\chi(G)$, while the optimum depth is $\log_2(\chi(G)+1)$. We then use the result of Feige and Kilian [8] on the inapproximability of the chromatic number to obtain the following two results.

THEOREM 1.1. *Unless NP = ZPP, no polynomial-time algorithm for optimizing the number of nodes in a classifier can approximate the optimum within any fixed polynomial. For $\epsilon > 0$ and large enough dimension n, the approximation ratio is no better than $n^{1-\epsilon}$.*

THEOREM 1.2. *Unless NP = ZPP, no polynomial-time algorithm for optimizing the depth a classifier can have approximation ratio better than any fixed constant.*

Here ZPP is the class of problems solved by polynomial expected-time *randomized* algorithms with neither false negatives nor false positives. NP = ZPP is a situation almost as unthinkable as NP = P. In the next section we prove these two results.

Finally, in section 3 we look at another aspect of the difficulty of optimizing classifiers: Suppose that the two point sets can be separated by a *single* linear inequality, but we want to find the inequality that separates them *and involves as few variables as possible*. This situation is of interest when we use functions of the points as additional coordinates to facilitate classification [4, 11]. We point out that variants of this problem are hard for various levels of the *W hierarchy* [3, 6], which implies that (unless an unlikely collapse occurs), they cannot be solved in polynomial time even if the optimum sought is small (bounded by any very slowly growing function).

2. Definitions and proofs. Let $W, B \subseteq \Re^n$ be two point sets. A *linear classifying tree* for W and B is a decision tree with internal nodes of the form $\sum_{i=1}^{n} a_i x_i > b$, each with two branches, the *true* branch and the *false* branch. A leaf ℓ of such a tree corresponds in a straightforward way to a convex cell in a subdivision of $\Re^n$, call it $C(\ell)$, containing all points that satisfy (or falsify) the inequality in each internal node I that is an ancestor of ℓ in the tree, and such that ℓ is in the *true* (respectively, *false*) subtree of I.

There are two important measures of the difficulty of such a classifier. The first is the *number of internal nodes* of the tree and corresponds to the program size of the classifier. The other is the *depth* of the tree and corresponds to the running time of the decision algorithm. We denote by $d(W, B)$ the depth of the classifier for W and B that has the smallest possible depth among all such classifiers; similarly, $n(W, B)$ is the optimum number of internal nodes.

For example, a classifier for the two 2-dimensional point sets W and B shown in Figure 2.1(a) is shown in Figure 2.1(b). The subdivisions corresponding to the leaves are also shown in Figure 2.1(a). It has depth two, and a total of three nodes. Here it is easy to see that $d(W, B) = 2$ and $n(W, B) = 2$; thus the tree shown is optimal with respect to depth, but not with respect to the number of nodes.

Let $G = (V, E)$ be any graph, with vertices $V = \{v_1, \ldots, v_n\}$ and edges $E = \{e_1, \ldots, e_m\}$. Consider the following two point sets in $\Re^n$ (indeed, on the $(n-1)$-dimensional hyperplane $\sum_{i=1}^{n} x_i = 1$): the white set $W(G) = \{w_1, \ldots, w_n\}$, where w_i is the ith elementary basis vector (that is, $(w_i)_i = 1$ and all other coordinates are zero); and the black set $B(G) = \{b_1, \ldots, b_m\}$, with $b_k = \frac{1}{2}(w_i + w_j)$, where $e_k = \{v_i, v_j\}$. In other words, the white points are the nodes of G placed at the vertices of the simplex, while the black points are the edges of G, each placed at the midpoint of its two endpoints.

The chromatic number of G, $\chi(G)$, is the smallest number of colors that can be used to color the nodes of G so that no two adjacent nodes have the same color;

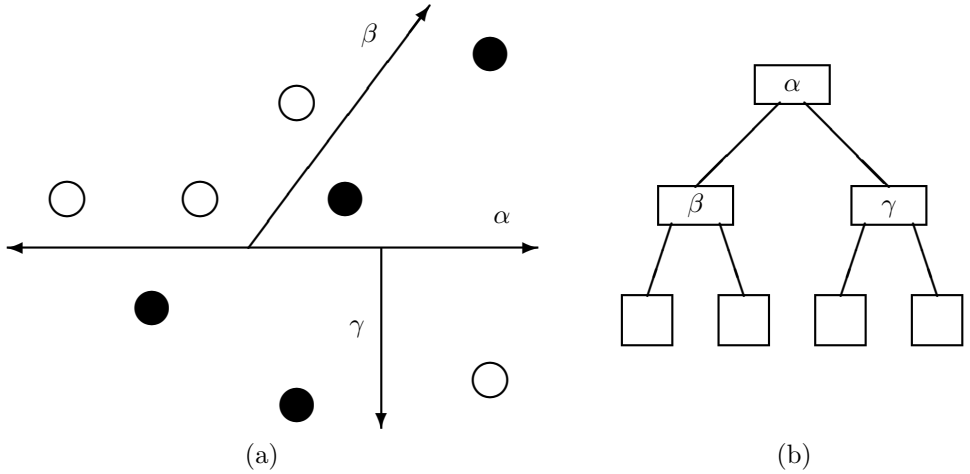


FIG. 2.1. *A 2-dimensional classifier.*

equivalently, it is the smallest number of independent sets that can be used to cover all nodes of G.

The following two lemmas now characterize the complexity of classifying $W(G)$ and $B(G)$ in terms of $\chi(G)$.

LEMMA 2.1. $n(W(G), B(G)) = \chi(G)$.

Proof. Consider any white leaf ℓ in any decision tree for $W(G), B(G)$. Since its cell $C(\ell)$ is convex, it follows that the nodes of G it contains share no edge, because otherwise the corresponding black midpoint would also be in $C(\ell)$. Thus, $C(\ell)$ contains an independent set of G. Since the leaves of the decision tree must cover all nodes of G, there are at least $\chi(G)$ white leaves in any decision tree. In addition there must be at least one black leaf, and hence there are at least $\chi(G) + 1$ leaves overall, and at least $\chi(G)$ internal nodes. It follows that $n(W(G), B(G)) \geq \chi(G)$.

For the other direction let $S_1, \ldots, S_{\chi(G)}$ be the independent sets in the optimum coloring of G. We can construct a decision tree with $\chi(G)$ internal nodes, of which the kth has the inequality $\sum_{v_i \in S_k} x_i \geq \frac{2}{3}$, with the *true* branch leading to a white leaf and the *false* branch leading to either the $k+1$st internal node or a black leaf if $k = \chi(G)$. It is easy to see that this is a classifier for $W(G), B(G)$, and hence $n(W(G), B(G)) \leq \chi(G)$. □

LEMMA 2.2. $\lceil \log_2(\chi(G) + 1) \rceil \leq d(W(G), B(G)) \leq \lceil \log_2(\chi(G) + 1) \rceil + 1$.

Proof. The lower bound follows from the previous lemma, since $d(W, B) \geq \lceil \log_2(n(W, B) + 1) \rceil$. For the upper bound, consider the optimum coloring of G with $\chi(G)$ colors. We let V_1 be the union of the first $\lfloor \frac{\chi(G)}{2} \rfloor$ color classes, and let V_2 be the remaining nodes of G. Our first inequality is $\sum_{v_i \in V_1} x_i \geq \frac{1}{3}$, and it separates the white nodes in two subgraphs, each with about half the chromatic number. Continuing the same way we arrive at nodes that contain white nodes that are independent, plus certain black nodes; these can be separated with one more internal node. The total depth is thus $\lceil \log_2 \chi((G) + 1) \rceil + 1$. □

To prove Theorems 1.1 and 1.2 from the lemmas, we now need only the following result of Feige and Kilian [8], building on earlier results of Lund and Yannakakis [15] and Fürer [9].

THEOREM 2.3. *Unless NP = ZPP, no polynomial-time algorithm for approximat-*

ing the chromatic number of a graph with n nodes can have an approximation ratio better than $n^{1-\epsilon}$ for a fixed $\epsilon > 0$ and large enough n.

In other words, for a given efficient algorithm and $\epsilon > 0$, there are graphs with chromatic number n^{ϵ}, such that the algorithm cannot find a coloring better than $n^{1-\epsilon}$. Theorem 1.1 then follows from Lemma 2.1 and Theorem 2.3, and Theorem 1.2 follows from Lemma 2.2 and Theorem 2.3.

3. Single linear decisions. In this section we point out aspects of the difficulty of classifier optimization that hold even in the case in which W and B are *separable*, that is, *there is a single linear inequality that separates W from B* (in other words, the optimum classifying tree has just one internal node). In this case we are interested in minimizing *the number of variables that are actually needed in the decision node.*

Naturally, the interesting classification problems are not linearly separable; however, the separable case is practically interesting because it arises when we introduce "extra variables" to make classification possible. For example, one may introduce low-degree monomials (products of variables) or radial basis functions (simple functions of the distance from a point) [11, 13] and then construct a linear decision tree treating the outputs of these functions as new variables. Or one could even allow more costly special-purpose classifying heuristics and also treat their outputs as variables. It is clear that any disjoint finite sets W and B may be separated given enough such extra functions, so the real question is how to minimize their number and cost. Besides the obvious consideration of computational efficiency, by the principle of *Occam's razor* we expect that optimal classifiers of this sort are in some sense "better-quality" classifiers.

We wish thus to solve the following problem: We are given two point sets $W, B \subseteq \Re^n$ that we know are separable by a single hyperplane. We are asked to find the hyperplane $\sum_{i=1}^{n} a_i x_i \geq b$ that separates W from B, and such that $|\{i : a_i \neq 0\}|$ is minimized. In another version (better suited for modeling the case of extra functions), the first $m < n$ variables are *free*, and we wish to minimize $|\{i > m : a_i \neq 0\}|$.

We next make a very useful simplification—we assume that $B = \{0\}$ (that is, there is only one black point, the origin): Given any classification problem W, B we can transform it into an equivalent classification problem $W - B, \{0\}$, where $W - B = \{w - b : w \in W \text{ and } b \in B\}$ is the *Minkowski difference*. Thus, we seek the hyperplane that separates a given point set W from the origin and has the smallest number of nonzero coefficients (respectively, excluding the coefficients of the first m variables). We call these problems the *smallest separating inequality problem* and its *version with free variables.*

Both versions of this problem are easily seen to be NP-complete. In this section we point out their high *parameterized complexity*. In [3, 6] a theory of parameterized complexity has been initiated. The issue is whether a minimization problem of the form "given instance x and integer parameter k, is the optimum k or less?" can be solved in time, say, $O(n^p)$, where n is the size of the input x, and the hidden constants (but not p) may depend on k. For some problems, such as bandwidth and node cover, such algorithms are possible; for others, no such algorithms are known. These latter problems classify into a hierarchy of classes, denoted $W[1], W[2], \ldots$, plus an ultimate class $W[P]$. Hardness of a problem (via "parameterized reductions" appropriate for these problems; see [3]) for such a class is evidence that the problem does not have a polynomial algorithm even when the parameter is severely bounded. The higher the class, the more devastating the evidence of intractability.

THEOREM 3.1. *The smallest separating inequality problem is hard for $W[2]$, and*

its version with free variables is hard for $W[P]$.

Proof. For $W[2]$-hardness we shall reduce the $W[2]$-complete *hitting set problem* [1, 12] to the minimum separating hyperplane problem. In the hitting set problem we are given a family $F = \{S_1, \ldots, S_k\}$ of subsets of some set $\{1, 2, \ldots, n\}$, and a parameter p, and we are asked to determine whether there is a set H, $|H| \leq p$, such that $H \cap S_i \neq \emptyset$ for all i. From F we construct a set of points $W = \{w_1, \ldots, w_k\} \subseteq \Re^n$, where w_i is the characteristic vector of S_i. Let $\sum_{i=1}^{n} a_i x_i = 1$ be a hyperplane separating W from the origin, and let $H = \{i : a_i \neq 0\}$. If $H \cap S_i = \emptyset$ for some i, then the hyperplane fails to separate w_i from the origin, and hence the nonzero coordinates of the hyperplane must be a hitting set. Conversely, for any hitting set H, the hyperplane $\sum_{i \in H} x_i = \frac{1}{2}$ separates W from the origin. This completes the proof of the first part.

For the second part, we shall reduce to the version of the problem with free variables the $W[P]$-complete *minimum monotone circuit value problem* [7]. In it we are given a monotone circuit, and a parameter k, and we wish to determine whether there is an input vector with k or fewer 1's that makes the output of the circuit 1. Given such a circuit with n gates, of which all but the first m are input gates, we construct the following point set W in $\Re^n$: If i is the output gate, we add to W the point $-e_i$—recall that e_i is the unit vector in the ith coordinate. If i is an OR gate with inputs j and ℓ, then we add to W the point $e_i - e_j - e_\ell$. If i is an AND gate with inputs j and ℓ, then we add to W the points $e_i - e_j$ and $e_i - e_\ell$. This completes the construction. It is not very hard to argue that there is a hyperplane separating W from the origin with k or fewer nonzero coefficients in its last $n - m$ coordinates, if and only if the given circuit has a satisfying truth assignment with k or fewer positive inputs. □

Acknowledgment. We wish to thank Mihalis Yannakakis for an interesting discussion on this problem.

REFERENCES

[1] G. Ausiello, A. D'Atri, and M. Protasi, *Structure preserving reductions among convex optimization problems*, J. Comput. System Sci., 21 (1980), pp. 136–153.

[2] A. L. Blum and R. L. Rivest, *Training a 3-node neural network is NP-complete*, Neural Networks, 5 (1992), pp. 117–127.

[3] H. L. Bodlaender, M. R. Fellows, and M. T. Hallett, *Beyond NP-completeness for problems of bounded width: Hardness for the W hierarchy*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing (STOC), Montreal, Quebec, Canada, 1994, pp. 449–458.

[4] B. E. Boser, I. M. Guyon, and V. N. Vapnik, *A training algorithm for optimal margin classifiers*, in Proceedings of the Fifth Annual ACM Workshop on Computational Learning Theory (COLT), Pittsburgh, PA, 1992, pp. 144–52.

[5] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, Wadsworth, Belmont, CA, 1984.

[6] L. Cai, J. Chen, R. Downey, and M. Fellows, *On the structure of parameterized problems in NP*, Inform. and Comput., 123 (1995), pp. 38–49.

[7] R. G. Downey, M. R. Fellows, B. M. Kapron, M. T. Hallett, and H. T. Wareham, *The parameterized complexity of some problems in logic and linguistics*, in Logical Foundations of Computer Science, Lecture Notes in Comput. Sci. 813, Springer-Verlag, New York, 1994, pp. 89–100.

[8] U. Feige and J. Kilian, *Zero knowledge and the chromatic number*, in Proceedings of the 11th Annual IEEE Conference on Computational Complexity (CCC), Philadelphia, PA, 1996, pp. 278–287.

[9] M. Fürer, *Improved hardness results for approximating the chromatic number*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science (FOCS),

Milwaukee, WI, 1995, pp. 414–421.

[10] M. T. Goodrich, V. Mirelli, M. Orletsky, and J. Salowe, *Decision Tree Construction in Fixed Dimensions: Being Global is Hard but Local Greed is Good*, Tech. Report TR-95-1, The Johns Hopkins University Press, Baltimore, MD, 1995.

[11] I. Guyon, B. Boser, and V. Vapnik, *Automatic capacity tuning of very large VC-dimension classifiers*, in Advances in Neural Information Processing Systems 5, S. J. Hanson, J. D. Cowan, and C. L. Giles, eds., Morgan-Kaufmann, San Francisco, CA, 1993, pp. 147–155.

[12] M. T. Hallett and H. T. Wareham, *A compendium of parameterized complexity results*, SIGACT News (ACM Special Interest Group on Automata and Computability Theory), 25 (1994), pp. 122–123. Also available online from ftp://ftp.csc.uvic.ca/pub/W_hierarchy/.

[13] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Macmillan, New York, 1994.

[14] D. Heath, S. Kasif, and S. Salzberg, *Learning oblique decision trees*, in Proceedings of the 13th International Joint Conference on Artificial Intelligence, Morgan-Kaufmann, San Francisco, CA, 1993, pp. 1002–1007.

[15] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.

[16] N. Megiddo, *On the complexity of polyhedral separability*, Discrete Comput. Geom., 3 (1988), pp. 325–337.

[17] S. K. Murthy, S. Kasif, and S. Salzburg, *A system for induction of oblique decision trees*, J. Artificial Intelligence Res., 2 (1994), pp. 1–33.

SIAM J. COMPUT.
Vol. 30, No. 1, pp. 324–346

TWO-PROVER PROTOCOLS—LOW ERROR AT AFFORDABLE RATES*

URIEL FEIGE† AND JOE KILIAN‡

Abstract. We introduce the *miss-match* form for two-prover one-round proof systems. Any two-prover one-round proof system can be easily modified so as to be in miss-match form. Proof systems in miss-match form have the "projection" property that is important for deriving hardness of approximation results for NP-hard combinatorial optimization problems.

Our main result is an upper bound on the number of parallel repetitions that suffice in order to reduce the error of miss-match proof systems from p to ϵ. This upper bound depends only on p and on ϵ (polynomial in $1/(1-p)$ and in $1/\epsilon$). Based on previous work, it follows that for any $\epsilon > 0$, NP has two-prover one-round proof systems with logarithmic-sized questions, constant-sized answers, and error at most ϵ.

As part of our proof we prove upper bounds on the influence of random variables on multivariate functions, which may be of independent interest.

Key words. interactive proof systems, complexity theory

AMS subject classification. 68Q15

PII. S0097539797325375

1. Introduction.

A two-prover one-round proof system [6] is a protocol by which two provers jointly try to convince a computationally limited probabilistic verifier that a common input belongs to a prespecified language. The verifier selects a pair of questions at random. Each prover sees only one of the two questions and sends back an answer. The verifier evaluates a predicate on the common input and the two questions and answers, and accepts or rejects according to the output of the predicate. For inputs in the language, the provers have a strategy (where a strategy for a prover is a function from incoming messages to outgoing messages) that always causes the verifier to accept. For inputs not in the language, regardless of the strategy used by the provers, the verifier accepts with probability at most ϵ. The smaller the value of ϵ, known as the *error*, the greater the verifier's confidence in the result of the proof system.

The class MIP(2,1) denotes those languages L accepted by a two-prover one-round proof system with a probabilistic polynomial-time verifier. More generally, an MIP(2,1) *game* is one in which a verifier engages in a single round of communication with two cooperating but not communicating players (the provers); the verifier determines based on the messages and its private coin tosses whether the players win. Similarly, in an MIP(2,1) *proof system*, "yes" instances (when $x \in L$) give rise to trivial games, in which the provers can always win (we only consider proof systems with perfect completeness) and "no" instances give rise to nontrivial games in which the provers can win with probability at most $\epsilon < 1$. We call ϵ the *error* of the proof

*Received by the editors August 4, 1997; accepted for publication (in revised form) November 23, 1999; published electronically May 15, 2000. A preliminary version of this manuscript appeared in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1994, pp. 172–183.

http://www.siam.org/journals/sicomp/30-1/32537.html

†Department of Computer Science and Applied Mathematics, The Weizmann Institute, Rehovot, Israel (feige@wisdom.weizmann.ac.il). This author is the incumbent of the Joseph and Celia Reskin Career Development Chair. This research was done in part while this author was visiting the NEC Research Institute.

‡NEC Research Institute, 4 Independence Way, Princeton, NJ (joe@research.nj.nec.com).

system. Reducing the error in MIP(2,1) proof systems (while preserving triviality for "yes" instances) is a subtle issue. A natural approach is to repeat an MIP(2,1) protocol n times and accept only if all executions are accepting. Ideally, one would hope that this method would reduce the error to ϵ^n. This is indeed true if each execution is performed with a fresh pair of provers, requiring n pairs of provers, or if the executions are performed sequentially (each prover must answer each question online before seeing the question for the next execution), requiring n rounds of communication. However, parallel repetition—in which there are only two provers and one round, and each prover sends out its answers only after receiving all its questions—is not guaranteed to reduce the error to ϵ^n [15]. Much work was invested in trying to analyze the rate at which parallel repetition reduces the error in MIP(2,1) proof systems (see, e.g., [15, 22, 7, 13, 14]).

We analyze a specific class of MIP(2,1) proof systems which we call *miss-match* proof systems. In the basic one-round proof system, the question to the first prover is composed of two "half questions" (α_1, α_2), and the first prover replies with two "half answers" (β_1, β_2). Based on $\alpha_1, \alpha_2, \beta_1$, and β_2, the verifier makes an initial decision on whether to reject or provisionally accept, pending the results of its interaction with the second prover. This acceptance predicate depends on the specific proof system and may differ from one miss-match proof system to another. The common feature of all miss-match proof systems is the way in which the second prover is used to confirm a decision to accept. Here the verifier has two options. The first, *miss* option, is to ask the second prover a null question λ, ignore the second prover's answer, and accept. The second, the *match*, is to ask the second prover one of the two half questions α_i sent to the first prover and to accept only if the second prover's answer β matches (i.e., is equal to) the half answer β_i given by the first prover. The verifier chooses its question to the second prover uniformly from $\{\alpha_1, \alpha_2, \lambda\}$. For formal definitions, see section 2.

Our main result is an upper bound, for proof systems in the miss-match form, on the number of parallel repetitions that suffice in order to reduce the error from an initial value of p to a desired value of ϵ. This upper bound is polynomial in $1/(1-p)$ and in $1/\epsilon$.

Though our upper bound applies only to proof systems in miss-match form, it can be used in order to reduce the error from one constant to another in any MIP(2,1) proof system. The reason is that any MIP(2,1) proof system can be easily transformed into miss-match form (see Proposition 3.1), with only constant overhead in communication and randomness, and insignificant loss in the error p. In [2] it was shown that any NP-language has an MIP(2,1) proof system with logarithmic-size questions and constant-size answers. The error p for these proof systems is a constant less than 1 but larger than $\frac{1}{2}$. Our main result implies that the error can be reduced to any constant $\epsilon > 0$, while increasing the question and answer sizes by only a multiplicative constant factor. We remark that these error-reducing transformations preserve triviality; on "yes" instances of the proof system the provers can always make the verifier accept.

A major application of MIP(2,1) proof systems for NP is to prove hardness of approximation results. Decreasing the error of an MIP(2,1) proof system often translates into stronger hardness of approximation results (cf. [12, 5, 4]).

Our analysis of parallel repetition of MIP(2,1) proof systems in miss-match form is based in part on an analysis of the influence of random variables on an arbitrary function. This part is presented in a self-contained way in section 4 and may be of independent interest. Essentially, we present an exact formulation of the intuition

that for any function, knowing the value of an α-fraction of the input variables (chosen at random) is expected to give at most an α-fraction of the information regarding the output of the function. Here, we measure "information" in terms of the variance of the function. Theorem 4.3 is an essentially optimal version of a lemma that appeared in the conference version of this paper [12]. It was developed with the help of Leonid Gurvits, who gave the first proof, based on Fourier analysis. In section 4, we present a more elementary proof of this theorem.

1.1. Related work. A preliminary version of this paper appeared in [12]. In discussing related work, we distinguish between work done prior to the preliminary version and work done since. We also discuss the improvements of the current version over the preliminary one.

Prior work. MIP(2,1) proof systems were introduced by Ben-Or, Goldwasser, Kilian, and Wigderson [6]. A major result there was a two-prover perfect zero-knowledge proof system for NP. The authors remarked that parallel repetition of these proof systems preserves zero-knowledge properties, but the issue of its effect on the error was not touched upon. Initial beliefs that n parallel repetitions reduce the error from ϵ to ϵ^n were refuted by an explicit example in [15]. Since then, the problem of error reduction for MIP(2,1) proof systems has attracted much attention, because of both the intellectual challenge and the potential applications of these proof systems. Applications were initially made to cryptography [18], and later to the design of efficient probabilistically checkable proofs and to proving hardness results for approximating NP-hard optimization problems [2, 20, 3].

Initially, bounds on the rate by which parallel repetition reduces the error in MIP(2,1) proof systems were obtained only in special cases (see, e.g., [8, 18]). In this respect, the result most related to our current work was the method of [7, 10] that can reduce the error in any MIP(2,1) proof system from $p > 1/2$ to $1/2 + \epsilon$ with an overhead that depends only on p and ϵ. Our work on miss-match proof systems gives a result of similar flavor that extends to arbitrarily small errors.

The most successful approach for reducing the error was through algebraic techniques [9, 19, 13]. These techniques may dramatically change the structure of the original protocol, but thereafter the analysis of error reduction becomes relatively simple and the rate of error reduction becomes exponential. Algebraic techniques became the method of choice for reducing the error in MIP(2,1) proof systems.

Our preliminary version. The preliminary version of our paper [12] had two parts. The first part, on which the current version is based, analyzed the effect of parallel repetition on MIP(2,1) proof systems of a certain form, which we called *confuse-or-compare*. The second part showed how to preserve the zero-knowledge property when algebraic error reduction techniques are applied. We shall not discuss this second part further, nor shall we discuss zero-knowledge.

Confuse-or-compare proof systems give the verifier two options of what to do with the second prover, similar to the case of miss-match proof systems. The "compare" option is identical to the "match" option. The "confuse" option was our original version of the "miss" option. Rather than send the second prover the null question, the verifier sends him a random half question γ, unrelated to α_1 and α_2. As with the miss option, the verifier ignores the second prover's answer if the confuse option is employed. The results presented for the confuse-or-compare proof systems were qualitatively similar to the results presented here for miss-match proof systems, and the analysis was slightly simpler. The reason we switched from confuse-or-compare to miss-match will be explained shortly.

The main point of the results in [12] was that they provided a way of reducing the error from one constant to another with only constant overhead. In contrast, known algebraic error reduction techniques require more than a constant overhead in modifying the original proof system. Using the confuse-or-compare proof systems, previously known hardness of approximation results could be derived under weaker complexity assumptions (e.g., under the assumption that $P \neq NP$ instead of the assumption that NP does not have quasi-polynomial algorithms). Additional results in [12] included further strengthening of hardness of approximation results for *clique* and *chromatic number*, based on observations regarding which parameters of proof systems are really relevant to obtaining such results.

Results obtained since. Concurrently with our preliminary work in [12], Verbitsky proved that for any MIP(2,1) proof system, parallel repetition can reduce the error to be arbitrarily small. However, the method of analysis used by Verbitsky does not give useful bounds on the number of repetitions needed [22]. Subsequently, Raz proved the following parallel repetition theorem, which gives almost tight bounds on the rate of error reduction by parallel repetition.

Parallel repetition theorem (*see* [21]). For any MIP(2,1) proof system with error ϵ there exists some constant $\alpha > 0$ (that depends on ϵ and on the answer length of the proof system) such that the error obtained by n parallel repetitions is at most $\epsilon^{\alpha n}$.

Raz's theorem is strongest when the initial error and answer length are constant, as then α is just some other constant. This is indeed the case in the places where the use of error reduction by transformation to a confuse-or-compare proof system was suggested, and hence Raz's theorem could replace the results of [12] in all suggested applications. Moreover, as was pointed out in [5, 4], Raz's theorem has the further advantage of not introducing the confuse rounds. Certain efficient constructions in the theory of probabilistically checkable proofs [4] (leading to better inapproximability ratios for NP-hard optimization problems) require that the underlying MIP(2,1) proof system have the "projection" property: the answer of the second prover is a function of the answer of the first prover. This indeed holds for compare rounds, but does not hold for confuse rounds. Even though the verifier ignores the answer of the second prover on the confuse rounds, the second prover cannot tell which are the confuse rounds, and for this reason cannot be used by the construction of Bellare, Goldreich, and Sudan [4]. With Raz's theorem, the problem does not arise because the confuse rounds are not needed in order to reduce the error.

Despite the above, the confuse-or-compare structure does offer advantages that cannot be achieved by straight parallel repetition. There are families of MIP(2,1) proof systems for which the number of parallel repetitions required in order to reduce the error from 3/4 to 1/8 is not bounded by any constant [14]. However, as will be discussed in section 3, any MIP(2,1) proof system with error at most 3/4 can be converted to one in miss-match form, with error at most 11/12, and then amplified by parallel repetition to achieve any constant error. This method uses only a constant number of "black box" invocations of the original proof system. Thus, we have an alternative generic amplification method that in some cases works substantially better than parallel repetition.

The current version. The main disadvantage of the confuse-or-compare structure, not having the projection property, is circumvented in the current version of the paper. Here the confuse-or-compare structure is replaced by the miss-match structure, which does have the projection property (as explained in section 2). This makes our results applicable for the construction of [4]. We analyze error reduction by parallel repetition

for miss-match proof systems and show that the number of repetitions that is required in order to reduce the error from p to ϵ is some constant that depends only on p and ϵ. Our analysis follows closely our preliminary analysis in [12], with a slight twist toward the end. (The inspiration for this slight twist came from [16].) We also improve our analysis of the influence of random variables on a function. This stronger analysis was developed with the help of Leonid Gurvits and gives essentially tight bounds. (Our previous bounds gave away unnecessary logarithmic terms.) In the current version of the paper, we omit discussion of applications of our results to the construction of efficient probabilistically checkable proofs and to obtaining hardness of approximation results. The reader is referred to [4] for this purpose.

For further discussion on error reduction by parallel repetition, see [11]. For a discussion on the influence of variables on Boolean functions, see [17].

2. Definitions. Two-prover one-round proof systems are often modeled as a game between a verifier and two cooperating but not communicating provers. It is not an adversarial game, as the strategy of the verifier is fixed in advance. Rather, it is a cooperative game with two players (provers) who coordinate a joint strategy that gives them the highest probability of winning. Modeling MIP(2,1) proof systems as games suppresses the language recognition aspects of these proof systems, but preserves the concept of error in a proof system and the issue of how error can be reduced. One may best think of a game as an instantiation of an MIP(2,1) proof system on a single input that is not in the given language. We use the following notation.

$G = G(X, Y, Q, \pi, A, B, V)$ - a two-prover one-round game;
X - set of questions to prover P_1;
Y - set of questions to prover P_2;
A - set of answers available to P_1;
B - set of answers available to P_2;
π - probability distribution on $X \times Y$;
Q - support of π. $Q \subseteq X \times Y$;
V - acceptance predicate on (X, Y, A, B).

Game G proceeds as follows. The verifier selects at random a question pair $(x, y) \in Q$, according to probability distribution π. Question x is sent to P_1, who replies with an answer $P_1(x) \in A$. Question y is sent to P_2, who replies with an answer $P_2(y) \in B$. (We identify the name of a prover and the strategy that it employs.) The verifier then evaluates a predicate $V(x, y, P_1(x), P_2(y))$ and accepts if the predicate is satisfied. The goal of the provers is to select a strategy (namely, two functions, one for each prover, specifying an answer to each possible question) that maximizes the probability that the verifier accepts. The probability that the verifier accepts under the optimal strategy of the provers is denoted by $\omega(G)$. If $\omega(G) = 1$ the provers are said to have a *perfect strategy* for G, and the game G is *trivial*. For nontrivial games, $\omega(G)$ is also called the *error* of the game. We shall be interested only in nontrivial games.

An *n-fold parallel repetition* of game G is a new game, denoted by G^n, played on n coordinates, corresponding to playing n versions of game G in parallel. The questions and answers are now n-vectors, the ith element corresponding to the ith game. The verifier treats each coordinate of G^n as an independent copy of the original game G and accepts in G^n only if it would have accepted all the n copies of G. The support set of G^n is $Q^n \subseteq X^n \times Y^n$, the answer sets are A^n and B^n. The verifier selects n question pairs $(x_i, y_i) \in Q$ independently, each according to the probability distribution π. We

denote $\vec{x} = x_1 x_2 \cdots x_n$ and $\vec{y} = y_1 y_2 \cdots y_n$. (Hence $\vec{x} \in X^n$ and $\vec{y} \in Y^n$.) A strategy for the provers is $2n$ functions, $P_1^i : X^n \longrightarrow A$ and $P_2^i : Y^n \longrightarrow B$, where $1 \leq i \leq n$. The acceptance predicate is $V^n = \bigwedge_{i=1}^n V(x_i, y_i, P_1^i(\vec{x}), P_2^i(\vec{y}))$. That is, the verifier accepts if all n copies of the original game G are accepting.

Observe that even though the verifier treats each coordinate of G^n independently, the provers may not. In particular, the answer a prover gives in coordinate i may depend on the questions that the prover receives in other coordinates. For this reason, it is not true in general that $\omega(G^n) = (\omega(G))^n$.

We shall often refer to the following special classes of games. A game is *uniform* if π is uniform on Q. A game is *free* if it is uniform and has full support, i.e., $Q = X \times Y$. (The term *free* was introduced in [8].) A game has the *projection* property if the question to prover P_2 is a projection of the question to prover P_1, and the verifier accepts only if the answer of prover P_2 is a projection of the answer to prover P_1. More formally, the question sets are of the form $X = X_1 \times X_2 \times \cdots \times X_k$ for some k, and $Y = \bigcup_{i=1}^k X_i$, where the sets X_i are disjoint. The support of the probability distribution π includes only pairs $(x, y) \in X \times Y$ for which y is x_i for some $i \in \{1, 2, \ldots, k\}$, where x_i is the ith component of the question $x = (x_1, x_2, \ldots, x_k)$. The answer sets are of the form $A = A_1 \times A_2 \times \cdots \times A_k$ for the same k as above, and $B = \bigcup_{i=1}^k A_i$, where the sets A_i are disjoint. The acceptance predicate V is a conjunction of two parts $V = V_1 \wedge V_2$. One is a predicate $V_1(X, A)$ that ignores the communication with P_2. The other is $V_2(X, Y, A, B)$, which accepts only if the answer of P_2 is precisely the ith component of the answer of P_1, where i is the same coordinate on which the question y is a projection of the question x. We observe that games with the projection property may be uniform, but cannot be free (unless the cardinality of X is one).

We are now ready to define *miss-match* games. These are games with the projection property, also satisfying the following requirements. The question set to prover P_1 is $X = X_1 \times X_2 \times \{\lambda\}$, where λ is a character which the reader can interpret as saying "a miss." Hence the question set to P_2 is $Y = X_1 \bigcup X_2 \bigcup \{\lambda\}$. The probability distribution π is arbitrary on $X_1 \times X_2$, and uniform with respect to selecting which of the three components of $x \in X$ to choose as y. Hence we shall often refer to π as a probability distribution over $X_1 \times X_2$. The answer set for prover P_1 is $A = A_1 \times A_2 \times \{\lambda\}$. Hence the answer set for P_2 is $B = A_1 \bigcup A_2 \bigcup \{\lambda\}$. Since the third component of the question to P_1 is $\{\lambda\}$ and the third component of his answer is $\{\lambda\}$, the acceptance predicate V_1 is defined only on $(X_1 \times X_2, A_1 \times A_2)$. The full acceptance predicate is $V = V_1 \wedge V_2$, where V_2 is the predicate for the projection property. Observe that the projection property forces P_2 to answer λ with a λ.

When the question to P_2 is λ we say that it is a *miss*. When the question to P_2 is in $X_1 \bigcup X_2$ we say that it is a *match*. When a miss-match game is repeated many times in parallel, some of the rounds will be miss rounds and the others will be match rounds. Though there is no advantage in introducing a miss when the game is played only once, our analysis of the error of the repeated game will make use of the presence of the miss rounds.

3. Preliminaries and main results. Though miss-match games are a restricted form of games, any other MIP(2,1) game can be easily transformed into a miss-match game. Let $G(X, Y, Q, \pi, A, B, V)$ be an arbitrary game. Then its miss-match version $G' = G'(X', Y', Q', \pi', A', B', V')$ is as follows:

- $X' = X \times Y \times \{\lambda\}$;
- π' is identical to π over $X \times Y$;

- $A' = A \times B \times \{\lambda\}$.

Recall that $V' = V_1 \wedge V_2$. V_1 is identical to V on (X, Y, A, B). The rest of the parameters (Y', Q', B', V_2) are implicit from G' being a miss-match game. Note that if G is trivial, then G' is trivial.

PROPOSITION 3.1. *For the games G and G' as described above, $\omega(G') \leq \frac{\omega(G)+2}{3}$.*

Proof. Recall that in G', prover P_2 receives questions $y' \in X \bigcup Y \bigcup \{\lambda\}$. Fix an arbitrary deterministic strategy for P_2 in G'. (P_2's strategy can be made deterministic without loss of generality.) For every question $x \in X$, this fixes a unique answer $a \in A$, and for every question $y \in Y$, this fixes a unique answer $b \in B$. Observe that the provers in the original game G can follow this strategy, and then the probability that V is satisfied is at most $\omega(G)$. Now fix an arbitrary strategy for P_1, who receives questions $x' \in X \times Y \times \{\lambda\}$. Use the strategies of P_1 and P_2 in order to partition all questions x' to P_1 into two classes. The first class contains those x' for which P_1's answers in $A \times B$ agree with P_2's fixed strategy. The second class contains those x' for which at least one of P_1's two answers in $A \times B$ disagrees with P_2's fixed strategy. Let q denote the probability that x' is from the first class; x' is from the second class with probability $(1-q)$. The joint probability that x' is from the first class and V_1 is satisfied is at most q and is also at most $\omega(G)$. (Otherwise, the provers for the original game can win with probability higher that $\omega(G)$.) Hence this joint probability is at most $\min[q, \omega(G)] \leq (\omega(G) + 2q)/3$. The joint probability that x' is from the second class and V_2 is satisfied is at most $\frac{2}{3}(1-q)$: whenever x' is from the second class (probability $(1-q)$), there is a $1/3$ chance of the inconsistent answer being detected. Thus, the acceptance probability is at most $(2 + \omega(G))/3$, the sum of these upper bounds. □

For a miss-match game G', we say that prover P_1 has a *projection strategy* if, for every two questions $x = (x_1, x_2, \lambda)$ and $x' = (x'_1, x'_2, \lambda)$ and their respective answers $a = (a_1, a_2, \lambda)$ and $a' = (a'_1, a'_2, \lambda)$, it holds that $a_1 = a'_1$ whenever $x_1 = x'_1$, and $a_2 = a'_2$ whenever $x_2 = x'_2$. Let the *basic error* $p(G')$ be the maximum probability that V_1 is satisfied, where the maximum is taken over all projection strategies of P_1. For miss-match games, we will find it more convenient to work with p rather than ω. Note that when transforming an arbitrary game G to a miss-match game G', there is a correspondence between deterministic strategies for G and projection strategies for G'. We thus obtain the following simple relation between $p(G')$ and $\omega(G)$.

PROPOSITION 3.2. *For G and G' as above, $p(G') = \omega(G)$.*

Propositions 3.1 and 3.2 imply that $\omega(G') \leq \frac{p(G')+2}{3}$.

The following theorem is our main result.

THEOREM 3.3. *Let G' be an arbitrary miss-match game with basic error $p < 1$. Then $\omega((G')^n) \leq \epsilon$, whenever $n \geq c/((1-p)\epsilon)^c$, where $c \geq 0$ is a universal constant independent of G', p, n, and ϵ.*

Using Proposition 3.2 and the fact that for any miss-match game G', $p(G') \leq \omega(G')$, we can restate Theorem 3.3 in terms of the original win probabilities.

COROLLARY 3.4. *For G' an arbitrary miss-match game, $\omega((G')^n) \leq \epsilon$ when $n \geq c/((1-\omega(G'))\epsilon)^c$, where $c \geq 0$ is some universal constant independent of G', p, and ϵ. Furthermore, for G an arbitrary nontrivial MIP(2,1) game, $\omega((G')^n) \leq \epsilon$ when $n \geq c/((1-\omega(G))\epsilon)^c$, where G' is obtained by transforming G as described above.*

We remark that the proof of our theorem is robust enough to support simple modifications to the notion of a miss-match game. (These modifications change only the value of the constant c.) In particular, the requirement that $X = X_1 \times X_2 \times \{\lambda\}$ can be relaxed to $X = X_1 \times X_2 \times \cdots \times X_k \times \{\lambda\}$ for any fixed k, and the requirement

that the question to P_2 is chosen uniformly from the $k+1$ parts of the question to P_1 can be relaxed to any other distribution with full support.

To appreciate Theorem 3.3, one should contrast it with the following theorem of [14].

Feige–Verbitsky theorem (*see* [14]). There exists an infinite family $\mathcal{G}$ of free games such that $\omega(G) \leq 3/4$ for every $G \in \mathcal{G}$, and for any n there is some $G \in \mathcal{G}$ with $\omega(G^n) \geq 1/8$.

Hence Theorem 3.3 does not hold for arbitrary games, and there is something special in the miss-match form that makes it work. Miss-match games have two ingredients—the miss (sending an occasional λ to P_2) and the match (the projection property). It turns out that the miss property alone does not suffice in order to prove Theorem 3.3. The constant $1/2$ in the following proposition is arbitrary and can be replaced with any other constant.

PROPOSITION 3.5. *Let $\lambda \circ G$ denote a modification of a game G in which, with probability $1/2$, the question to P_2 is replaced by λ, and the verifier accepts. For any* free *game G, and for any integer $n \geq 1$, $\omega((\lambda \circ G)^n) \geq \omega(G^n)$.*

Proof. Fix an optimal strategy S for P_1 and P_2 in G^n. Based on S, we design a randomized strategy for the provers in $(\lambda \circ G)^n$. P_1 deterministically answers each question of $(\lambda \circ G)^n$ with the answer that P_1 would have given on the same question using strategy S. P_2 uses the following randomized strategy. Independently, in each coordinate on which P_2 receives a λ in $(\lambda \circ G)^n$, prover P_2 replaces λ by a question $y \in Y$ chosen uniformly at random. As a result, the question that P_2 receives in $(\lambda \circ G)^n$ is transformed into a randomly distributed question in G^n. Moreover, the assumption that G is a free game implies that the question pair that P_1 and P_2 now hold is distributed uniformly at random over all question pairs of the game G^n. Hence if the provers use strategy S, then the probability that the verifier of G^n accepts (taken over the choice of question to P_1 and transformed question to P_2) is at least $\omega(G^n)$. Indeed, P_2 answers the transformed question according to strategy S. □

COROLLARY 3.6. *There exists an infinite family $\mathcal{G}$ of free games such that $\omega(\lambda \circ G) \leq 7/8$ for every $G \in \mathcal{G}$, and for any n there is some $G \in \mathcal{G}$ with $\omega((\lambda \circ G)^n) \geq 1/8$.*

Proof. Apply Proposition 3.5 to the family of games from the Feige–Verbitsky theorem [14] cited above. □

Corollary 3.6 implies that there are families of games that have the miss property yet for which parallel repetition does not yield the error reduction obtained in Theorem 3.3. We were unable to resolve the question of whether the projection property alone makes Theorem 3.3 work or whether the combination of projection and miss is required.

We now give some intuition to explain why having miss rounds facilitates error reduction through parallel repetition. To win several parallel games with high probability, the provers must coordinate their strategies very carefully. A miss round disrupts this coordination: each prover has less understanding about what the other prover knows. Unfortunately, we know of no direct, easily motivated way of analyzing this effect.

We now give some intuition for the proof of Theorem 3.3. First, we actually analyze the error rate of G^{4n}, not G^n; the difference between n and $4n$ is easily swallowed up by an appropriate choice of c. In the analysis, out of the $4n$ rounds of G^{4n}, we concentrate only on n rounds, of which only $m \ll n$ are match rounds and the rest are miss rounds. (In fact, our analysis is somewhat simplified if we start directly with a proof system that has n rounds, of which m rounds chosen at random are match

rounds, and $n - m$ rounds are miss rounds.) Oversimplifying, P_1 has two possible modes of behavior. One is to employ a projection strategy for all n rounds, answering each round independently, and within each round using a projection strategy. The other is to base the answer in any particular round on the questions in all other rounds. If P_1 employs a projection strategy, then he is not using the fact that rounds are repeated in parallel, and standard Chernoff bounds imply that the probability of simultaneous success on all rounds is low. If P_1 does not employ a projection strategy, then his answers on the m match rounds highly depend on the questions received on the $n - m$ miss rounds. But prover P_2 receives only a λ on the miss rounds and has no idea what P_1 receives. So P_2 cannot know how to answer the match rounds in a way that indeed produces a match.

The rest of the paper is organized as follows. The proof of the main theorem itself appears in section 6. Prior to the proof, in sections 4 and 5, we present some general properties of multivariate functions. In section 4 we study the influence of a small number of random variables on the value of a function. This is related to the information available to the second prover, who sees only the questions on m match rounds, regarding the output of the first prover on these m rounds, which is a function of the questions of all n rounds. In section 5 we characterize the "gray area" between the two extreme strategies for the first prover in the simplified overview above: the projection strategy and a strategy in which the answer in each round is highly influenced by the questions in all other rounds. We show that in some exact sense, there is no gray area in between.

4. The influence of random variables on a function. The results of this section were developed with the help of Leonid Gurvits.

Let Q be a finite set and let $x_i \leftarrow \pi_i$ denote a random variable $x_i \in Q$ chosen at random according to probability distribution π_i over Q. Let $\pi = \pi_1 \times \pi_2 \times \cdots \times \pi_n$ be a product distribution over Q^n, and let $x \leftarrow \pi$ denote a random n-vector chosen according to probability distribution π. That is, for each coordinate i of x, $x_i \leftarrow \pi_i$, independently of the other coordinates of x. Let $f : Q^n \to \mathcal{R}^\ell$ be a function whose values are points in ℓ-dimensional real space, and let the *mean* $\mu[f, \pi]$ of f under π denote the expectation of f over the choice of $x \leftarrow \pi$ (which is the center of mass of the points in $\mathcal{R}^\ell$ if probability is interpreted as mass). That is,

$$\mu[f, \pi] \stackrel{\text{def}}{=} \mathrm{E}_{x \leftarrow \pi}[f(x)] = \sum_x \pi(x) f(x).$$

Similarly, we define the variance $\sigma^2[f, \pi]$ by

$$\sigma^2[f, \pi] \stackrel{\text{def}}{=} \mathrm{E}_{x \leftarrow \pi}\left[(f(x) - \mu[f, \pi])^2\right] = \mathrm{E}_{x \leftarrow \pi}[f(x)^2] - \mu[f, \pi]^2.$$

Let $[n]$ denote the set of integers from 1 to n. For $m \le n$, we will be interested in m-vectors $\vec{\imath} \in [n]^m$ in which all coordinates are distinct and ascending. Thus, $\vec{\imath}$ denotes an m-element subset of $[n]$. As a convention, $\vec{\imath} \leftarrow [n]^m$ denotes a vector chosen uniformly at random from all vectors of $[n]^m$ with distinct ascending entries. Let $\vec{q}$ denote a vector in Q^m. Let $x[\vec{\imath}]$ denote the projection of x to the coordinates indexed by $\vec{\imath}$ ($x[i]$, for scalar i, is interpreted in the obvious manner).

DEFINITION 4.1. *An m-block $B = (\vec{\imath}, \vec{q})$ is a sequence of m pairs (i_j, q_j) (for $1 \le j \le m$), where $q_j \in Q$, $i_j \in [n]$, and $i_j < i_k$ when $j < k$. In our intended use, an m-block specifies the values of m of the input variables for an n-variate function.*

A vector $x \in Q^n$ *satisfies* the m-block B if for every pair $(i_j, q_j) \in B$, $x_{i_j} = q_j$. For a fixed m-block B, let the conditional mean $\mu[f, \pi|B]$ denote the expectation of

$f(x)$ when x is chosen at random according to π, conditioned on x satisfying B. Let $B \overset{m}{\leftarrow} \pi$ denote an m-block $(\vec{i}, \vec{q})$ chosen at random, where $\vec{i} \leftarrow [n]^m$, and each entry of q_j of $\vec{q}$ is chosen independently according to π_{i_j}.

Clearly, for any function f, the expectation of the conditional mean is the same as the mean. That is, $\mathrm{E}_{B \overset{m}{\leftarrow} \pi} \mu[f, \pi|B] = \mu[f, \pi]$. We shall show that if m is small compared to n, then for most choices of an m-block B, the conditional mean is close to the mean. In other words, the value of x on m random coordinates is unlikely to significantly influence the mean of $f(x)$. Our analysis is essentially tight.

We define the *inner product* of two vectors u and v by $\langle u, v \rangle \overset{\text{def}}{=} \sum u_i v_i$ and the *norm* of vector x by $\|x\| \overset{\text{def}}{=} \sqrt{\langle x, x \rangle}$.

DEFINITION 4.2. *For a function $f : Q^n \to \mathcal{R}^\ell$ and probability distribution π, the* m-variance $\sigma^2[f, \pi, m]$ *is the mean square distance between the mean of f and the conditional mean of f. Formally,*

$$\sigma^2 [f, \pi, m] \overset{def}{=} \mathrm{E}_{B \overset{m}{\leftarrow} \pi} \left[\| \mu [f, \pi|B] - \mu [f, \pi] \|^2 \right].$$

By the linearity of expectation and of the inner product operation, it can be easily verified that $\mathrm{E}_{B \overset{m}{\leftarrow} \pi}[\|\mu[f, \pi|B] - \mu[f, \pi]\|^2] = \mathrm{E}_{B \overset{m}{\leftarrow} \pi}[\|\mu[f, \pi|B]\|^2] - \|\mu[f, \pi]\|^2$, giving an alternative formulation of the variance which we shall often use. To simplify notation, we use the convention that v^2 is interpreted as $\|v\|^2$ whenever v is a vector.

The m-variance generalizes the notion of the variance in the sense that when $m = n$ (that is, the input as a whole is revealed), $\sigma^2[f, \pi, n] = \mathrm{E}_{x \leftarrow \pi}[\|f(x)\|^2] - \|\mu[f, \pi]\|^2 = \sigma^2[f, \pi]$. Intuitively, the m-variance is related to the influence that a random set of m variables has on the value of a function.

For simple examples of the concept of m-variance, consider the following Boolean functions, where $n > 1$, π is uniform over $\{0, 1\}^n$, and x_i denotes the ith variable (coordinate) of x. The parity function $f(x) = \sum_{i=1}^n x_i (\mathrm{mod}\ 2)$ has zero m-variance for all $m \leq n - 1$, since as long as one variable remains unknown, the parity function remains balanced. The majority function, $f(x) = 1$ iff $\sum_{i=1}^n x_i \geq n/2$, has 1-variance $\Theta(1/n)$, because knowing the value of one variable biases the majority function by $\Theta(1/\sqrt{n})$. More generally, the k-majority function, $f(x) = 1$ iff $\sum_{i=1}^k x_i \geq k/2$, also has 1-variance $\Theta(1/n)$ for any k, because with probability k/n the sampled variable is among the k variables on which the majority is computed, and then it biases their majority by $\Theta(1/\sqrt{k})$.

Note that if the function f depends only on one variable, then $\sigma^2[f, \pi, m] = m\sigma^2[f, \pi]/n$, since a block B influences the value of f if it contains the distinguished variable (which happens with probability m/n), and then it completely determines the value of the function. The following theorem generalizes this to arbitrary functions.

THEOREM 4.3. *Let $f : Q^n \to \mathcal{R}^\ell$ be an arbitrary function and let π be an arbitrary product probability distribution on Q^n. Then for any m, where $1 \leq m \leq n$, the m-variance of f satisfies*

$$\sigma^2 [f, \pi, m] \leq \frac{m}{n} \sigma^2 [f, \pi].$$

Proof. We first concentrate on $f : Q^n \to \mathcal{R}$; we relax this restriction later.

In our computations of expectations and variances, x_i is always distributed according to π_i. We will at times specify a set of variables in the subscript of an expectation {variance}; this denotes taking the expectation {variance} over the choice of all the variables in the subscript, where each variable x_i is independently distributed according to π_i.

Given a subset of the variable indices, $S = \{i_1, \ldots, i_k\}$, we define f_S by

$$f_S(x_{i_1}, \ldots, x_{i_k}) = \mathrm{E}_{x_j : j \notin S}[f(x_1, \ldots, x_n)].$$

That is, f_S gives the expected value of f conditioned on the values of $x_{i_1}, \ldots, x_{i_k}$. We define $\pi_S = (\pi_{i_1}, \ldots, \pi_{i_k})$ and $\sigma^2[f, \pi, S]$ by

$$\sigma^2[f, \pi, S] \stackrel{\text{def}}{=} \sigma^2[f_S, \pi_S] = \mathrm{E}_{x_i : i \in S}[f_S(x_{i_1}, \ldots, x_{i_k})^2] - \mu[f, \pi]^2, \tag{4.1}$$

where the latter equality follows from $\mu[f, \pi] = \mu[f_S, \pi_S]$.

By definition, $\sigma^2[f, \pi, m] = \mathrm{E}_S[\sigma^2[f, \pi, S]]$, where S is distributed uniformly over m-element subsets of $[n]$. The crux of our proof is to upper bound $\sigma^2[f, \pi, S]$ by a quantity I_S whose expected value (over the choice of S) is $(m/n)\sigma^2[f, \pi]$. Let $P = i_1, \ldots, i_n$ be an arbitrary ordering of the variables. We define

$$I_{i_j} = \sigma^2[f, \pi, \{i_1, \ldots, i_j\}] - \sigma^2[f, \pi, \{i_1, \ldots, i_{j-1}\}]$$

and

$$I_S = \sum_{i \in S} I_i.$$

When P is unclear we write $I_i(P)$. It follows via a telescoping sum that $\sigma^2[f, \pi] = \sum_i I_i$; by the linearity of expectation, it follows that $\mathrm{E}_S[I_S] = (m/n) \sum_i I_i$, hence $\mathrm{E}_S[I_S] = (m/n)\sigma^2[f, \pi]$. The theorem, restricted to the case where the range of f is $\mathcal{R}$, then follows immediately from Lemma 4.4 below.

To extend the proof to the case that the range of f is $\mathcal{R}^\ell$, consider an arbitrary orthonormal basis for $\mathcal{R}^\ell$. Now view f as ℓ different functions $f_1, \ldots, f_\ell$, where $f_i(x)$ is the projection of $f(x)$ on the ith basis vector. For each function f_i separately, the range is just $\mathcal{R}$, and hence the theorem holds. As the square of the distance of two points in $\mathcal{R}^\ell$ is the sum of squares of their distances when projected to the ℓ basis vectors, the theorem follows also for f. □

LEMMA 4.4. *For any ordering P on $\{1, \ldots, n\}$, $\sigma^2[f, \pi, S] \leq I_S$.*

For our purposes, a single P that works for all S would suffice, but considering an arbitrary P will be useful in the proof. Before proving Lemma 4.4, we prove a useful, presumably known inequality on variances.

PROPOSITION 4.5. *Let $f : Q^2 \longrightarrow R$ be an arbitrary function and let (x, y) be distributed according to an arbitrary product probability distribution on Q^2. Then*

$$\sigma_x^2[\mathrm{E}_y[f(x, y)]] \leq \mathrm{E}_y[\sigma_x^2[f(x, y)]].$$

Proof. We first observe that for any function $g(y)$, the function $f'(x, y) = f(x, y) + g(y)$ satisfies $\sigma_x^2[f'(x, y)] = \sigma_x^2[f(x, y)]$ for all y, and $\mathrm{E}_y[f'(x, y)] = \mathrm{E}_y[f(x, y)] + \mathrm{E}_y[g(y)]$ for all x. It follows that $\sigma_x^2[\mathrm{E}_y[f(x, y)]] \leq \mathrm{E}_y[\sigma_x^2[f(x, y)]]$ iff $\sigma_x^2[\mathrm{E}_y[f'(x, y)]] \leq \mathrm{E}_y[\sigma_x^2[f'(x, y)]]$, since both quantities are unchanged by this translation. We therefore assume without loss of generality that for all y, $\mathrm{E}_x[f(x, y)] = 0$; if not, translate by $g(y) = -\mathrm{E}_x[f(x, y)]$. It follows that $\mathrm{E}_x \mathrm{E}_y[f(x, y)] = 0$, and hence that

$$\sigma_x^2[\mathrm{E}_y[f(x, y)]] = \mathrm{E}_x \mathrm{E}_y[f(x, y)^2].$$

Similarly, it follows that

$$\begin{aligned} \mathrm{E}_y[\sigma_x^2[f(x, y)]] &= \mathrm{E}_y \mathrm{E}_x[f(x, y)^2] \\ &= \mathrm{E}_x \mathrm{E}_y[f(x, y)^2]. \end{aligned}$$

Fixing x, we have $\sigma_y^2[f(x,y)] = \mathrm{E}_y[f(x,y)^2] - \mathrm{E}_y[f(x,y)]^2$. Hence, by the positivity of the variance, $\mathrm{E}_y[f(x,y)]^2 \le \mathrm{E}_y[f(x,y)^2]$. The proposition follows. □

Proof of Lemma 4.4. When S is a prefix of P, then by the definitions and a telescoping sum it follows that $\sigma^2[f,\pi,S] = I_S$. To show the inequality for arbitrary S, we make incremental changes $P \to P'$ such that $I_S(P') \le I_S(P)$, finally obtaining an ordering Q that contains S as a prefix. Then, $I_S(Q) = \sigma^2[f,\pi,S]$ and $I_S(Q) \le I_S(P)$, implying the lemma.

For ease of exposition, we assume without loss of generality that $P = 1,\ldots,n$. We consider the ordering

$$P' = 1,\ldots,i-2,i,i-1,i+1,\ldots,n,$$

where $i \in S$ and $i-1 \notin S$. That is, we move a variable indexed by S one step closer to the beginning of the ordering, moving it past a variable that is not in S. Clearly, a sequence of such operations can be used to move all the variables of S to the beginning of the sequence. It remains to show that $I_S(P') \le I_S(P)$. Clearly, $I_j(P') = I_j(P)$ for $j \notin \{i-1,i\}$. It thus suffices to show that $I_i(P') \le I_i(P)$, since I_{i-1} does not affect I_S.

Let $A = \{1,\ldots,i\}$, $B = \{1,\ldots,i-1\}$, $C = \{1,\ldots,i-2,i\}$, and $D = \{1,\ldots,i-2\}$. Expanding out the definitions, we obtain,

$$\begin{aligned} I_i(P) = {} & \left(\mathrm{E}_{x_1,\ldots,x_{i-2}}\mathrm{E}_{x_{i-1}}\mathrm{E}_{x_i}[f_A(x_1,\ldots,x_i)^2] - \mu[f,\pi]^2\right) \\ & - \left(\mathrm{E}_{x_1,\ldots,x_{i-2}}\mathrm{E}_{x_{i-1}}[f_B(x_1,\ldots,x_{i-1})^2] - \mu[f,\pi]^2\right). \end{aligned}$$

Given $x_1,\ldots,x_{i-2}$, we define $g(x_{i-1},x_i) = f_A(x_1,\ldots,x_i)$. Noting that

$$f_B(x_1,\ldots,x_{i-1}) = \mathrm{E}_{x_i}[f_A(x_1,\ldots,x_i)],$$

we obtain,

$$I_i(P) = \mathrm{E}_{x_1,\ldots,x_{i-2}}\mathrm{E}_{x_{i-1}}[\sigma_{x_i}^2[g(x_{i-1},x_i)]].$$

By similar manipulations, we obtain

$$\begin{aligned} I_i(P') &= \left(\mathrm{E}_{x_1,\ldots,x_{i-2}}\mathrm{E}_{x_i}[f_C(x_1,\ldots,x_{i-2},x_i)^2] - \mu[f,\pi]^2\right) \\ &\quad - \left(\mathrm{E}_{x_1,\ldots,x_{i-2}}[f_D(x_1,\ldots,x_{i-2})^2] - \mu[f,\pi]^2\right) \\ &= \mathrm{E}_{x_1,\ldots,x_{i-2}}[\sigma_{x_i}^2[\mathrm{E}_{x_{i-1}}[g(x_{i-1},x_i)]]]. \end{aligned}$$

However, by Proposition 4.5 it follows that for all $x_1,\ldots,x_{i-2}$,

$$\sigma_{x_i}^2[\mathrm{E}_{x_{i-1}}[g(x_{i-1},x_i)]] \le \mathrm{E}_{x_{i-1}}[\sigma_{x_i}^2[g(x_{i-1},x_i)],$$

implying the lemma. □

Theorem 4.3 has the following useful corollary, that we shall use repeatedly in subsequent parts of the paper.

COROLLARY 4.6. *Let Q and A be finite sets, let $f : Q^n \to A$ be an arbitrary function, and let π be an arbitrary product probability distribution on Q^n. Then the influence of a random m-block on the probability that $f(x)$ attains any particular value $a \in A$ is bounded as follows:*

$$\mathrm{E}_{B \overset{m}{\leftarrow} \pi}\left[\sum_{a \in A}\left(\Pr_{x \leftarrow (Q^n,\pi)}[f(x) = a] - \Pr_{x \leftarrow (\pi|B)}[f(x) = a]\right)^2\right] \le m/n.$$

Proof. By the linearity of expectation, it suffices to show that

$$\sum_{a\in A} \mathrm{E}_{B\overset{m}{\leftarrow}\pi}\left[\left(\Pr_{x\leftarrow\pi}\left[f\left(x\right)=a\right]-\Pr_{x\leftarrow(\pi|B)}\left[f\left(x\right)=a\right]\right)^2\right]\le m/n. \tag{4.2}$$

For each $a \in A$, define $f_a : Q^n \to \{0,1\}$ by $f_a(x) = 1$ if $f(x) = a$ and $f_a(x) = 0$ otherwise. For any distribution D, we have $\mu[f_a, D] = \Pr_{x\leftarrow D}[f(x) = a]$. By Definition 4.2, we can write the left-hand side of (4.2) as $\sum_{a\in A}\sigma^2[f_a,\pi,m]$. By Theorem 4.3, $\sigma^2[f_a,\pi,m] \le \frac{m}{n}\sigma^2[f_a,\pi]$. Now, $\sigma^2[f_a,\pi] = p_a(1-p_a) \le p_a$, where $p_a = \Pr_{x\leftarrow\pi}[f(x) = a]$, so

$$\sum_{a\in A}\sigma^2[f_a,\pi,m] \le \frac{m}{n}\sum_{a\in A} p_a = \frac{m}{n}. \qquad \square$$

5. Correlations within multivalued functions. Let Q and A be finite sets, let $f : Q^n \to A^n$ be a function, and let π be a probability distribution over Q, extended as a product distribution π^n over Q^n. Recall that $\vec{\imath} \in [n]^m$ denotes a vector of m distinct, ascending entries in $\{1,\ldots,n\}$, the notion of an m-block $(\vec{\imath},\vec{q})$, and the projection notation from section 4. Let $\Pr[\vec{a}|(\vec{\imath},\vec{q})]$ denote the probability that $f(x)[\vec{\imath}] = \vec{a}$, conditioned on $x[\vec{\imath}] = \vec{q}$ (i.e., the probability is taken over $x \leftarrow (Q^n, \pi^n|(\vec{\imath},\vec{q}))$). This notation is extended in a natural way to $\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)]$ (in which case the conditioning is done on one coordinate more than the projection), and to $\Pr[\vec{a},a|(\vec{\imath},\vec{q})(i,q)]$ for $a \in A$. Throughout we use the convention that i has to be distinct from all coordinates of $\vec{\imath}$.

DEFINITION 5.1. *For a parameter $\varepsilon > 0$, the m-triple $(\vec{\imath},\vec{q},\vec{a})$ is* alive *if $\Pr[\vec{a}|(\vec{\imath},\vec{q})] \ge \varepsilon$. The m-block $(\vec{\imath},\vec{q})$ is* alive *if for some $\vec{a}$ the m-triple $(\vec{\imath},\vec{q},\vec{a})$ is alive.*

DEFINITION 5.2. *An m-block $(\vec{\imath},\vec{q})$ $(1-\eta)$-*determines *a pair (i,q) if for every live m-triple $(\vec{\imath},\vec{q},\vec{a})$ there exists $a \in A$ such that*

$$\Pr\left[\vec{a},a|\left(\vec{\imath},\vec{q}\right)(i,q)\right] \ge (1-\eta)\Pr\left[\vec{a}|\left(\vec{\imath},\vec{q}\right)(i,q)\right].$$

The above inequality may be thought of in terms of conditional probability. Conditioned on $x[\vec{\imath},i] = \vec{q},q$ and $f(x)[\vec{q}] = \vec{a}$, $f(x)[i] = a$ with probability at least $(1-\eta)$.

DEFINITION 5.3. *An m-block $(\vec{\imath},\vec{q})$ is* $(1-\eta)$-good *if it is alive and* $(1-\eta)$-determines *a randomly chosen pair (i,q) ($i \leftarrow ([n]-\vec{\imath})$ and $q \leftarrow \pi$) with probability at least $1-\eta$.*

It is useful to consider the "majority answer" guess for the value of $f(x)$ at some coordinate, given partial information about x and $f(x)$.

DEFINITION 5.4. *Given $(\vec{\imath},\vec{q},\vec{a})$, we define* $\mathrm{maj}(i,q)$ *to be the most likely (breaking ties lexicographically) value for $f(x)[i]$, over the choice of x, conditioned on $x[\vec{\imath}] = \vec{q}$, $f(x)[\vec{\imath}] = \vec{a}$, and $x[i] = q$.*

A good block $(\vec{\imath},\vec{q})$ has the property that for any live $(\vec{\imath},\vec{q},\vec{a})$ a large fraction of the other answers are expected to agree with the majority answer defined above.

The following proposition will be used in the proof of our main theorem.

PROPOSITION 5.5. *Suppose that $(\vec{\imath},\vec{q})$ is $(1-\eta)$-good, for $\eta < \frac{1}{2}$, and that $(\vec{\imath},\vec{q},\vec{a})$ is alive. Then $f(x)[i] = \mathrm{maj}(i,q)$ with probability at least $1 - 2\eta/\varepsilon$ in the following experiment:*

1. *Choose x according to π^n, conditioned on $x[\vec{\imath}] = \vec{q}$ and $f(x)[\vec{\imath}] = \vec{a}$.*
2. *Choose i uniformly from $[n] - \vec{\imath}$.*
3. *Let $q = x[i]$.*

Proof. Consider the experiment of choosing (i,q) by $i \leftarrow ([n]-\vec{\imath})$, $q = x[i]$, conditioned on $x[\vec{\imath}] = \vec{q}$ but *not* conditioned on $f(x)[\vec{\imath}] = \vec{a}$. Since $(\vec{\imath},\vec{q})$ is good, by Definition 5.3, (i,q) fails to be $(1-\eta)$-determined by $(\vec{\imath},\vec{q})$ with probability at most η. The probability that it fails to be $(1-\eta)$-determined, conditioned on $f(x)[\vec{\imath}] = \vec{a}$, is at most $\eta/\Pr(f(x)[\vec{\imath}] = \vec{a}) \le \eta/\varepsilon$. Hence, (i,q) is $(1-\eta)$-determined with probability $1-\eta/\varepsilon$, conditioned on $x[\vec{\imath}] = \vec{q}$ and $f(x)[\vec{\imath}] = \vec{a}$. By Definition 5.2, there is some a such that under this same conditioning (and additionally conditioning on $x[i] = q$) $f(x)[i] = a$ with probability at least $(1-\eta)$; by the definition of maj and the above, we have that $f(x)[i] = \mathrm{maj}(i,q)$ with probability at least

$$(1-\eta/\varepsilon)\,(1-\eta) \ge 1 - 2\eta/\varepsilon. \qquad \square$$

Intuitively, for any function $f : Q^n \to A^n$, either the value of the projection of $f(x)$ on a random coordinate can be guessed with high confidence by looking only at the same coordinate of x, or else, when one looks at the value of x at m random coordinates, errors build up, making it improbable to simultaneously guess correctly the projection of $f(x)$ on the respective m coordinates. The following lemma builds upon this intuition.

LEMMA 5.6. *For $m < n/2$, $n > 2^{10}\eta^{-4}\varepsilon^{-8}$, $\eta \le \frac{1}{2}$, and $m > 2^5\eta^{-2}\varepsilon^{-4}$, there exists a* good *block size k, $1 \le k \le m$, such that one of the following two conditions holds:*

1. *The probability that a random k-block $(\vec{\imath},\vec{q}) \overset{k}{\leftarrow} \pi^n$ is alive is at most ε.*
2. *If one chooses $(\vec{\imath},\vec{q})$ at random from the live k-blocks, $(\vec{\imath},\vec{q})$ will be $(1-\eta)$-good with probability at least $(1-\varepsilon)$. That is,*

$$\Pr_{(\vec{\imath},\vec{q}) \overset{k}{\leftarrow} \pi^n}\left[(\vec{\imath},\vec{q}) \text{ is } (1-\eta)\text{-good} \mid (\vec{\imath},\vec{q}) \text{ live}\right] \ge 1-\varepsilon.$$

Proof. For each j, $1 \le j \le n/2$, let E_j denote the following expectation:

$$E_j = \mathrm{E}_{(\vec{\imath},\vec{q})}\left[\sum_{\vec{a}\in A^j} \Pr\left[\vec{a}\right|(\vec{\imath},\vec{q})\left]^2\right],$$

where the expectation is taken over the choice of random j-block $(\vec{\imath},\vec{q})$. For every j, $0 < E_j \le 1$.

Let us give some insight on what E_j is measuring. Given $(\vec{\imath},\vec{q})$, the summation term measures the predictability of $\vec{a}$: if $\vec{a}$ is completely determined by $(\vec{\imath},\vec{q})$, the summation will be 1; if $\vec{a}$ is uniformly distributed with N possible values, the summation will be $1/N$. There are two competing considerations governing the relationship between E_j and E_{j+1}. Having more questions, and hence more answers, tends to fragment the set of possible answer vectors, causing E_j to be larger than E_{j+1}. However, seeing more questions may give so much information about the answers that the answer vectors are less fragmented. For example, if questions (q_1,q_2) are answered by (q_2,q_1), then E_1 may be quite small and $E_2 = 1$. We will show that, for our choice of parameters, if j is not a good block size, then the former consideration dominates; quantitatively, $E_j - E_{j+1} \ge 1/m$. Hence if there is no good block size between 1 and m, then $E_1 - E_{m+1} \ge 1$, a contradiction.

In order to lower bound $E_j - E_{j+1}$, we introduce a hybrid quantity:

$$E_j^* = \mathrm{E}_{(\vec{\imath},\vec{q})(i,q)}\left[\sum_{\vec{a}\in A^j} \Pr\left[\vec{a}\right|(\vec{\imath},\vec{q})\,(i,q)\left]^2\right],$$

where the expectation is taken over the choice of random j-block $(\vec{\imath},\vec{q})$ and random pair (i,q) $(i \notin \vec{\imath})$.

Let us give some more insight. E_j^* measures the average predictability of $\vec{a}$ given $(\vec{\imath},\vec{q})$ and an additional pair (i,q). Our proof proceeds as follows. First, we show that E_j^* is not much larger that E_j. The idea is that a random (i,q) is not likely to have much influence on $\vec{a}$, and hence knowing (i,q) will not make $\vec{a}$ that much more predictable on average. We then show that if j is not a good block size, then E_{j+1} is significantly smaller than E_j^*. That is, given $(\vec{\imath},\vec{q})$ and (i,q), $(\vec{a},a)$ is significantly harder to predict on average than just $\vec{a}$. Combining these relations between E_j, E_j^*, and E_{j+1}, we show that E_{j+1} is significantly smaller than E_j when $j \le n/2$ is not a good block size.

It follows from the above definitions that

$$E_j^* - E_j = \mathrm{E}_{(\vec{\imath},\vec{q})}\left[\mathrm{E}_{(i,q)}\left[\sum_{\vec{a}\in A^j} \Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\right]^2 - \Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)]^2\right]\right] \tag{5.1}$$

and

$$E_j^* - E_{j+1} = \mathrm{E}_{(\vec{\imath},\vec{q})(i,q)}\left[\sum_{\vec{a}\in A^j}\left(\Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\right]^2 - \sum_{a\in A}\Pr\left[\vec{a},a\right|\left(\vec{\imath},\vec{q}\right)(i,q)\right]^2\right)\right]. \tag{5.2}$$

PROPOSITION 5.7. *For any block size $j \le n/2$ (whether good or not),*

$$E_j^* - E_j \le \frac{2}{\sqrt{n-j}} \le \sqrt{\frac{8}{n}}$$

.

Proof. Fix a j-block $(\vec{\imath},\vec{q})$. The value of $\vec{a}$ is now a function of $n-j$ variables. Let ℓ denote the cardinality of A and consider the vector $v \in \mathcal{R}^{\ell^j}$ of probabilities $\Pr[\vec{a}|(\vec{\imath},\vec{q})]$. Then $\|v\|^2 = \sum_{\vec{a}\in A^j}(\Pr[\vec{a}|(\vec{\imath},\vec{q})])^2 \le 1$. Now fix also a pair (i,q) and consider the vector $u \in \mathcal{R}^{\ell^j}$ of probabilities $\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)]$. Then $\|u\|^2 = \sum_{\vec{a}\in A^j}\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)]^2 \le 1$, and by the triangle inequality, $\|u+v\| \le 2$. Moreover,

$$\begin{aligned}\sum_{\vec{a}\in A^j}\left(\Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\right]^2 - \Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)]^2\right) &= \|u\|^2 - \|v\|^2 = \langle (u+v),(u-v)\rangle \\ &\le \|u+v\|\cdot\|u-v\| \le 2\|u-v\|.\end{aligned}$$

By convexity and Corollary 4.6 (with $m=1$) we obtain

$$\mathrm{E}_{(i,q)}\left[\|u-v\|\right]^2 \le \mathrm{E}_{(i,q)}\left[\|u-v\|^2\right] \le \frac{1}{n-j}.$$

Combining this with the previous inequality yields

$$\mathrm{E}_{(i,q)}\left[\sum_{\vec{a}\in A^j}\left(\Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\right]^2 - \Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)]^2\right)\right] \le \frac{2}{\sqrt{n-j}}.$$

The proposition follows by taking expectations over $(\vec{\imath},\vec{q})$. ∎

PROPOSITION 5.8. *If block size $j \le n/2$ is not good, then $E_j^* - E_{j+1} \ge \eta^2\varepsilon^4/8$.*

Proof. Clearly, for any j-block $(\vec{\imath},\vec{q})$ and any pair (i,q) with $i \notin \vec{\imath}$, and any $\vec{a} \in A^j$,

$$\Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2 \;\ge\; \sum_{a\in A} \Pr\left[\vec{a},a\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2\right. . \tag{5.3}$$

When(i,q) is not $(1-\eta)$-determined by $(\vec{\imath},\vec{q})$, then strict inequality holds. We quantify the effect of these inequalities. First, we show that with a reasonably large probability, some useful events occur.

PROPOSITION 5.9. *Suppose that $j \le n/2$ is not a good block size. Let $(\vec{\imath},\vec{q})$ be a randomly chosen j-block and let (i,q) be a randomly chosen pair with $i \notin \vec{\imath}$. Then with probability at least $\eta\varepsilon^2/2$, there will exist some $\vec{a}$ such that*

1. *(For all $a \in A$)*$\Pr[\vec{a},a|(\vec{\imath},\vec{q})(i,q)] \le (1-\eta)\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)]$*, and*
2. $\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)] \ge \varepsilon/2$.

We defer the proof of Proposition 5.9 and proceed to analyzing its consequences. We first use a straightforward convexity argument to show that for $(\vec{\imath},\vec{q})$, (i,q), and $\vec{a}$ satisfying the conclusions of Proposition 5.9,

$$\sum_{a\in A} \Pr\left[\vec{a},a\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2\right. \le \left(1-2\eta+2\eta^2\right)\cdot \Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2\right. .$$

In general, if one wants to maximize $X_1^2+\cdots+X_k^2$ subject to $X = X_1+\cdots+X_k$, where all quantities are positive, it always helps to subtract from smaller values in order to add to larger values. Given the added constraint that $X_i \le (1-\eta)X$, the maximum is attained by setting $X_1 = (1-\eta)X$, $X_2 = \eta X$, and all the rest equal to 0, giving $X_1^2+\cdots+X_k^2 = (1-2\eta+2\eta^2)X^2$.

This implies (when $\eta \le 1/2$) that

$$\Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2 - \sum_{a\in A} \Pr\left[\vec{a},a\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2\right. \ge \eta\cdot \Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]^2\right. \ge \eta\cdot(\varepsilon/2)^2. \tag{5.4}$$

By (5.3), (5.1), and (5.4), each $((\vec{\imath},\vec{q}),(i,q))$ that satisfies the condition of Proposition 5.9 contributes at least $\eta(\varepsilon/2)^2 = \eta\varepsilon^2/4$ times the probability of the event occurring to $E_j^* - E_{j+1}$. Thus the good events contribute at least $(\eta\varepsilon^2/2)(\eta\varepsilon^2/4) = \eta^2\varepsilon^4/8$ to $E_j^* - E_{j+1}$. (Recall that all the other $((\vec{\imath},\vec{q}),(i,q))$ terms contribute nonnegatively, by (5.3).) □

Proof of Proposition 5.9. The statement of the proposition considers $\vec{a}$ such that (among other properties) $\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)] \ge \varepsilon/2$. We first show that if we modify this property, asking instead that $\Pr[\vec{a}|(\vec{\imath},\vec{q})] \ge \varepsilon$, then such $\vec{a}$ exist with sufficiently high probability. We then show it unlikely that an $\vec{a}$ exists that has the modified property but not the original property.

If j is not a good block size, then the probability that $(\vec{\imath},\vec{q})$ is alive is at least ε, and conditioned on being alive, the probability that $(\vec{\imath},\vec{q})$ is not good is again at least ε. Thereafter, the probability that $(\vec{\imath},\vec{q})$ does not $(1-\eta)$-determine a random (i,q) is at least η. Assume that all three of these events occur (this has probability at least $\eta\varepsilon^2$). Then because $(\vec{\imath},\vec{q})$ is alive and $(\vec{\imath},\vec{q})$ does not $(1-\eta)$-determine (i,q), there is some $\vec{a}$ such that $\Pr[\vec{a}|(\vec{\imath},\vec{q})] \ge \varepsilon$ and, for every $a \in A$,

$$\Pr\left[\vec{a},a\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left] \le (1-\eta)\Pr\left[\vec{a}\right|\left(\vec{\imath},\vec{q}\right)(i,q)\left]\right. .$$

It remains to bound the probability that for some $\vec{a}$, $\Pr[\vec{a}|(\vec{\imath},\vec{q})] \geq \varepsilon$ but $\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)] < \varepsilon/2$. We say that $((\vec{\imath},\vec{q}),(i,q))$ is *discardable* if, for some $\vec{a}$, $\Pr[\vec{a}|(\vec{\imath},\vec{q})] - \Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)] \geq \varepsilon/2$. Note that if $\Pr[\vec{a}|(\vec{\imath},\vec{q})] \geq \varepsilon$ and $((\vec{\imath},\vec{q}),(i,q))$ is not discardable, then $\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)] \geq \varepsilon/2$. We show that discardable $((\vec{\imath},\vec{q}),(i,q))$ occur with probability at most $\eta\varepsilon^2/2$; combined with the above bound, this implies the proposition.

Consider a live j-block $(\vec{\imath},\vec{q})$ and define a function $g : Q^{n-j} \to A^j$ that receives as an argument the value of an input x on the coordinates other than $\vec{\imath}$, and outputs the value of $f(x)$ on the coordinates indexed by $\vec{\imath}$. By Corollary 4.6,

$$\mathrm{E}_{(i,q)\leftarrow(Q,\pi)}\left[\sum_{\vec{a}\in A^j}\left(\Pr_{x\leftarrow(Q^n,\pi^n|(\vec{\imath},\vec{q}))}\left[g\left(x\right)=\vec{a}\right]-\Pr_{x\leftarrow(Q^n,\pi^n|(\vec{\imath},\vec{q})(i,q))}\left[g\left(x\right)=\vec{a}\right]\right)^2\right]$$
$$\leq 1/\left(n-j\right)\leq 2/n.$$

Using Markov's inequality and simple manipulations, it follows that the probability (over the choice of (i,q)) that there is any $\vec{a}$ with $\Pr[\vec{a}|(\vec{\imath},\vec{q})]-\Pr[\vec{a}|(\vec{\imath},\vec{q})(i,q)] \geq \varepsilon/2$ is at most $(2/n)/(\varepsilon/2)^2 = 8/\varepsilon^2 n$. For $n > 16/\eta\varepsilon^4$ this probability is at most $\eta\varepsilon^2/2$. □

Finally, we conclude the proof of Lemma 5.6. From Propositions 5.7 and 5.8 we obtain that $E_j - E_{j+1} \geq \eta^2\varepsilon^4/8 - \sqrt{\frac{8}{n}}$. For $j \leq m \leq n/2$ and $n > 2^{10}\eta^{-4}\varepsilon^{-8}$, $E_j - E_{j+1} \geq \eta^2\varepsilon^4/32$. For $m > 2^5\eta^{-2}\varepsilon^{-4}$, this difference is more than $1/m$ and the lemma follows. □

6. Proof of the main theorem. For notational convenience we write the initial miss-match game as G rather than G'. For ease of exposition, rather than consider G^n, we shall consider a related game that we denote by $\hat{G}$, which is derived from G^n by conditioning that there be exactly m match rounds. Hence $\hat{G}$ contains n parallel rounds of G, with the restriction of having exactly m match rounds and $n-m$ miss rounds, where $m < n/2$ will be determined later. The locations of the match rounds is random. Proposition 6.1 implies that upper bounding the error for $\hat{G}$ will suffice to prove our main theorem, though with a different value for c (that "swallows up" the difference between $4n$ and n).

PROPOSITION 6.1. *For $\hat{G}$ as described above, $\omega(\hat{G}) \geq \omega(G^{4n}) - 4e^{-n/32}$.*

Proof. We describe a probability measure on strategies $P = (p_1, p_2)$ for the provers in $\hat{G}$. Each strategy corresponds to a particular way in which the game $\hat{G}$ can be completed to a game G^{4n}. Let p_k denote the probability that G^{4n} has exactly k match rounds, conditioned on it having at least m match rounds and at least $n-m$ miss rounds (the latter condition is equivalent to having at most $3n+m$ match rounds).

We can view such a completion as the following random process:

1. The provers uniformly choose a set $S \subset [4n]$ indexing the n rounds into which they place the rounds of $\hat{G}$.
2. The provers choose $k \in [m, 3n+m]$ according to the probability measure $\Pr[k] = p_k$.
3. The provers uniformly choose a set $M \subseteq [4n] \setminus S$ of size $k-m$. For each round indexed by M the provers agree on the questions for match rounds, generated according to G.
4. For all the remaining rounds (not indexed by S or M), the provers agree on the questions for miss rounds, generated according to G.

The provers, prior to the start of the protocol, jointly sample one member P from the space of completion strategies.

When a prover receives a list of n questions in the game $\hat{G}$, the prover uses the agreed upon completion to embed these as part of a list of $4n$ questions in the game G^{4n}; then the prover employs the optimal strategy of the game G^{4n} and sends back only the n answers that correspond to the original questions of $\hat{G}$. It can readily be seen that the probability that V accepts is at least the probability that the optimal strategy for G^{4n} wins, given the distribution on the questions. This distribution is the "usual" distribution for G^{4n}, conditioned on there being at least m match rounds and $n-m$ miss rounds. By standard Chernoff bounds, the probability that there are less than $n-m$ miss rounds or less than m match rounds out of $4n$ rounds is at most $4e^{-n/32}$; this probability is negligible compared to ϵ when n is sufficiently large. Hence the expected probability of winning is at least $\omega(G^{4n}) - 4e^{-n/32}$, where expectation is taken over the random choice of completion strategy and random questions for $\hat{G}$.

Finally, by averaging, there is at least one deterministic strategy P that ensures probability of acceptance of at least $\omega(G^{4n}) - 4e^{-n/32}$ in the game $\hat{G}$. □

We now prove Theorem 3.3 by showing that $\omega(\hat{G}) \leq \epsilon$.

Proof. Fix a (deterministic) strategy (p_1, p_2) for $\hat{G}$ with highest probability of success for the two provers. This strategy is a pair of functions $p_1 : (X_1 \times X_2)^n \rightarrow (A_1 \times A_2)^n$ and $p_2 : (X_1 \bigcup X_2 \bigcup \{\lambda\})^n \rightarrow (A_1 \bigcup A_2)^m$. Observe that once p_1 is fixed, the function p_2 is characterized using the maximum likelihood principle. That is, on seeing the sequence $\vec{q}$ of n questions, it is best for P_2 to consider all possible sequences of n questions to P_1 that are consistent with P_2 receiving $\vec{q}$, compute the answers of P_1 on the match rounds for each of these sequences, and then answer the match rounds of $\vec{q}$ in a way identical to the most likely answer sequence by P_1 (breaking ties arbitrarily).

Let $Q = X_1 \times X_2 \times \{1,2\}$, let $A = A_1 \bigcup A_2$, and let π' be a probability distribution over Q that agrees with π over $X_1 \times X_2$ and is uniform over $\{1,2\}$ (that is, choose the first two coordinates according to π and then choose the last coordinate uniformly from $\{1,2\}$). Based on p_1 we define the function $f : Q^n \rightarrow A^n$, described as follows. For an input in Q^n, first strip off the last component of each coordinate, obtaining an input in $(X_1 \times X_2)^n$. Then compute the output of p_1 on the stripped input. This will be a vector in $(A_1 \times A_2)^n$. Finally, for each coordinate in this output vector, save only one of its two components (either the one in A_1 or the one in A_2), based on the respective value of the $\{1,2\}$ component of the original input vector. The function f corresponds to what P_2 would need to respond if all rounds were match rounds. In $\hat{G}$ some rounds are miss rounds, and our analysis will concentrate on coordinates of f that correspond to match rounds.

Recall the notion of a k-block from section 4. Assume that $m < n/2$. (We will enforce this condition in our choice of m.) Applying Lemma 5.6 with respect to f and probability distribution $(\pi')^n$, we have a *good* block of size $k \leq m$ in the sense defined in Lemma 5.6. (We shall expand on this below.) Fix this k for the rest of the proof.

For the purpose of our proof, we describe a three-step procedure by which the verifier selects the questions to the two provers in the n rounds. It can easily be verified that this three-step procedure induces the correct probability distribution on the questions.

Step 1. Select at random a k-block $(\vec{\imath}, \vec{q}) \stackrel{k}{\leftarrow} (\pi')^n$ for f. That is, select at random k distinct rounds, and in each such round select a question pair from $X_1 \times X_2$ with probability π (to be sent to P_1), and an index uniformly from $\{1,2\}$ specifying which half is sent to P_2. This specifies k match rounds.

Step 2. From the remaining rounds, select at random an $(m-k)$-block, B, specifying $m-k$ additional match rounds.

Step 3. Make the remaining $n-m$ rounds miss rounds. Select the questions to P_1 for these $n-m$ rounds. (P_2 receives λ on these rounds.)

To outline the rest of the analysis, we describe five events, at least one of which will occur when the verifier accepts. For each event, we bound the probability that the event occurs and that the verifier accepts. Although these events are not independent, we can conservatively bound the probability that the verifier accepts by the sum of the individual bounds.

Consider the k-block $(\vec{\imath}, \vec{q})$ selected by the verifier in Step 1, and consider the eventual answer $\vec{a}$ that P_2 gives on his respective half questions on the rounds specified by $\vec{\imath}$. Based on $(\vec{\imath}, \vec{q})$ and on function f, consider a new function $g : Q^{n-k} \to A^k$ that gives the value of f on the k coordinates specified by $\vec{\imath}$ as a function of the remaining $n-k$ coordinates of the input. For the verifier to accept, P_2 must give the same answers on these half questions. There are two cases:

1. The answer $\vec{a}$ of P_2 was an unlikely answer for P_1, in the sense that $\Pr_{x \leftarrow (\pi')^{n-k}}[g(x) = \vec{a}] < \varepsilon$, where $\varepsilon < \epsilon$ will be chosen later. There are two subcases to consider, depending on whether $\vec{a}$ continues to be an unlikely answer even after the $(m-k)$-block B is chosen.
   (a) $\Pr_{x \leftarrow ((\pi')^{n-k}|B)}[g(x) = \vec{a}] < 2\varepsilon$. This subcase is handled by Event 1.
   (b) $\Pr_{x \leftarrow ((\pi')^{n-k}|B)}[g(x) = \vec{a}] \geq 2\varepsilon$. This subcase is handled by Event 2.
2. $\Pr_{x \leftarrow (\pi')^{n-k}}[g(x) = \vec{a}] \geq \varepsilon$. This implies that the k-block $(\vec{\imath}, \vec{q})$ is alive (with respect to f). We have two subcases.
   (a) The k-block $(\vec{\imath}, \vec{q})$ is not $(1-\eta)$-good, where the precise value of η will be chosen later. This is handled by Event 3.
   (b) The k-block $(\vec{\imath}, \vec{q})$ is $(1-\eta)$-good. There are two subcases depending on the number of the remaining coordinates that are answered by P_1 according to the majority strategy. The subcase that this number is small is handled by Event 4, and the subcase that this number is large is handled by Event 5.

We now describe and analyze these events in greater detail.

Event 1. If $\Pr_{x \leftarrow ((\pi')^{n-k}|B)}[g(x) = \vec{a}] < 2\varepsilon$, we say that Event 1 occurs and assume that the verifier accepts with probability 2ε. This is indeed an upper bound on the acceptance probability in this case, because the verifier accepts only if P_1 and P_2 answer identically these k questions, and $\Pr[g(x) = \vec{a}]$ measures the probability (over the choices of the questions to P_1 in the miss rounds) that P_1 answers the same way as P_2 does. (Note that P_2's answer $\vec{a}$ is determined by $(\vec{\imath}, \vec{q})$ and B.) The contribution of Event 1 to the total acceptance probability is 2ε. (Note that we need not bound the probability of an event if we can bound the error probability conditioned on the event.)

Event 2. The outcome of Step 1 is a k-block $(\vec{\imath}, \vec{q})$. Event 2 happens when

1. $\Pr_{x \leftarrow (\pi')^{n-k}}[g(x) = \vec{a}] < \varepsilon$, and
2. $\Pr_{x \leftarrow ((\pi')^{n-k}|B)}[g(x) = \vec{a}] \geq 2\varepsilon$.

It follows that

$$\Pr_{x \leftarrow ((\pi')^{n-k}|B)}[g(x) = \vec{a}] - \Pr_{x \leftarrow (\pi')^{n-k}}[g(x) = \vec{a}] \geq \varepsilon.$$

We show that this happens with low probability, thus bounding the probability that Event 2 occurs and the verifier accepts. (Note that we do not need to consider the probability that the verifier accepts if we can bound the probability of the event.)

For function g, we apply Corollary 4.6 to bound the influence of Step 2 (in which an additional $(m-k)$-block is selected), obtaining

$$\mathrm{E}_{B \overset{m-k}{\leftarrow} (\pi')^{m-k}} \left[\sum_{\vec{a} \in A^k} \left(\Pr_{x \leftarrow (\pi')^{n-k}} [g(x) = \vec{a}] - \Pr_{x \leftarrow ((\pi')^{n-k} | B)} [g(x) = \vec{a}] \right)^2 \right] \leq (m-k)/(n-k) \leq m/n.$$

Now, if Event 2 occurs, then there is some $\vec{a}$ that contributes ε^2 to the above summation term. Note that all the other terms in the summation are nonnegative. By Markov's inequality, the probability of Event 2, taken over the choice of B, is at most $m/n\varepsilon^2$.

Event 3. Event 3 is the event that $(\vec{i}, \vec{q})$, selected in Step 1, is alive but not $(1-\eta)$-good. We upper bound the probability of Event 3 by ε as follows. Recall that k was selected to be good with respect to (f, π') in the sense of Lemma 5.6. Hence, either $(\vec{i}, \vec{q})$ is alive with probability at most ε or it is the case that live $(\vec{i}, \vec{q})$ fail to be $(1-\eta)$-good with probability at most ε. In either case, the probability that $(\vec{i}, \vec{q})$ is alive but not $(1-\eta)$-good is at most ε.

To define Events 4 and 5, recall the notion of *majority answer* from Definition 5.4. We introduced the notation $\mathrm{maj}(i, q)$ to denote the most likely value for $f(x)[i]$ when $x[i] = q \overset{\text{def}}{=} (q_1, q_2, b)$ (also conditioned on $x[\vec{i}] = \vec{q}$ and $f(x)[\vec{i}] = \vec{a}$). In our case, this corresponds to P_1's most likely bth half answer on coordinate i when $x[i] = q$. For $j \in \{1, 2\}$, we extend this notation to $\mathrm{maj}(i, q, j)$ to denote P_1's most likely jth half answer on coordinate i when $x[i] = q$. Note that j need not be equal to b.

Event 4. Event 4 occurs when

1. $(\vec{i}, \vec{q})$ is *alive* and *good*;
2. $\Pr_{x \leftarrow (\pi')^{n-k}}[g(x) = \vec{a}] \geq \varepsilon$; and
3. given $(\vec{i}, \vec{q}, \vec{a})$, less than a $1 - 2\eta/\delta$ fraction of the $2(n-k)$ remaining half answers of P_1 equal their respective majority answer, $\mathrm{maj}(i, q, j)$. Here, δ is a parameter that will be chosen later.

Note that for a fixed k-block $(\vec{i}, \vec{q})$, at most $1/\varepsilon$ different $\vec{a}$ may satisfy this condition. We shall concentrate on an arbitrary one such $\vec{a}$ and later multiply the probability of acceptance by a factor of $1/\varepsilon$.

The k-block $(\vec{i}, \vec{q})$ is good and the triple $(\vec{i}, \vec{q}, \vec{a})$ is alive. By Proposition 5.5, the following experiment succeeds with probability $\alpha \geq 1 - 2\eta/\varepsilon$.

1. Choose i uniformly from $[n] - \vec{i}$ and choose z according to $(\pi')^n$, conditioned on $z[\vec{i}] = \vec{q}$ and $f(z)[\vec{i}] = \vec{a}$.
2. Let $q = z[i]$. Succeed iff $f(z)[i] = \mathrm{maj}(i, q)$.

Recall that maj is defined with respect to $(\vec{i}, \vec{q}, \vec{a})$.

Let β denote the expected fraction of half answers given by P_1 that are equal to their respective $\mathrm{maj}(i, q, j)$, where the conditioning is as in the definition of Event 4. We will show that $\alpha = \beta$, and hence that $\beta \geq 1 - 2\eta/\varepsilon$. By Markov's inequality, the probability that P_1 gives the majority answer on less than a $1 - 2\eta/\delta$ fraction of the half questions is at most δ/ε. Summing over at most $1/\varepsilon$ possible $\vec{a}$, we have that Event 4 occurs with probability at most δ/ε^2.

It remains to show that $\alpha = \beta$. Given $z = (q_1^1, q_2^1, b^1), (q_1^2, q_2^2, b^2), \ldots$, define $z \backslash b = (q_1^1, q_2^1), (q_1^2, q_2^2), \ldots$ and $b(z) = b^1, b^2, \ldots$. Using the linearity of expectation we can express β as the success probability of the following experiment.

1. Choose i uniformly from $[n]-\vec{\imath}$ and choose z according to $(\pi')^n$, conditioned on $z[\vec{\imath}]=\vec{q}$ and $f(z)[\vec{\imath}]=\vec{a}$.
2. Let $q=z[i]\stackrel{\text{def}}{=}(q_1,q_2,b)$. Choose j uniformly from $\{1,2\}$. Let $(a_1,a_2)=p_1(z\backslash b)[i]$. Succeed iff $a_j=\text{maj}(i,q,j)$.

By the definition of f, we can rewrite the second step in the experiment defining α to be the following:

2. Let $q=z[i]\stackrel{\text{def}}{=}(q_1,q_2,b)$. Choose $j\in\{1,2\}$ as $j=b$. Let $(a_1,a_2)=p_1(z\backslash b)[i]$. Succeed iff $a_j=\text{maj}(i,q,j)$.

Thus, the only difference between the two experiments is in the distribution on j. The experiment for β chooses j uniformly, independently of all other events, whereas the experiment for α chooses $j=b$. We now use the fact that b is not an input to P_1, and hence the answers of P_1 are independent of b.

We can imagine choosing z according to $(\pi')^n$, then conditioning on $(\vec{\imath},\vec{q})$ and then conditioning on $f(z)[\vec{\imath}]=\vec{a}$. Initially, $b(z)$ is uniformly distributed, conditioned on $z\backslash b$. After conditioning on $\vec{q}$, $b(z)[\vec{\imath}]$ is fixed, but $b(z)[[n]-\vec{\imath}]$ remains uniformly distributed over $\{0,1\}^{n-k}$. Now, $f(z)[\vec{q}]$ is completely independent of $b(z)[[n]-\vec{\imath}]$, so further conditioning based on this value has no effect on the distribution of $b(z)[[n]-\vec{\imath}]$. Hence b is uniformly distributed and so is j in the experiment for α.

We established that j is identically distributed in the two experiments. To complete the proof of $\alpha=\beta$ we observe that the correlation between j and b in the experiment defining α is irrelevant to its probability of success, because neither (a_1,a_2) nor $(\text{maj}(i,q,1),\text{maj}(i,q,2))$ depends on b.

Event 5. Event 5 occurs when

1. $(\vec{\imath},\vec{q})$ is *alive* and *good*;
2. $\Pr_{x\leftarrow(\pi')^{n-k}}[g(x)=\vec{a}]\geq\varepsilon$; and
3. given $(\vec{\imath},\vec{q},\vec{a})$, at least a $1-2\eta/\delta$ fraction of the $2(n-k)$ remaining half answers of P_1 equal their respective majority answer, $\text{maj}(i,q,j)$.

It follows that for at least a $1-4\eta/\delta$ fraction of these coordinates, P_1 answers both half questions according to a majority strategy. Which majority strategy is used depends on $\vec{a}$; there are at most $1/\varepsilon$ values of $\vec{a}$ such that $\Pr_{x\leftarrow(Q^{n-k},(\pi')^{n-k})}[g(x)=\vec{a}]\geq\varepsilon$. Fixing $\vec{a}$ and assuming that P_1 plays the majority strategy for a $1-4\eta/\delta$ fraction of the $n-k$ subgames, consider the probability that verifier accepts on each of them. This probability is bounded above by the probability that the verifier accepts on a $1-4\eta/\delta$ fraction of the $n-k$ subgames, when P_1 plays the majority strategy for all $n-k$ subgames.

Fixing a strategy for P_1, whether the verifier accepts or rejects is a function of its random coins, which are chosen independently for each subgame. Since the majority strategy is a projection strategy, for each subgame the probability that the verifier accepts is at most p, independent of any of the other games. We can therefore apply a Chernoff bound (Theorem A.4 in [1]) and conclude that the probability that the verifier accepts in more than a $(1+p)/2$ fraction of the $n-k$ coordinates is at most $e^{-(1-p)^2(n-k)/2}<e^{-(1-p)^2n/4}$. When $\eta<(1-p)\delta/8$, then $(1+p)/2<1-4\eta/\delta$, and Event 5 occurs with probability at most $e^{-(1-p)^2n/4}/\varepsilon$.

Summing up the probabilities for the five events, the verifier accepts with probability at most

$$2\varepsilon+m/n\varepsilon^2+\varepsilon+\delta/\varepsilon^2+e^{-(1-p)^2n/4}/\varepsilon.$$

The various parameters need to satisfy $m<n/2$, $\eta<\frac{1}{2}$, $n>2^{10}\eta^{-4}\varepsilon^{-8}$, and $m>2^5\eta^{-2}\varepsilon^{-4}$ from Lemma 5.6 and $\eta<(1-p)\delta/8$ from Event 5. If $n\geq c/((1-p)\epsilon)^c$,

for a sufficiently large c, then the other parameters can be chosen to make the above expression at most ϵ, as desired. Needless to say, our analysis is not tight. □

Remark. The main difference between the proof of the above theorem in the current version and the preliminary version [12] is in the definition of the function f. In the miss-match case, the function f is based on the strategy of prover P_1, and its definition is a bit complicated. In the confuse-or-compare case [12], the function f was simply the strategy p_2 of prover P_2. The simpler definition for f allowed subsequent arguments to be stated more simply.

Acknowledgments. We thank Leonid Gurvits for his help in proving Theorem 4.3. We thank the referees for numerous useful comments.

REFERENCES

[1] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley, New York, 1992.
[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.
[3] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistic checkable proofs and applications to approximation*, in Proceedings of the 25th Annual ACM Symposium on the Theory of Computing, San Diego, CA, 1993, pp. 294–304.
[4] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.
[5] M. Bellare and M. Sudan, *Improved non-approximability results*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Montreal, Canada, 1994, pp. 184–193.
[6] M. Ben-Or, S. Goldwasser, J. Kilian, and A. Wigderson, *Multi prover interactive proofs: How to remove intractability assumptions*, in Proceedings of the 20th Annual ACM Symposium on the Theory of Computing, Chicago, IL, 1988, pp. 113–131.
[7] J. Cai, A. Condon, and R. Lipton, *On bounded round multi-prover interactive proof systems*, in Proceedings of the 5th IEEE Symposium on Structure in Complexity Theory, Barcelona, Spain, 1990, pp. 45–54.
[8] J. Cai, A. Condon, and R. Lipton, *Playing games of incomplete information*, Theoret. Comput. Sci., 103 (1992), pp. 25–38.
[9] J. Cai, A. Condon, and R. Lipton, *PSPACE is provable by two provers in one round*, J. Comput. System Sci., 48 (1994), pp. 183–193.
[10] U. Feige, *On the success probability of the two provers in one round proof systems*, in Proceedings of the 6th IEEE Symposium on Structure in Complexity Theory, Chicago, IL, 1991, pp. 116–123.
[11] U. Feige, *Error Reduction by Parallel Repetition—the State of the Art*, Technical report CS95-32, Weizmann Institute, Rehovot, Israel, 1995.
[12] U. Feige and J. Kilian, *Two prover protocols—low error at affordable rates*, in Proceedings of the 26th Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1994, pp. 172–183.
[13] U. Feige and L. Lovasz, *Two-prover one-round proof systems, their power and their problems*, in Proceedings of the 24th Annual ACM Symposium on the Theory of Computing, Victoria, Canada, 1992, pp. 733–744.
[14] U. Feige and O. Verbitsky, *Error reduction by parallel repetition—a negative example*, in Proceedings of the 11th Annual IEEE Conference on Computational Complexity, Philadelphia, PA, 1996, pp. 70–76.
[15] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, Theoret. Comput. Sci., 134 (1994), pp. 545–557.
[16] J. Hastad, *Clique is hard to approximate within* $n^{1-\epsilon}$, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 627–636.
[17] J. Kahn, G. Kalai, and N. Linial, *The influence of variables on Boolean functions*, in Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 68–80.
[18] D. Lapidot and A. Shamir, *A one-round, two-prover, zero-knowledge protocol for NP*, Combinatorica, 15 (1995), pp. 203–214.

[19] D. Lapidot and A. Shamir, *Fully parallelized multi prover protocols for NEXP-time*, J. Comput. System Sci., 54 (1997), pp. 215–220.
[20] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41(5) (1194), pp. 960–981.
[21] R. Raz, *A parallel repetition theorem*, SIAM J. Comput., 27 (1998), pp. 763–803.
[22] O. Verbitsky, *Towards the parallel repetition conjecture,* Theoret. Comput. Sci., 157 (1996), pp. 277–282.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 347–358

MESSAGE MULTICASTING IN HETEROGENEOUS NETWORKS*

AMOTZ BAR-NOY†, SUDIPTO GUHA‡, JOSEPH (SEFFI) NAOR§, AND BARUCH SCHIEBER¶

Abstract. In heterogeneous networks, sending messages may incur different delays on different links, and each node may have a different switching time between messages. The well-studied telephone model is obtained when all link delays and switching times are equal to one unit. We investigate the problem of finding the minimum time required to multicast a message from one source to a subset of the nodes of size k. The problem is NP-hard even in the basic telephone model. We present a polynomial-time algorithm that approximates the minimum multicast time within a factor of $O(\log k)$. Our algorithm improves on the best known approximation factor for the telephone model by a factor of $O(\frac{\log n}{\log \log k})$. No approximation algorithms were known for the general model considered in this paper.

Key words. approximation algorithms, multicast, postal model, LogP model, heterogeneous networks, combinatorial optimization

AMS subject classifications. 68Q25, 68R10, 05C85, 68W25, 90B18, 68M10

PII. S0097539798347906

1. Introduction.

The task of disseminating a message from a source node to the rest of the nodes in a communication network is called *broadcasting*. The goal is to complete the task as fast as possible assuming all nodes in the network participate in the effort. When the message needs to be disseminated only to a subset of the nodes, this task is referred to as *multicasting*. Broadcasting and multicasting are important and basic communication primitives in many multiprocessor systems. Current networks usually provide point-to-point communication only between some of the pairs of the nodes in the network. Yet, in many applications, a node in the network may wish to send a message to a subset of the nodes, where some of them are not connected to the sender directly. Due to the significance of this operation, it is important to design efficient algorithms for it.

Broadcast and multicast operations are frequently used in many applications for message-passing systems (see [10]). It is also provided as a communication primitive by several collective communication libraries, such as Express by Parasoft [7] and the message-passing library (MPL) [1, 2] of the IBM SP2 parallel systems. This operation is also included as part of the collective communication routines in the

*Received by the editors November 23, 1998; accepted for publication (in revised form) November 18, 1999; published electronically June 3, 2000. Part of this research was done while the first three authors visited IBM T.J. Watson Research Center. A preliminary version of this paper appeared in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 448–453.
http://www.siam.org/journals/sicomp/30-2/34790.html

†AT&T Research Labs, 180 Park Ave., P.O. Box 971, Florham Park, NJ 07932 (amotz@research.att.com). On leave from the Department of Electrical Engineering–Systems, Tel Aviv University, Tel Aviv 69978, Israel.

‡Computer Science Department, Stanford University, Stanford, CA 94305 (sudipto@cs.stanford.edu). This author was supported by an IBM fellowship, ARO MURI grant DAAH04-96-1-0007, and NSF award CCR-9357849, with matching funds from IBM, Schlumberger Foundation, Shell Foundation, and Xerox Corporation.

§Bell Laboratories, Lucent Technologies, 600 Mountain Ave., Murray Hill, NJ 07974 (naor@research.bell-labs.com). On leave from the Computer Science Department, Technion, Haifa 32000, Israel.

¶IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598 (sbar@watson.ibm.com).

message-passing interface (MPI) standard proposal [22]. Application domains that use broadcast and multicast operations extensively include scientific computations, network management protocols, database transactions, and multimedia applications. In most of these applications the efficiency depends on the time it takes to complete the broadcast or multicast operations.

There are two basic models in which trivial optimal solutions exist. In the first model, all nodes are assumed to be connected, a node may send a message to at most one other node in each round, and it takes one unit of time (round) for a message to cross a link. Therefore, in each round the number of nodes receiving the message can be doubled. If the target set of nodes is of size k, then this process terminates in $\lceil \log k \rceil$ rounds. In the second model the communication network is represented by an arbitrary graph, where each node is capable of sending a message to all of its neighbors in one unit of time. Here, the number of rounds required to deliver a message to a subset of the nodes is the maximum distance from the source node to any of the nodes in the subset.

The model in which a node may send a message to at most one other node in each round is known as the *telephone* model. It is known that for arbitrary communication graphs, the problem of finding an optimal broadcast in the telephone model is NP-hard [11], even for 3-regular planar graphs [19]. Following the two easy cases given above, it is not hard to verify that in the telephone model two trivial lower bounds hold for the minimum broadcast time. The first one is $\lceil \log n \rceil$, where n denotes the number of nodes in the graph, and the second one is the maximum distance from the source node to any of the other nodes. Research in the past three decades has focused on finding optimal broadcast algorithms for various classes of graphs such as trees, grids, and hypercubes. Also, researchers have looked for graphs with minimum number of links for which a broadcast time of $\lceil \log n \rceil$ can be achieved from any source node. Problems related to broadcast which were extensively investigated are the problems of broadcast multiple messages, gossiping, and computing certain functions on all n inputs in a network. See, e.g., [4, 5, 6, 8, 12, 13, 14, 18, 21, 24, 25].

An assumption central to the telephone model is that both sender and receiver are busy during the whole sending process. That is, only after the receiver received the message, both ends may send the message to other nodes. More realistic models in this context are the postal model [3] and the LogP model [17]. The idea there is that the sender may send another message before the current message is completely received by the receiver, and the receiver is free during the early stages of the sending process. We note that in both the postal model and the LogP model it is assumed that the delay of a message between any pair of nodes is the same.

Optimal solutions for broadcast in the postal model are known for the case of a complete graph and for some other classes of graphs. However, not much is known for arbitrary graphs. In the postal model, researchers have also concentrated on other dissemination primitives and almost always assumed that the communication graph is complete.

1.1. Our results. In this paper we define a more general model based on the postal model and call it the *heterogeneous postal* model. Assume node u sends a message to node v at time 0 and the message arrives at v at time λ_{uv}. The assumption is that u is free to send a new message at time s_u, and v is free from time 0 to time $\lambda_{uv} - r_v$. We call λ_{uv} the *delay* of the link (u, v), s_u the *sending* (or *switching*) time of u, and r_v the *receiving* time of v. By definition, both s_u and r_v are smaller than λ_{uv}. In the single-message multicast problem each node receives no more than a single

message. Thus, for this problem the receiving time has almost no relevance. Because of this, and to keep the presentation clearer, we assume for the rest of the paper that $r_u = s_u$ for all nodes u. Observe that when the delay, sending time, and receiving time are all equal to 1, we obtain the telephone model.

We believe that our framework may be useful to model modern communication networks, where the major components—the processors and the communication links—are not homogeneous. Some processors are faster than others, and some links have more bandwidth than others. These disparities are captured by the different values of the delay and the switching time.

Since finding the minimum multicast time is NP-hard even in the telephone model, we turn our focus to approximation algorithms. The main result we present is an approximation algorithm for computing a multicast scheme in the heterogeneous postal model. The approximation factor is $O(\log k)$, where k denotes the number of processors in the target set. Previous approximation algorithms for multicasting were known only in the telephone model. Kortsarz and Peleg [16] gave an approximation algorithm that produces a solution whose value is bounded away from the optimal solution by an $O(\sqrt{n})$ additive term. This term is quite large, especially for graphs in which the broadcast (multicast) time is polylogarithmic. Later, Ravi [20] gave an algorithm that achieves a multiplicative approximation factor of $O(\frac{\log n \log k}{\log \log k})$.

We also show that it is NP-hard to approximate the minimum broadcast time within a factor of three in a model which is only slightly more complicated than the telephone model.

The rest of the paper is organized as follows. In section 2 we define our model. In section 3 we describe our solution. Finally, in section 4 we show that this problem is hard to approximate by a small constant factor.

2. The model and the problem. We define our model as follows. Let $G = (V, E)$ be an undirected graph representing a communication network, where V is a set of n nodes and E is the set of point-to-point communication links. Let $U \subseteq V$ denote a special set of *terminals*, and let r be a special node termed the *root*. Let the cardinality of the set U be k. To simplify notation assume that $r \in U$.

We associate with each node $v \in V$ a parameter s_v that denotes the sending time. We sometimes refer to s_v as the switching time of v to indicate that this is the time it takes node v to send a new message. In other words, $1/s_v$ is the number of messages node v can send in one round (unit of time). We associate with each node $v \in V$ a parameter r_v that denotes the receiving time. We assume that $r_v = s_v$, for each node v. We associate with each link $(u, v) \in E$ a length λ_{uv} that denotes the communication delay between nodes u and v. By definition, λ_{uv} is greater than both s_u and r_v $(= s_v)$. We can think of the delay λ_{uv} as taking into account the sending time at u and the receiving time at v.

Let the *generalized degree* of node $v \in V$ be the actual degree of v in the graph G multiplied by the switching time s_v. Observe that the generalized degree measures the time it would have taken the node v to send a message to all of its neighbors.

Our goal is to find a minimum-time multicast scheme, that is, a scheme in which the time it takes for all nodes in the set U to receive the message from the root r is minimized. Without loss of generality, we may consider only multicast schemes that are "not lazy," i.e., schemes in which a node that has not finished sending the message to its neighbors (but has already started) is not idle. Such multicast schemes can be represented by an outward directed tree T that is rooted at r and spans all the nodes in U, together with orderings on the edges outgoing from each node in the tree. The

multicast scheme corresponding to such a tree and orderings is a multicast in which each node in the tree upon receiving the message (through its single incoming edge) sends the message along each of its outgoing edges in the specified order. From now on, we refer to the tree in the representation of a multicast scheme as the tree "used" by the scheme.

For a rooted tree T, denote by Δ_T its maximum generalized degree, and by L_T the maximum distance from r to any of the nodes in U (with respect to the lengths λ_{xy} associated with each link (x, y)). By definition, the multicast time of tree T is greater than Δ_T and greater than L_T. Hence, we obtain the following lemma.

LEMMA 1. *Let OPT denote the multicast time of an optimal solution using tree T^*; then $OPT \geq \frac{1}{2}(\Delta_{T^*} + L_{T^*})$*

3. The approximation algorithm. In this section we describe the approximation algorithm for multicasting a message to a set of terminals U from a root node $r \in U$.

The main tool used by our algorithm is a procedure $ComputeCore(U')$ that computes for a given set of terminals U', where $r \in U'$,

(1) a subset $W \subset U'$ which we call the *core* of U', of size at most $\frac{3}{4}|U'|$, where $r \in W$;
(2) a scheme to disseminate a message known to all the nodes in W to the rest of the nodes in U' in time proportional to the minimum multicast time from r to U'.

The algorithm that computes the multicast scheme proceeds in ℓ phases. Let $U_0 = U$. Upon termination, $U_\ell = \{r\}$. In the ith phase, $i = 1, \ldots, \ell$, procedure $ComputeCore(U_{i-1})$ is invoked to compute

(1) the core of U_{i-1}, denoted by U_i;
(2) a scheme to disseminate the message from U_i to the set U_{i-1} in time proportional to the minimum multicast time from r to U_{i-1}.

Since $|U_i| \leq \frac{3}{4} \cdot |U_{i-1}|$ and $|U_0| = k$, we have that $\ell = O(\log k)$. The resulting multicast scheme is given by looking at the rounds of the algorithm in backward order. Namely, starting at $i = \ell$ downwards, in each round of the multicast scheme the message is disseminated from U_i to U_{i-1}. Since $U_\ell \subset U_{\ell-1} \subset \cdots \subset U_0 = U$, each dissemination phase takes time proportional to the minimum multicast time from r to U. It follows that the multicast time is up to $O(\log k)$ times the optimal multicast time.

In the rest of the section we describe the procedure $ComputeCore(U')$. Let OPT be the minimum multicast time from r to U'. Lemma 1 implies that there exists a tree T^* spanning the set U' such that $\Delta_{T^*} + L_{T^*} \leq 2 \cdot OPT$. The procedure $ComputeCore(U')$ has two main parts. In the first part, we find a set of $|U'|$ paths, one for each terminal, where the ith path connects the terminal u_i to another terminal called $Mate(u_i)$. The paths have the following *path properties*:

Length property: The length of each path is at most $4 \cdot (\Delta_{T^*} + L_{T^*})$.

Congestion property: The generalized degree of the nodes in the graph induced by the paths is at most $6 \cdot (\Delta_{T^*} + L_{T^*})$.

In the second part we design a dissemination scheme using the above paths. We do it by transforming the paths into a set of disjoint *spider* graphs—graphs in which at most one node has degree more than two. These spider graphs have the following *spider properties*:

- Each spider contains at least two terminals from U'.
- The set of spiders spans at least half the nodes in U'.
- The diameter of each spider is at most $4 \cdot (\Delta_{T^*} + L_{T^*})$.

- The generalized degree of the *center* of a spider is at most $6 \cdot (\Delta_{T^*} + L_{T^*})$, where the center of a spider is the unique node with degree larger than two, if such exists, or one of the endpoints of the spider otherwise.

Now, for each spider, we arbitrarily select one of the nodes from U' to the core of U'. Note that each such node can multicast the message to the rest of the terminals in its spider in $O(\Delta_{T^*} + L_{T^*})$ time (linear in OPT). We add all the terminals not contained in any of the spiders to the core of U'. We claim that the size of the core is at most $\frac{3}{4}|U'|$. To see this, let x denote the number of spiders and let y be the number of the terminals in all the spiders. It follows that the size of the core is $|U'| - y + x$. By the first spider property we have that $x \leq y/2$ and by the second spider property we get that $y \geq |U'|/2$. Thus,

$$|core(U')| = |U'| - y + x \leq |U'| - \frac{y}{2} \leq |U'| - \frac{1}{4}|U'| = \frac{3}{4}|U'| \ .$$

We now turn to describe each of the two parts of the procedure $ComputeCore(U')$.

3.1. Finding a set of paths. We first claim the following lemma, which is analogous to the "tree-pairing" lemma of Ravi [20].

LEMMA 2. *Let T be a tree that spans a set $U' \subseteq V$, and suppose that $|U'|$ is even. There exists a way to pair the nodes of U' and to find paths (in the tree T) connecting each pair such that*

(1) *the paths are edge disjoint;*
(2) *the length of each path is bounded by $2L_T$;*
(3) *the generalized degree of each node in the graph induced by the paths is at most Δ_T.*

Proof. The tree-pairing lemma [20] states that there exists a pairing such that the paths in T connecting each of the pairs are edge disjoint. Consider these paths. Clearly the length of each of these paths is bounded by $2L_T$. The degree, and hence the generalized degree, of every node in the graph induced by the paths is no more than the (generalized) degree in T since we use only the edges of the tree T. Hence, it is bounded by Δ_T. ∎

The following corollary handles the odd case as well.

COROLLARY 3. *Let T be a tree that spans a set $U' \subseteq V$. There exists a way to pair the nodes of U' and to find paths (in the tree T) connecting each pair such that*

(1) *the length of each path is bounded by $2L_T$;*
(2) *the generalized degree of each node in the graph induced by the paths is at most $2\Delta_T$.*

Proof. The corollary clearly holds if $|U'|$ is even. If $|U'|$ is odd, we pair $|U'| - 1$ of the nodes as in Lemma 2 and pair the last node with any other node. The length of the path connecting the last pair is still bounded by $2L_T$. However, the degree of the subgraph may double up to $2\Delta_T$. ∎

Recall that the tree T^* spans the nodes of U' and $(\Delta_{T*} + L_{T*}) \leq 2 \cdot OPT$. Our objective is to find the set of paths as guaranteed by Corollary 3 with respect to T^*. However, we do not know T^*. Thus, instead, we find a set of *fractional* paths satisfying similar properties. To this end, we write a linear program for finding a set of (fractional) paths that minimizes the sum of two quantities: (1) the maximum over all pairs of the average length of the paths connecting a pair, and (2) the maximum generalized degree of the subgraph induced by the paths connecting the pairs.

The linear program is a variant of multicommodity flow. For each edge (u, v), we define the directed edges (u, v) and (v, u) both of length λ_{uv}. Let $U' = \{v_1, \ldots, v_h\}$.

With each node $v_j \in U'$ we associate commodity j. Node v_j is the source of commodity j and we create an artificial sink t_j with $r_{t_j} = s_{t_j} = 0$. We connect each of the nodes $v' \in U'$, where $v_j \neq v'$, to t_j by a directed edge (v', t_j) of length 0. The objective is to minimize $(L + \Delta)$, where exactly one unit of flow has to be shipped from each v_j to t_j, such that the average length of the flow paths from v_j to t_j is at most $2L$, and the maximum weighted congestion (generalized degree) of the induced subgraph is at most 3Δ.

More formally, let A denote the set of directed edges, and let $f^i(u, v)$ denote the flow of commodity i on directed edge (u, v). The linear programming formulation is as follows.

Minimize $\Delta + L$

subject to:

For all $1 \leq i \leq h$ and $v \in V - \{v_i, t_i\}$:	$\sum_{(u,v)\in A} f^i(u,v) - \sum_{(v,w)\in A} f^i(v,w)$	$=$	$0.$
For all $1 \leq i \leq h$:	$\sum_{(u,t_i)\in A} f^i(u,t_i)$	$=$	$1.$
For all $1 \leq i \leq h$:	$\sum_{(v_i,u)\in A} f^i(v_i,u)$	$=$	$1.$
For all $1 \leq i \leq h$:	$\sum_{(u,v)\in A} f^i(u,v)\lambda_{uv}$	$\leq$	$2L.$
For all $v \in V - \{t_1, \ldots, t_h\}$:	$s_v \cdot \sum_i \sum_{u\in V} \left(f^i(u,v) + f^i(v,u)\right)$	$\leq$	$3\Delta.$
For all $1 \leq i \leq h$:	$f^i(u,v)$	$\geq$	$0.$

We now show that the set of paths guaranteed by Corollary 3 with respect to T^* can be modified so as to obtain an integral solution for the linear program as follows. If $|U'|$ is even, the solution is obtained by using each path connecting a pair (u_i, u_j) to ship one unit of flow from u_j through u_i to t_j, and another unit of flow from u_i through u_j to t_i. The length of each path is bounded by $2L_{T^*}$, and since we use each path twice, the generalized degree is bounded by $2\Delta_{T^*}$. If $|U'|$ is odd, the solution is obtained by using each of the $\frac{1}{2}(|U'| - 1)$ paths connecting the first $|U'| - 1$ nodes of U' twice (once in each direction), and using the path connecting the last node in U' to its mate to ship flow out of this node. The length of each path is still bounded by $2L_{T^*}$. However, because of the additional path, the degree is only bounded by $3\Delta_{T^*}$.

It follows that the value of the objective function for this solution is $\Delta_{T^*} + L_{T^*}$, and thus the linear program is guaranteed to find a solution whose value is upper bounded by this value. Let L_T and Δ_T denote the values of the length and congestion in the optimal solution of the above linear program.

The optimal solution is a "fractional" solution in the sense that the (unit) flow of each commodity is split between several flow paths. We round the fractional solution into an integral solution using an algorithm proposed by Srinivasan and Teo [23]. This algorithm builds on a theorem proved by Karp et al. [15]. For completeness, and

since the details are slightly different, we now describe the rounding of the fractional solution.

THEOREM 4 (see [15]). *Let A be a real valued $r \times s$ matrix, and y be a real valued s-vector. Let b be a real valued vector such that $Ay = b$. Let t be a positive real number such that in every column of A,*

(1) *the sum of all positive entries $\leq t$; and*

(2) *the sum of all negative entries $\geq -t$.*

Then, we can compute (in polynomial time) an integral vector $\bar{y}$ such that for every i, either $\bar{y}_i = \lfloor y \rfloor$ or $\bar{y}_i = \lceil y \rceil$ and $A\bar{y} = \bar{b}$, where $\bar{b}_i - b_i < t$.

We now show how to find an integral flow of congestion at most $6\Delta_T + 4L_T$, where each flow path (of each commodity) has length at most $4L_T$. We first decompose the flow into (polynomially many) flow paths. If any path in this decomposition is longer than $4L_T$, we discard it. We observe that since the average length is less than $2L_T$, discarding these long paths leaves at least half of a unit of flow between each pair (v_i, t_i). We scale the flows appropriately such that the total flow to each t_i is exactly 1. This can at most double the flow on an edge, and the total congestion is now at most $6\Delta_T$.

Let $P_1, P_2, \ldots$ denote the length-bounded flow paths. Denote the set of nodes in a path P_i by $V(P_i)$ and the set of edges by $E(P_i)$. Let $f(P_i)$ denote the amount of flow pushed on path P_i. Define the set $\mathcal{P}^j$ as the set of all paths that carry flow of the jth commodity. Observe that each path belongs to exactly one $\mathcal{P}^j$. The linear system $Ay = b$ needed for Theorem 4 is defined by the following linear equations, where the ith equation corresponds to the ith row of A and the ith element of b.

$$\begin{array}{lrcl} \text{For each } v & s_v \cdot \displaystyle\sum_{i:\, v \in V(P_i)} f(P_i) & \leq & 6\Delta_T; \\ \text{for all } j & -4L_T \cdot \displaystyle\sum_{i:\, P_i \in \mathcal{P}^j} f(P_i) & = & -4L_T. \end{array}$$

The second set of inequalities captures the fact that the flow on all the paths corresponding to commodity j is exactly 1. Now the sum of the positive entries in a column is

$$\sum_{v \in V(P_i)} s_v \leq \sum_{(v,w) \in E(p_i)} \lambda_{vw} + s_{t_j} = (\text{length of path } P_j) \leq 4L_T \ .$$

The second part of the inequality follows since $s_v \leq \lambda_{vw}$ for all v, w and $s_{t_j} = 0$. The sum of the negative entries in a column is at most $4L_T$; this follows due to the fact that each P_i belongs to exactly one $\mathcal{P}^j$. Invoking Theorem 4 gives us a set of paths such that

$$\begin{array}{lrcl} \text{for each } v & s_v \cdot \displaystyle\sum_{i:\, v \in V(P_i)} \bar{f}(P_i) & < & 6\Delta_T + 4L_T; \\ \text{for all } j & -4L_T \cdot \displaystyle\sum_{i:\, P_i \in \mathcal{P}^j} \bar{f}(P_i) & < & 0. \end{array}$$

The second set of inequalities implies that each commodity has at least one flow path. So we have a set of flow paths such that the congestion is at most $6\Delta_T + 4L_T$ and their length is at most $4L_T$. Since $\Delta_T + L_T \leq \Delta_{T^*} + L_{T^*}$ these paths satisfy the length and congestion properties as desired.

3.2. Finding a spider decomposition. We now show how to obtain a spider decomposition satisfying the spider properties previously defined. Recall that we are now given a set of paths connecting each terminal u_j with another terminal $Mate(u_j)$, and that this set of paths satisfies the length and congestion properties.

We find a set of at least $|U'|/2$ *trees* that satisfy the following properties which are similar to the spider properties.

- Each tree spans at least two terminals from U'.
- The diameter of each tree is at most $4L_T \leq 4 \cdot (\Delta_{T^*} + L_{T^*})$.
- The generalized degree of each node in each of the trees is at most $6\Delta_T + 4L_T \leq 6(\Delta_{T^*} + L_{T^*})$.

Before showing how to find these trees, we show how to transform them into the required spiders. Repeatedly, consider the tree edges, and remove a tree edge if it separates the tree into two subtrees such that either both subtrees contain at least two terminals or one of them contains no terminals. (In this case this subtree is removed as well.) Repeat this process until no more edges can be removed. The process terminates since the number of edges is finite. Observe that upon termination if a connected component is not a spider, then another edge could be deleted. Thus, we get the following claim.

Claim 5. *When the process terminates, each connected component is a spider.*

Clearly, all the terminals spanned by the trees are also spanned by the spiders. The diameter of each of these spiders is at most $4L_T$, since the distance between every pair of nodes in U' spanned by a tree is at most $4L_T$ to begin with. Also, the generalized degree of the "center" of the spider is at most the generalized degree of its originating tree since we have not introduced any new edges in the process. We conclude that the spiders satisfy the desired spider properties.

Now, we show how to find the required trees. Define G_p to be the undirected graph induced by the paths from each terminal to its mate. Observe that a spanning forest of this graph may not satisfy the required diameter property above, and hence some extra refinement is necessary.

For each node u in G_p, find a unique terminal in U' that is closest to u (with respect to the lengths λ_{xy} associated with each link (x, y)). Ties are broken arbitrarily.

We modify the paths starting at each terminal as follows. From each terminal u begin tracing the path connecting u to $Mate(u)$. At some node v along this path, the closest terminal to v will not be u. We are guaranteed to encounter such a node because the closest node to $Mate(u)$ is $Mate(u)$ itself. From this node v trace the path to its closest terminal. This creates a path from u to another terminal denoted $NewMate(u)$. Note that $NewMate(u)$ may be different from $Mate(u)$. However, we are guaranteed that the path from u to $NewMate(u)$ is not longer than the path from u to $Mate(u)$ and thus bounded by $4L_T$.

Define an auxiliary directed graph H on the set of terminals U' with the set of edges $(u \rightarrow NewMate(u))$ for $u \in U'$. By definition each node in H has outdegree one. Thus, each connected component of (the undirected skeleton of) H contains exactly one directed cycle. Discard one edge from each such connected component to make it a rooted tree in which all edges are oriented towards the root. (The root is unique since the outdegree of each node is one.) Note that every nontrivial strongly connected component of H is a cycle. Thus, this can be done just by discarding an arbitrary edge from each strongly connected component of H. Let H' be the resulting forest.

Define the level of a node in H' to be the number of edges in the path to it from the root of its component. (We flip the direction of the edges in H' for the purpose

of measuring distances.) Distinguish between nodes of even level and nodes of odd level. Each edge of H' goes either from an odd-level node to an even-level node or vice-versa.

Consider two collections of stars in H'. One collection consisting of edges from odd-level nodes to even-level nodes, and the other consisting of edges from even-level nodes to odd-level nodes. Every terminal with positive indegree and outdegree (in H') is spanned by a star in each one of the two collections. Every terminal with either indegree or outdegree zero (in H') is spanned by a star in only one of the two collections. However, by a simple pigeonhole argument, at least one of the collections spans at least half of the terminals.

Consider such a collection. First, note that each star in this collection induces an undirected tree in the original graph when replacing each star edge by its originating path. We now claim the following.

LEMMA 6. *The induced trees of any two stars belonging to the same collection are node disjoint.*

Proof. To obtain a contradiction assume they are not disjoint. Then there exist two distinct terminals with the same even or odd parity, say, u and v, such that $NewMate(u) \neq NewMate(v)$, but the paths traced from u to $NewMate(u)$ and from v to $NewMate(v)$ have a common node x. Consider the terminal chosen by x as its closest terminal. We distinguish between two cases.

Case 1. The terminal chosen by x is u. Then u must be $NewMate(v)$, contradicting the fact that u and v are of the same parity. The case where v is the chosen terminal of x is symmetric.

Case 2. The terminal chosen by x is $NewMate(u)$. Then $NewMate(v)$ must be the same as $NewMate(u)$, a contradiction. The case where $NewMate(v)$ is the chosen terminal of x is symmetric. □

It is easy to see that the trees induced by the stars in the collection satisfy the required properties. This concludes the construction.

4. Hardness of approximations. In this section we show that the best possible approximation factor of the minimum broadcast time in the heterogeneous postal model is $3 - \epsilon$. We show this hardness result even for a restricted model in which $s_i \in \{0, 1\}$ and $\lambda_{uv} \in \{1, d\}$ for some constant d. Note that when $s_i = 0$, node u_i can broadcast the message concurrently to all of its neighbors. The proof is by a reduction to the set cover problem. In the unweighted version of the set cover problem we are given a set U of elements and a collection S of subsets of U. The goal is to find the smallest number of subsets from S whose union is the set U. Feige [9] proved the following hardness result.

LEMMA 7. *Unless* $NP \subseteq DTIME(n^{\log \log n})$, *the set cover problem cannot be approximated by a factor which is better than* $\ln n$, *and hence it cannot be approximated within any constant factor.*

In our proof, we will only use the fact that it is NP-hard to approximate the optimal set cover within any constant factor.

THEOREM 8. *It is NP-hard to approximate the minimum broadcast time of any graph within a factor of* $3 - \epsilon$.

Proof. Assume to the contrary that there exists an algorithm that violates the claim of the theorem for some ϵ. We show how to approximate the set cover problem within a constant factor using this algorithm.

To approximate the set cover problem we "guess" the size of the optimal set cover and use our approximate minimum broadcast time algorithm to check our guess. Since

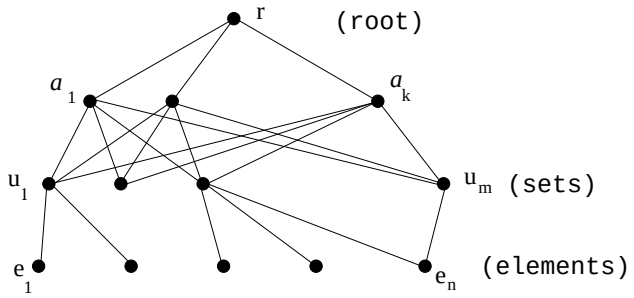


FIG. 1. *The graph G.*

the size of the optimal set cover is polynomial, we need to check only a polynomial number of guesses.

Consider an instance of set cover $I = (U, S)$, where U is the set of elements, and S a collection of subsets of U. Let $|U| = n$ and $|S| = m$. Let the guess on the size of the optimal set cover be k. We construct the following graph G (see Figure 1). The graph G consists of $1 + n + m + k$ vertices: a distinguished root vertex r, vertices $e_1, \ldots, e_n$ corresponding to the elements of U, vertices $u_1, \ldots, u_m$ corresponding to the subsets, and k additional vertices $a_1, \ldots, a_k$.

The root r has switching time $s(r) = 0$ and is connected to $a_1, \ldots, a_k$ by edges with delay $\lambda_{ra_\ell} = 1$. Each vertex a_ℓ has switching time $s(a_\ell) = 1$ and is connected to all u_j with delay $\lambda_{a_\ell u_j} = 1$. Each vertex u_j has switching time $s(u_j) = 0$ and is connected to a vertex e_i iff the jth set contains the ith element. The delay of such an edge is $\lambda_{u_j e_i} = d$, where $d > \frac{4-2\epsilon}{\epsilon}$ is a constant. Each vertex e_i has switching time $s(e_i) = 1$. Finally, to complete the instance of the multicasting problem, the target multicast set consists of all vertices e_i.

We first show that if there is a set cover of size k, then there is a multicast scheme of length $d + 2$. After time 1, all the vertices a_ℓ receive the message. After time 2, all the vertices u_j corresponding to sets which are in this cover receive the message. This is possible since all a_ℓ are connected to all u_j. Finally, these vertices send the message to all the elements that they cover. Since $s(u_j) = 0$ and $\lambda_{u_j e_i} = d$, it follows that the multicast time is $d + 2$.

Suppose that the algorithm for the multicasting problem completes the multicasting at time t. By the contradiction assumption, its approximation factor is $3 - \epsilon$. Since by our guess on the size of the set cover the optimal multicast time no more than $d+2$, we have $t \leq (3-\epsilon)(d+2) = 3d+6-2\epsilon-\epsilon d < 3d+6-2\epsilon-4+2\epsilon = 3d+2$. The strict inequality follows from the choice of d.

We first claim that all the vertices u_j that participate in the multicast receive the message from some a_ℓ. Otherwise there exists a vertex $e_{i'}$ that received the message via a path of a type $(r, a_\ell, u_j, e_i, u_{j'}, e_{i'})$. This means that $e_{i'}$ received the message at or after time $3d + 2 > t$. Our second claim is that each vertex a_ℓ sends the message to at most $2d$ vertices u_j. This is because the $(2d + 1)$st vertex would receive the message at time $2d + 2$ and would not be able to help in the multicast effort that is completed before time $3d + 2$.

Combining our two claims we get that the multicasting was completed with the help of $2dk$ vertices u_j. The corresponding $2dk$ sets cover all the elements e_i. This violates Lemma 7, which states that the set cover problem cannot be approximated within any constant factor. □

Remark. In our proof we considered a restricted model in which the switching time may get only two possible values and the delay may get only three possible values (assuming that when an edge does not exist, then the delay is infinity). Observe that this hardness result does not apply to the telephone model, in which the switching time is always 1 and the delay is either 1 or infinity. We have similar hardness results for other special cases. However, none of them is better than 3 and all use similar techniques.

Acknowledgment. We thank David Williamson for his helpful comments.

REFERENCES

[1] V. Bala, J. Bruck, R. Bryant, R. Cypher, P. deJong, P. Elustondo, D. Frye, A. Ho, C. T. Ho, G. Irwin, S. Kipnis, R. Lawrence, and M. Snir, *The IBM external user interface for scalable parallel systems*, Parallel Comput., 20 (1994), pp. 445–462.

[2] V. Bala, J. Bruck, R. Cypher, P. Elustondo, A. Ho, C. T. Ho, S. Kipnis, and M. Snir, *CCL: A portable and tunable collective communication library for scalable parallel computers*, in Proceedings of the Eighth International Parallel Processing Symposium, Cancun, Mexico, IEEE Press, Piscataway, NJ, 1994, pp. 835–844.

[3] A. Bar-Noy and S. Kipnis, *Designing broadcasting algorithms in the Postal model for message-passing systems*, Math. Systems Theory, 27 (1994), pp. 431–452.

[4] A. Bar-Noy, S. Kipnis, and B. Schieber, *Optimal multiple message broadcasting in telephone-like communication systems*, in Proceedings of the Sixth Symposium on Parallel and Distributed Processing, Dallas, TX, IEEE Press, Piscataway, NJ, 1994, pp. 216–223.

[5] E. Cockayne and A. Thomason, *Optimal multi-message broadcasting in complete graphs*, in Proceedings of the Eleventh SE Conference on Combinatorics, Graph Theory, and Computing, 1980, pp. 181–199.

[6] M. J. Dinneen, M. R. Fellows, and V. Faber, *Algebraic construction of efficient networks*, Applied Algebra, Algebraic Algorithms, and Error Correcting Codes (AAECC), Lecture Notes in Comput. Sci. 539, Springer, Berlin, 1991, pp. 152–158.

[7] *Express User's Guide, Version* 3.0, Parasoft Corporation, Monrovia, CA, 1995.

[8] A. M. Farley, *Broadcast time in communication networks*, SIAM J. Appl. Math., 39 (1980), pp. 385–390.

[9] U. Feige, *A threshold of* $\ln n$ *for approximating set cover*, in Proceedings of the 28th Symposium on Theory of Computing (STOC), Association for Computing Machinery, New York, 1996, pp. 314–318.

[10] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, and D. Walker, *Solving Problems on Concurrent Processors, Volume* I : *General Techniques and Regular Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1988.

[11] M. R. Garey and D. S. Johnson, *Computers and Intractability*: *A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[12] L. Gargano and U. Vaccaro, *On the construction of minimal broadcast networks*, Networks, 19 (1989), pp. 373–389.

[13] M. Grigni and D. Peleg, *Tight bounds on minimum broadcast networks*, SIAM J. Discrete Math., 4 (1991), pp. 207–222.

[14] S. M. Hedetniemi, S. T. Hedetniemi, and A. L. Liestman, *A survey of gossiping and broadcasting in communication networks*, Networks, 18 (1988), pp. 319–349.

[15] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani, *Global wire routing in two-dimensional arrays*, Algorithmica, 2 (1987), pp. 113–129.

[16] G. Kortsarz and D. Peleg, *Approximation algorithm for minimum time broadcast*, SIAM J. Discrete Math., 8 (1995), pp. 401–427.

[17] R. Karp, A. Sahay, E. Santos, and K. E. Schauser, *Optimal broadcast and summation in the LogP model*, in Proceedings of the Fifth Symposium on Parallel Algorithms and Architectures (SPAA), Association for Computing Machinery, New York, 1993.

[18] A. L. Leistman and J. G. Peters, *Broadcast networks of bounded degree*, SIAM J. Discrete Math., 1 (1988), pp. 531–540.

[19] M. Middendorf, *Minimum broadcast time is NP-complete for* 3*-regular planar graphs and deadline* 2, Inform. Process. Lett., 46 (1993), pp. 281–287.

[20] R. Ravi, *Rapid rumor ramification: Approximating the minimum broadcasting time*, in Proceedings of the 35th Symposium on Foundations of Computer Science (FOCS), Santa Fe, NM, IEEE Press, Piscataway, NJ, 1994, pp. 202–213.

[21] D. Richards and A. L. Leisterman, *Generalizations of broadcasting and gossiping*, Networks, 18 (1988), pp. 125–138.

[22] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI—The Complete Reference, Vol.* 1*, The MPI*-1 *Core*, 2nd ed., MIT Press, Cambridge, MA, 1998.

[23] A. Srinivasan and C.-P. Teo, *A constant-factor approximation algorithm for packet routing, and balancing local vs. global criteria*, in Proceedings of the 29th Symposium on Theory of Computing (STOC), Association for Computing Machinery, New York, 1997, pp. 636–643.

[24] J. A. Ventura and X. Weng, *A new method for constructing minimal broadcast networks*, Networks, 23 (1993), pp. 481–497.

[25] D. B. West, *A class of solutions to the gossip problem, Part* I, Discrete Math., 39 (1992), pp. 307–326.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 359–382

COMPLEXITY OF SOME PROBLEMS CONCERNING VARIETIES AND QUASI-VARIETIES OF ALGEBRAS*

CLIFFORD BERGMAN† AND GIORA SLUTZKI‡

Abstract. In this paper we consider the complexity of several problems involving finite algebraic structures. Given finite algebras **A** and **B**, these problems ask the following. (1) Do **A** and **B** satisfy precisely the same identities? (2) Do they satisfy the same quasi-identities? (3) Do **A** and **B** have the same set of term operations?

In addition to the general case in which we allow arbitrary (finite) algebras, we consider each of these problems under the restrictions that all operations are unary and that **A** and **B** have cardinality two. We briefly discuss the relationship of these problems to algebraic specification theory.

Key words. variety, quasi-variety, clone, term-equivalence, computational complexity, logarithmic space, polynomial space, hyperexponential time, nondeterminism

AMS subject classifications. Primary, 68Q25; Secondary, 08B15, 08C15, 08A40, 68Q65

PII. S0097539798345944

1. Introduction.

There are several relationships between mathematical structures that might be considered "fundamental." First and foremost is certainly the isomorphism relation. Questions about isomorphic structures occur throughout mathematics and apply to universal algebras, topological spaces, graphs, partially ordered sets, etc. Many other relationships are more specialized. For example, given two graphs **G** and **H**, one may wish to know whether **H** is a subgraph of **G** or perhaps a minor of **G**.

Properly formulated, questions about these relationships give rise to complexity questions. Generally speaking, we must impose some sort of finiteness assumption on the structures in question so that notions of computational complexity make sense. The complexity of various isomorphism problems has received a great deal of attention. The graph isomorphism problem has been intensively studied, partly because its exact relationship to the classes **P** and **NP** is still unknown, and partly because it provides a paradigm for other problems of unknown complexity status. In this case, both graphs are assumed to have finitely many vertices and finitely many edges. With a similar formulation, the isomorphism problem for algebras has the same complexity as does graph isomorphism. More generally, Kozen [17] showed that the isomorphism problem for finitely presented algebras has this same complexity. See [4, 16, 19] for further discussion and references on the isomorphism problem.

In this paper we consider the complexity of three relationships that arise from considerations in universal algebra. Any algebraic structure satisfies certain identities and fails to satisfy others. Roughly speaking, an identity is an equality between two expressions built from the operations of the algebra. Examples of identities are the associative law (which involves one binary operation) and DeMorgan's law (two binary

*Received by the editors October 19, 1998; accepted for publication (in revised form) October 26, 1999; published electronically June 3, 2000. A preliminary version of this paper appeared in *The* 16th *Symposium on Theoretical Computer Science (STACS* '99*),* Lecture Notes in Comput. Sci. 1563, Springer-Verlag, Berlin, 1999, pp. 163–172.

http://www.siam.org/journals/sicomp/30-2/34594.html

†Department of Mathematics, Iowa State University, 400 Carver Hall, Ames, IA 50011-5454 (cbergman@iastate.edu).

‡Department of Computer Science, Iowa State University, 226 Atanasoff Hall, Ames, IA 50011-5454 (slutzki@cs.iastate.edu).

operations and one unary operation). Identities are one of the primary organizing tools in algebra.

Given two algebras $\mathbf{A}$ and $\mathbf{B}$, we may ask whether they satisfy precisely the same set of identities. Notice that this is a far weaker notion than isomorphism. For example, any algebra satisfies the same identities as each of its direct powers. Nevertheless, if $\mathbf{A}$ and $\mathbf{B}$ satisfy the same identities, then they will be constrained to behave in a similar way. One of our problems, called VAR-EQUIV, is this: Given two finite algebras of the same finite similarity type, determine whether they satisfy the same identities.

This problem has implications for several areas of computer science. *Formal algebraic specifications* are expressions in a language which describe the properties and input-output behavior that a software system must exhibit, without putting any restrictions on the way in which these properties are implemented. This *abstraction* makes formal specifications extremely useful in the process of developing software systems where it serves as a reference point for users, implementers, testers, and writers of instruction manuals. Formal specifications have been applied successfully in deployment of sophisticated software systems; see [33], especially the references there.

Mathematically, formal algebraic specifications are firmly grounded on algebraic concepts, especially ideas, notions, and methods from universal algebra [6]. The relationship between implementation and equational specification corresponds, in algebraic terms, to the relationship between an algebra and a set of identities satisfied by the algebra. Thus, two algebras that satisfy the same identities correspond to a pair of implementations with precisely the same specifications. The computational complexity of these problems, in the universal algebraic framework, is thus quite relevant to the body of research in formal specification theory, and to the construction of supporting tools such as theorem provers and model checkers.

Generalizing the notion of identity, we arrive at a quasi-identity. We shall leave a precise definition for section 2, but crudely speaking, a quasi-identity involves a conjunction of identities and an implication. An example is the left-cancellation law (for, say, a semigroup). In direct analogy with the previous problem we can ask for the complexity of the following. Given two finite algebras of the same finite similarity type, determine whether they satisfy exactly the same quasi-identities. This notion too extends to algebraic specification theory, since "conditional specifications" take the form of quasi-identities.

Our third problem involves the term operations of an algebra. Although an algebra may be endowed with only finitely many basic operations, we can construct many more by composing the basic ones in various combinations. These are called the term operations of the algebra. Two algebras (presumably of different similarity types) are called term-equivalent if they have the same universe and exactly the same set of term operations. In universal algebra, term-equivalent algebras are considered the same "for all practical purposes." The problem we call TERM-EQUIV is that of determining whether two finite algebras are term-equivalent. Returning once again to the realm of specification theory, in this problem we are asking whether a pair of implementations for two entirely different specifications has the property that it exhibits the same input-output behavior. See [25], where this notion is called the "behavioral equivalence of specifications."

Each of these three problems makes sense for arbitrary finite algebras with an arbitrary (but finite) set of basic operations. In addition to this most general formu-

lation, we consider, for each of the three problems, two more restricted settings that, experience tells us, may result in different complexities (see Table 4.1). The first is to require that all basic operations on our algebras be unary. In the second, we only consider algebras of cardinality two.

Unary algebras constitute, from the standpoint of similarity type, the simplest sort of algebraic structure. The set of available term operations is quite small, and the free algebras in the generated variety have a simple structure. Furthermore, (finite) unary algebras capture the algebraic aspects of deterministic (finite-state) automata. (Here, the universe of the algebra corresponds to the set of states, and the similarity type corresponds to the input alphabet of the automata.)

Algebras of cardinality two are, of course, the smallest nontrivial algebras. These are the "Boolean" algebras, and they play an important role in the study of Boolean functions and circuits; see [23] and [32]. For us, the clue that the complexity of our problems will be lower when restricted to two-element algebras comes from the lattice of clones, called Post's lattice [24, 23]. Over a set of cardinality two, the lattice of clones is highly structured and quite manageable. Most of the nice properties of the lattice seem to disappear over larger base sets. One can hope that a good understanding of Post's lattice will lead to the design of more efficient algorithms. Algorithm 7.3 is an example of such an improvement over the "obvious" approach.

The first two sections of this paper are devoted to a development of the necessary background in both universal algebra and complexity theory, for the benefit of those unfamiliar with the basic notions in these fields. In section 3, we formally state the problems we will discuss and outline the major results. Then one section is devoted to each of the three main problems under consideration.

2. Universal algebraic preliminaries. Our primary reference for definitions and basic facts of universal algebra is [20]. Other good references are [5], especially for the material on quasi-varieties, and [10]. Although a bit dated, Taylor's survey in [31] is particularly readable. However, for the benefit of those readers unfamiliar with this material, we give an informal summary of the most important concepts that we will need in this paper.

Let A be a nonempty set and k a nonnegative integer. A k-ary *operation on* A is a function from A^k to A. The integer k is called the *rank* of the operation. Note that if $k = 0$, then $A^k = \{\varnothing\}$, so that a nullary operation is effectively an element of A.

Let F be a set of symbols (called *operation symbols*), and let ρ be a function from F to the nonnegative integers. An *algebra of similarity type* ρ is a structure $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ in which A is a nonempty set and $F^{\mathbf{A}} = \langle f^{\mathbf{A}} : f \in F \rangle$, where each $f^{\mathbf{A}}$ is an operation on A of rank $\rho(f)$. The members of $F^{\mathbf{A}}$ are called the *basic operations* of $\mathbf{A}$, and the set A is called the *universe,* or *underlying set* of $\mathbf{A}$. We will often leave off the superscript $\mathbf{A}$ when no confusion will result. The notation "$\mathbf{A} \sim \mathbf{B}$" will indicate that $\mathbf{A}$ and $\mathbf{B}$ have the same similarity type.

Suppose that $\mathbf{A} = \langle A, F^{\mathbf{A}} \rangle$ and $\mathbf{B} = \langle B, F^{\mathbf{B}} \rangle$ are two algebras of similarity type ρ. A function $\psi\colon B \to A$ is called a *homomorphism* if, for every $f \in F$ and $b_1, \ldots, b_{\rho(f)} \in B$, we have $\psi f^{\mathbf{B}}(b_1, \ldots, b_{\rho(f)}) = f^{\mathbf{A}}(\psi b_1, \ldots, \psi b_{\rho(f)})$. Injective homomorphisms are often called *embeddings.* The algebras $\mathbf{A}$ and $\mathbf{B}$ are *isomorphic,* denoted $\mathbf{A} \cong \mathbf{B}$, if there is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ that is a bijection. $\mathbf{B}$ is called a *subalgebra* of $\mathbf{A}$ if $B \subseteq A$ and the inclusion map is a homomorphism. Thus $\mathbf{B}$ is isomorphic to a subalgebra of $\mathbf{A}$ if and only if there is an embedding of $\mathbf{B}$ into $\mathbf{A}$.

Let J be a set, and for each $j \in J$ let A_j be a nonempty set. Then the Cartesian product, $\prod_{j \in J} A_j$, is the set of all sequences $a = \langle\, a_j : j \in J \,\rangle$ such that for every

$j \in J$, $a_j \in A_j$. For each $i \in J$, there is a surjective function $\pi_i \colon \prod_{j\in J} A_j \to A_i$ mapping the sequence a to its ith component, a_i. We shall reserve the symbol "π" for these mappings. Suppose now that $\langle\, \mathbf{A}_j : j \in J \,\rangle$ is a sequence of algebras, all of similarity type ρ. Then $\prod_{j\in J} \mathbf{A}_j$ is the algebra (of type ρ) whose universe is the Cartesian product of the sets A_j, with basic operations that act coordinatewise, using the basic operations of each $\mathbf{A}_j$. In other words, if f is a basic operation symbol of rank n and $x_1, x_2, \dots, x_n \in \prod_{j\in J} A_j$, then

$$\pi_i f^{\prod \mathbf{A}_j}(x_1, \dots, x_n) = f^{\mathbf{A}_i}(\pi_i x_1, \dots, \pi_i x_n) \qquad \forall\ i \in J.$$

In addition to the basic operations of an algebra, one can create new operations by composing the basic ones. Specifically, let A be a set, f an n-ary operation on A, and let $g_1, \dots, g_n$ be k-ary operations on A. Then the *generalized composition of f with $g_1, \dots, g_n$,* denoted $f[g_1, \dots, g_n]$, is the k-ary operation that maps the k-tuple $\mathbf{a} = (a_1, \dots, a_k)$ to $f\big(g_1(\mathbf{a}), \dots, g_n(\mathbf{a})\big)$.

For each positive integer n and $1 \le j \le n$ we define the jth n-ary *projection operation* by $p_j^n(x_1, \dots, x_n) = x_j$. In particular, p_1^1 is the identity operation. A *clone* on a set A is a set of operations on A containing all projections and closed under generalized composition. The set of all clones on A is obviously ordered by set-theoretic inclusion. The smallest clone consists of nothing but the projection operations, while the largest clone contains all operations on A. It is easy to see that the intersection of a family of clones on A is again a clone. Therefore, if E is any set of operations on A, we define the *clone on A generated by E* to be

$$\mathrm{Clo}^A(E) = \bigcap \{\, C : E \subseteq C \text{ and } C \text{ a clone on } A \,\}. \tag{2.1}$$

For an algebra $\mathbf{A} = \langle A, F\rangle$, the clone of *term operations of* $\mathbf{A}$ is simply $\mathrm{Clo}^A(F)$, the clone on A generated by F. This is typically denoted $\mathrm{Clo}(\mathbf{A})$. For any positive integer m, the set of m-ary members of $\mathrm{Clo}(\mathbf{A})$ is denoted $\mathrm{Clo}_m(\mathbf{A})$.

While (2.1) serves as a definition of $\mathrm{Clo}(\mathbf{A})$, it does not provide any information as to the contents of this clone. Intuitively, an operation on A is a member of $\mathrm{Clo}(\mathbf{A})$ if and only if it can be built up by generalized composition, from the basic operations of $\mathbf{A}$ and the projections. This is formalized in the following theorem. The proof is straightforward; see [20, Theorem 4.3].

THEOREM 2.1. *Let F be a set of operations on a set A, and let m be a positive integer. The set* $\mathrm{Clo}_m^A(F)$ *of* m-ary *members of* $\mathrm{Clo}^A(F)$ *is the smallest set X of* m-ary *operations on A such that*

(i) $p_i^m \in X$ *for* $i = 1, 2, \dots, m$;
(ii) *if* $f \in F$ *and* $g_1, g_2, \dots, g_{\rho(f)} \in X$, *then* $f[g_1, \dots, g_{\rho(f)}] \in X$.

Just as the basic operations of an algebra $\mathbf{A}$ are instances of the operation symbols, we would like to have syntactic objects that correspond to the term operations of $\mathbf{A}$. One way to do this is as follows.

DEFINITION 2.2. *Let $X = \{x_1, x_2, \dots\}$ be a countably infinite set of variables and $\rho \colon F \to \{0, 1, \dots\}$ a similarity type, with F disjoint from X. The set of* terms of type ρ *is the smallest set T of strings such that*

1. $X \subseteq T$;
2. $\rho^{-1}(0) \subseteq T$;
3. *if* $f \in F$ *and* $t_1, \dots, t_{\rho(f)} \in T$, *then* $f(t_1, \dots, t_{\rho(f)}) \in T$.

A term is n-ary *if the only variables that appear in the string come from* $\{x_1, \dots, x_n\}$.

If $\mathbf{A}$ is an algebra and t is an n-ary term, then we can assign an n-ary term operation $t^{\mathbf{A}}$ to t as follows. If $t = x_i$, then $t^{\mathbf{A}} = (p_i^n)^{\mathbf{A}}$. If $t = f(t_1, \dots, t_k)$,

then $t^{\mathbf{A}} = f^{\mathbf{A}}[t_1^{\mathbf{A}}, \ldots, t_k^{\mathbf{A}}]$. Comparing the assertions in Theorem 2.1 to the above definitions, we see that $\mathrm{Clo}(\mathbf{A}) = \{\, t^{\mathbf{A}} : t \text{ is a term} \,\}$.

An *identity* of type ρ is simply a pair of terms, although we usually write it in the form $s \approx t$. An algebra $\mathbf{A}$ satisfies the identity $s \approx t$ if $s^{\mathbf{A}} = t^{\mathbf{A}}$. In the usual terminology of first-order logic, this is the same as asserting that the model $\mathbf{A}$ satisfies the sentence

$$(\forall x_1)(\forall x_2)\cdots(\forall x_n)\,\big(s(x_1, \ldots, x_n) \approx t(x_1, \ldots, x_n)\big),$$

where s and t are n-ary terms.

For example, if our similarity type consists of two binary operation symbols, $+$ and $*$, then both the commutative law $x + y \approx y + x$ and the distributive law $x * (y + z) \approx (x * y) + (x * z)$ are identities that may or may not hold in any particular algebra. Notice that in this example we have adopted the usual custom of writing binary operations in "infix" form and using variables x, y, z instead of x_1, x_2, x_3.

A *quasi-identity* is a first-order sentence of the form

$$(\forall x_1)\cdots(\forall x_n)\,\Big(\bigwedge_{j=1}^{m} s_j(x_1, \ldots, x_n) \approx t_j(x_1, \ldots, x_n) \longrightarrow u(x_1, \ldots, x_n) \approx v(x_1, \ldots, x_n)\Big),$$

where each s_j, t_j, u, and v is an n-ary term, and "$\bigwedge$" denotes conjunction. Every identity is a quasi-identity, by taking $m = 0$. In the context of algebraic structures, quasi-identities are precisely the universal Horn sentences. An example of a quasi-identity that is not an identity is the left-cancellation law $x * y \approx x * z \to y \approx z$, which holds, for example, in the positive integers under multiplication but not for all integers under multiplication. By contrast, the formula

$$(x \neq 0 \ \wedge \ x * y \approx x * z) \to y \approx z$$

is not a quasi-identity, since we do not permit negation symbols.

Now, fix a similarity type ρ, and let $\mathcal{K}$ be a class of algebras, all of type ρ. $\mathcal{K}$ is called a *variety,* or *equational class,* if there is a set Σ of identities (not necessarily finite) such that $\mathcal{K}$ is exactly the class of all algebras satisfying every identity in Σ. Notice that many familiar classes of algebras are varieties. For example, the class of all groups is a variety if we take our similarity type to consist of one binary, one unary, and one nullary operation, and Σ to consist of the five identities

$$x \cdot (y \cdot z) \approx (x \cdot y) \cdot z, \quad x \cdot e \approx e \cdot x \approx x, \quad x \cdot x^{-1} \approx x^{-1} \cdot x \approx e.$$

A classical theorem due to Birkhoff [3] asserts that a class $\mathcal{K}$ is a variety if and only if $\mathcal{K}$ is closed under the formation of subalgebras, arbitrary products, and homomorphic images. This remarkable fact connects the purely syntactic idea of an equation to the familiar algebraic constructions we discussed earlier.

It is convenient to introduce the following notation. Let $\mathcal{K}$ be a class of algebras of the same similarity type. Then $\mathbf{H}(\mathcal{K}), \mathbf{S}(\mathcal{K})$, and $\mathbf{P}(\mathcal{K})$ denote the class of all

algebras isomorphic to a homomorphic image, subalgebra, and product of members of $\mathcal{K}$, respectively. Thus $\mathcal{K}$ is a variety if and only if $\mathcal{K} = \mathbf{H}(\mathcal{K}) = \mathbf{S}(\mathcal{K}) = \mathbf{P}(\mathcal{K})$.

It is easy to see that the intersection of a family of varieties (all of similarity type ρ) is again a variety. In fact, a defining set of identities will be the union of defining sets for each of the component varieties. Thus we define, for any class $\mathcal{K}$, the *variety generated by* $\mathcal{K}$ to be

$$\mathbf{V}(\mathcal{K}) = \bigcap \{ \mathcal{V} : \mathcal{V} \text{ is a variety and } \mathcal{K} \subseteq \mathcal{V} \}.$$

It is not hard to show that for any class $\mathcal{K}$,

$$\mathbf{V}(\mathcal{K}) = \mathbf{HSP}(\mathcal{K}) = \mathrm{Mod}(\mathrm{Id}(\mathcal{K})). \tag{2.2}$$

In this equation, $\mathrm{Id}(\mathcal{K})$ denotes the set of all identities true in every member of $\mathcal{K}$, and $\mathrm{Mod}(\Sigma)$ denotes the class of all algebras in which every identity in Σ holds. For a single algebra $\mathbf{A}$, it is customary to write $\mathbf{V}(\mathbf{A})$ instead of $\mathbf{V}(\{\mathbf{A}\})$.

Just as a variety is defined by identities, a *quasi-variety* is defined by quasi-identities. Most of the assertions we have made about varieties have analogous formulations for quasi-varieties. For example, there is a Birkhoff-type theorem that states that a class $\mathcal{K}$ is a quasi-variety if and only if it is closed under the formation of subalgebras, products, and ultraproducts. (We will not need the notion of an ultraproduct here. See [5, p. 210].) The *quasi-variety generated by* $\mathcal{K}$ is

$$\mathbf{Q}(\mathcal{K}) = \bigcap \{ \mathcal{Q} : \mathcal{Q} \text{ is a quasi-variety and } \mathcal{K} \subseteq \mathcal{Q} \}.$$

Since every variety is also a quasi-variety, we always have $\mathcal{K} \subseteq \mathbf{Q}(\mathcal{K}) \subseteq \mathbf{V}(\mathcal{K})$, but the reverse inclusions are usually false. For example, let $\mathbb{Z}$ denote the group of integers, and $\mathcal{K} = \{\mathbb{Z}\}$. Then $\mathbf{Q}(\mathcal{K})$ is the class of torsion-free Abelian groups, while $\mathbf{V}(\mathcal{K})$ consists of all Abelian groups. (A group is *torsion-free* if no element except the identity has finite order.)

There is one feature of the quasi-variety notion that deserves special mention. Since we will need it later, we state it formally.

THEOREM 2.3. *Let* $\mathbf{A}$ *be a finite algebra. Then* $\mathbf{Q}(\mathbf{A}) = \mathbf{SP}(\mathbf{A})$.

Proof. Since $\mathbf{A} \in \mathbf{SP}(\mathbf{A}) \subseteq \mathbf{Q}(\mathbf{A})$, it suffices to show that $\mathbf{SP}(\mathbf{A})$ is a quasi-variety. Closure under $\mathbf{S}$ and $\mathbf{P}$ is easy. Closure under ultraproducts boils down to the fact that, since A is finite, an ultraproduct of copies of $\mathbf{A}$ is simply isomorphic to $\mathbf{A}$ again. See [5, Lemma 6.5 and Theorem 2.25]. □

An algebra $\mathbf{B}$ is called *simple* if it has more than one element and every homomorphism with domain B is either injective or trivial (i.e., has a one-element image). For example, a group is simple in this sense if and only if it has exactly two normal subgroups. We will need the following easy result.

PROPOSITION 2.4. *Let* $\mathbf{B}$ *be a simple algebra and* $\mathbf{A}$ *be any algebra of the same similarity type. If* $\mathbf{B} \in \mathbf{SP}(\mathbf{A})$, *then* $\mathbf{B} \in \mathbf{S}(\mathbf{A})$.

Proof. By assumption there is a set I and an embedding $\psi\colon \mathbf{B} \to \mathbf{A}^I$. Pick distinct elements a, b from B. Then $\psi(a) \neq \psi(b)$, so for some $i \in I$, $\psi(a)$ and $\psi(b)$ differ in the ith component. Let π_i be the mapping from $\mathbf{A}^I$ to $\mathbf{A}$ that assigns to each I-tuple its ith coordinate. Then $\pi_i \circ \psi$ is a homomorphism from $\mathbf{B}$ to $\mathbf{A}$ which is nontrivial, since $\pi_i\psi(a) \neq \pi_i\psi(b)$. From the simplicity of $\mathbf{B}$, it follows that $\pi_i \circ \psi$ is an embedding. Hence $\mathbf{B} \in \mathbf{S}(\mathbf{A})$. □

TABLE 3.1
Some complexity classes.

$\mathbf{L} = \mathbf{DSPACE}(\log n)$,	logarithmic space;
$\mathbf{NL} = \mathbf{NSPACE}(\log n)$,	nondeterministic logarithmic space;
$\mathbf{P} = \bigcup_{k\geq 1} \mathbf{DTIME}(n^k)$,	polynomial time;
$\mathbf{NP} = \bigcup_{k\geq 1} \mathbf{NTIME}(n^k)$,	nondeterministic polynomial time;
$\mathbf{PSPACE} = \bigcup_{k\geq 1} \mathbf{DSPACE}(n^k)$,	polynomial space;
$\mathbf{EXPTIME} = \bigcup_{k\geq 1} \mathbf{DTIME}(2^{n^k})$,	exponential time;
$\mathbf{2\text{-}EXPTIME} = \bigcup_{k\geq 1} \mathbf{DTIME}(2^{2^{n^k}})$,	hyperexponential time.

3. Complexity preliminaries. Since this paper deals with the computational complexity of problems in universal algebra, we will include a brief review of the complexity classes used in this paper, mainly for the benefit of those readers unfamiliar with the common notation and terminology used. Consult any of [27, 22, 13, 9] for an in-depth treatment of computational complexity.

Languages (i.e., sets of finite strings over some fixed alphabet) are viewed as encodings of problems. Given a function $f\colon \mathbb{N} \to \mathbb{N}$, we denote by $\mathbf{DTIME}(f(n))$ (respectively, $\mathbf{DSPACE}(f(n))$) the set of all languages decidable by a deterministic Turing machine in time (respectively, space) $O(f(n))$. In an analogous way, the nondeterministic classes $\mathbf{NTIME}(f(n))$ and $\mathbf{NSPACE}(f(n))$ are defined in terms of nondeterministic Turing machines. The complexity classes referred to in this paper are defined in Table 3.1 (see [13]). The class **P** consists of those problems for which it is considered to be feasible to use a computer to find a solution. By contrast, a problem lies in **NP** if a proposed solution can be *verified* in polynomial time.

These classes form a chain of inclusions:

$$\mathbf{L} \subseteq \mathbf{NL} \subseteq \mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{PSPACE} \subseteq \mathbf{EXPTIME} \subseteq \mathbf{2\text{-}EXPTIME}. \tag{3.1}$$

Of these the following are known to be proper:

$$\mathbf{NL} \subsetneq \mathbf{PSPACE}, \qquad \mathbf{P} \subsetneq \mathbf{EXPTIME} \subsetneq \mathbf{2\text{-}EXPTIME}. \tag{3.2}$$

All of the other inclusions (except for the obvious ones that follow from (3.2)) represent deep open problems in theoretical computer science, the most famous of which is the seemingly unapproachable $\mathbf{P} \stackrel{?}{=} \mathbf{NP}$ problem.

In addition to the classes listed above, several others are worth noting briefly. First, both **EXPTIME** and **2-EXPTIME** have nondeterministic analogues which could be added to our list. Surprisingly, the same is not true for **PSPACE**. In 1970, Savitch [26] proved that for any function f with $f(n) \geq \log n$, $\mathbf{NSPACE}(f(n)) \subseteq \mathbf{DSPACE}(f(n)^2)$. In particular, $\mathbf{PSPACE} = \mathbf{NPSPACE}$.

Second, for any complexity class **C**, one can define the dual class **co-C**, consisting of those languages (problems) whose *complements* lie in **C**. For example, since the graph-isomorphism problem lies in **NP**, the graph-nonisomorphism problem lies in

co-NP. It is not hard to see that every deterministic class **C** is closed under complements, i.e., **C** = **co-C**. Furthermore, it was shown independently by Immerman and Szelepcsényi [12, 29], that if $f(n) \geq \log n$, then **NSPACE**$(f(n))$ is closed under complements. From this we obtain **NL** = **co-NL**.

Among the most useful tools in complexity theory are the concepts of resource-bounded reducibility and completeness, both borrowed from recursive function theory. Given two languages A and B, we say that A is *polynomial-time, many-one reducible to B,* and we write $A \leq_m^p B$, if there is a polynomial-time computable function f such that

$$x \in A \iff f(x) \in B. \tag{3.3}$$

It is not hard to verify that "$\leq_m^p$" is both reflexive and transitive. A language (i.e., problem) A is $\leq_m^p$*-hard* (or just *hard*) for a class **C** of problems, if for every $C \in \mathbf{C}$, $C \leq_m^p A$. The language A is $\leq_m^p$*-complete* (or just complete) for **C** if A is hard for **C** and $A \in \mathbf{C}$.

Unfortunately, polynomial-time reductions are not useful within the class **P**, since for any nontrivial $A, B \in \mathbf{P}$ we have $A \leq_m^p B \leq_m^p A$. Instead, we must resort to a weaker notion called *log-space reduction.* This simply means that in (3.3), the function f must be computable in logarithmic space.

Since it is generally believed that all of the inclusions in (3.1) are in fact proper, a proof of completeness of a problem A in any of those classes is viewed as providing overwhelming evidence that A does not belong to the immediately preceding class in (3.1). It follows from (3.2) that any problem which is complete for **EXPTIME** (such as our Term-Equiv) fails to lie in **P**. Such a problem is *provably* intractable.

In the problems we will be investigating, the input consists of pairs of finite algebras of *finite similarity type* (i.e., algebras having only finitely many basic operation symbols). Let us be more specific as to the form we assume the input will take. The underlying set of an algebra can be assumed to be $\{0, 1, \ldots, n-1\}$ for some positive integer n. In fact, this set can be represented in the input by its cardinality, which requires $\log n$ bits of storage. (All logarithms will be to the base 2.) A k-ary operation on this set is represented as a table of values or, in other words, a k-dimensional array with both the indices and entries coming from $\{0, 1, \ldots, n-1\}$. Notice that this can be represented in the input stream using $n^k \cdot \log n$ bits.

Let us define the *rank* of an algebra to be the maximum rank of any of its basic operations. Thus an algebra of cardinality n and rank k will require at least $n^k \cdot \log n$ bits to specify.

There are certainly other ways of specifying operations, such as with circuits or Turing machines, but we shall not pursue this idea here. Also, from now on we shall assume that all algebras are finite and of finite similarity type.

4. Discussion of the problems. In this paper we shall consider three equivalence relations on algebraic structures. First, given two algebras **A** and **B** of the same similarity type, is $\mathbf{V}(\mathbf{A}) = \mathbf{V}(\mathbf{B})$? In light of (2.2), this is equivalent to asking whether **A** and **B** satisfy exactly the same identities. Note that this only makes sense if the two algebras have the same similarity type. It was shown in [14] that this problem is decidable. We shall denote this problem Var-Equiv. Thus

$$\text{Var-Equiv} = \{\, (\mathbf{A}, \mathbf{B}) : \mathbf{A} \sim \mathbf{B} \;\&\; \mathbf{V}(\mathbf{A}) = \mathbf{V}(\mathbf{B}) \,\}.$$

Recently, Z. Székely proved that Var-Equiv is **NP**-hard; see [28].

We have an analogous problem for quasi-varieties:

$$\text{QVAR-EQUIV} = \{\, (\mathbf{A}, \mathbf{B}) : \mathbf{A} \sim \mathbf{B} \;\&\; \mathbf{Q}(\mathbf{A}) = \mathbf{Q}(\mathbf{B}) \,\}.$$

The assertion $(\mathbf{A}, \mathbf{B}) \in$ QVAR-EQUIV is equivalent to $\mathbf{A}$ and $\mathbf{B}$ satisfying exactly the same quasi-identities. Surprisingly, even though the logical form of a quasi-identity is much more complicated than that of an identity, QVAR-EQUIV has a relatively low computational complexity compared to VAR-EQUIV. Note that QVAR-EQUIV $\subseteq$ VAR-EQUIV as sets.

The third problem we shall consider is term-equivalence. Two algebras $\mathbf{A}$ and $\mathbf{B}$ are *term-equivalent* if and only if they have the same underlying set and $\mathrm{Clo}(\mathbf{A}) = \mathrm{Clo}(\mathbf{B})$. For this problem, we do not require that $\mathbf{A}$ and $\mathbf{B}$ have the same similarity type, but we do require that they have the same universe:

$$\text{TERM-EQUIV} = \{\, (\mathbf{A}, \mathbf{B}) : A = B \;\&\; \mathrm{Clo}(\mathbf{A}) = \mathrm{Clo}(\mathbf{B}) \,\}.$$

It was shown in [1] that TERM-EQUIV is complete for **EXPTIME**.

There are several restrictions of these problems which are of interest and which turn out to have a lower complexity. In particular, we can bound either the cardinality of the underlying sets or the ranks of the algebras. For example, it was shown in [18] that TERM-EQUIV is complete for **PSPACE** when restricted to *unary algebras,* that is, algebras in which every operation has rank 1. For each of our three problems, we shall consider, in addition to the general case, the subcases obtained by considering only unary algebras and only two-element algebras. We shall denote the subcase by appending a superscript "1" or subscript "2" to the problem. To be precise, let us define

$$U = \{\, \mathbf{A} : \mathbf{A} \text{ is a unary algebra} \,\},$$
$$T = \{\, \mathbf{A} : A = \{0, 1\} \,\}.$$

Then $\mathrm{X}^1 = \mathrm{X} \cap (U \times U)$ and $\mathrm{X}_2 = \mathrm{X} \cap (T \times T)$ for X any one of TERM-EQUIV, VAR-EQUIV, or QVAR-EQUIV.

Our results for each of these nine problems are summarized in Table 4.1. In this

TABLE 4.1
Summary of results.

	QVAR-EQUIV	TERM-EQUIV	VAR-EQUIV
card2	**L**	**NL**	**L**
unary	**NP**	**PSPACE***	**PSPACE**
general	**NP***	**EXPTIME***	**2-EXPTIME**

table, the first row concerns the subcase consisting of two-element algebras, the second concerns the subcase of unary algebras, and the third concerns the general case. Each of the nine entries gives the smallest complexity class known to contain the problem, and a superscript "*" indicates that the result is sharp, i.e., the problem is complete for the given complexity class.

5. The quasi-variety problems. We begin with the problems that ask whether two algebras generate the same quasi-variety. It is sometimes convenient to work with an asymmetric variant of this problem:

$$\text{QVAR-MEM} = \{\, (\mathbf{A}, \mathbf{B}) : \mathbf{A} \sim \mathbf{B} \;\&\; \mathbf{B} \in \mathbf{Q}(\mathbf{A}) \,\}.$$

Since the "**Q**" operator has the usual properties of closure, we obviously have

$$(\mathbf{A},\mathbf{B}) \in \text{QVAR-EQUIV} \iff (\mathbf{A},\mathbf{B}),\ (\mathbf{B},\mathbf{A}) \in \text{QVAR-MEM}. \tag{5.1}$$

It follows that for an instance of size s, membership in QVAR-EQUIV can be tested with two calls to an algorithm for QVAR-MEM, both using inputs of size s. In a natural way, we also have the restricted problems QVAR-MEM^1 and QVAR-MEM$_2$ consisting of pairs of unary and two-element algebras, respectively.

THEOREM 5.1. QVAR-MEM $\in$ **NP**.

Proof. Let $\mathbf{A}$ and $\mathbf{B}$ be a pair of similar, finite algebras. We wish to determine whether $\mathbf{B} \in \mathbf{Q}(\mathbf{A})$. Here is a nondeterministic algorithm. For each unordered pair $\{a,b\}$ of distinct elements of B, guess a function $\psi_{\{a,b\}}\colon B \to A$ such that $\psi_{\{a,b\}}(a) \neq \psi_{\{a,b\}}(b)$. Test whether $\psi_{\{a,b\}}$ is a homomorphism. If it is not, then reject. But if every $\psi_{\{a,b\}}$ passes the homomorphism test, then accept.

To see that our algorithm is correct, suppose that we accept the pair $(\mathbf{A},\mathbf{B})$. Enumerate the doubletons $\{a_i,b_i\}$, $i = 1,2,\dots,n$, of elements of B. Define a function $\psi\colon B \to A^n$ such that the ith coordinate of $\psi(x)$ is $\psi_{\{a_i,b_i\}}(x)$. The fact that every $\psi_{\{a_i,b_i\}}$ is a homomorphism ensures that ψ is a homomorphism. And ψ will be injective since, if $a \neq b$, then $\psi(a)$ and $\psi(b)$ differ in the ith coordinate, where $\{a,b\}$ is the ith pair in our enumeration. It follows that $\mathbf{B}$ is isomorphic to a subalgebra of a direct power of $\mathbf{A}$, and consequently $\mathbf{B}$ lies in the quasi-variety generated by $\mathbf{A}$.

Conversely, suppose that $\mathbf{B} \in \mathbf{Q}(\mathbf{A})$. By Theorem 2.3 there is a set J and an embedding $\psi\colon \mathbf{B} \to \mathbf{A}^J$. Using the notation of the previous paragraph, for each $i = 1,\dots,n$, we have $\psi(a_i) \neq \psi(b_i)$, and hence there is some $j_i \in J$ such that $\psi(a_i)$ and $\psi(b_i)$ differ in their j_ith coordinate. Let $\pi_{j_i}\colon \mathbf{A}^J \to \mathbf{A}$ denote the coordinate projection homomorphism. Then $\psi_{\{a_i,b_i\}} = \pi_{j_i} \circ \psi$ constitutes an appropriate guess. Thus our algorithm will accept the pair $(\mathbf{A},\mathbf{B})$.

Finally, we need to estimate the (nondeterministic) running time of the algorithm. Let s denote the size of the input. A function ψ from B to A can be guessed in time on the order of $|B|\cdot|A|$, which is at most s^2. The verification that ψ is a homomorphism also takes time in $O(s^2)$. The total number of functions we need to construct is $\binom{|B|}{2} \leq s^2$. Thus the total running time lies in $O(s^4)$. □

COROLLARY 5.2. *All of the following lie in* **NP***:* QVAR-EQUIV*,* QVAR-MEM^1*,* QVAR-MEM$_2$*,* QVAR-EQUIV^1*, and* QVAR-EQUIV$_2$*.*

Proof. That QVAR-EQUIV $\in$ **NP** follows from Theorem 5.1 and (5.1). The class **NP** is closed under polynomial-time, many-one reductions. Since QVAR-MEM^1 and QVAR-EQUIV^1 are reducible to (indeed special cases of) QVAR-MEM and QVAR-EQUIV, respectively, they too lie in **NP**. The same argument applies to the two-element versions of the problems. □

We shall improve the bounds on QVAR-MEM$_2$ and QVAR-EQUIV$_2$ in Theorem 5.7 below. But first we consider the **NP**-completeness of the other problems discussed in Corollary 5.2. To do this, we use a transformation from (directed) graphs to unary algebras described in [11]. By a *digraph* we shall mean a structure $\mathbf{G} = \langle G,\theta\rangle$ in which G is a nonempty set and $\theta \subseteq G \times G$. $\mathbf{G}$ is *loopless* if for no x in G do we have $(x,x) \in \theta$.

Let $\mathbf{G} = \langle G,\theta\rangle$ and $\mathbf{H} = \langle H,\tau\rangle$ be digraphs. A morphism from $\mathbf{H}$ to $\mathbf{G}$ is a function $\psi\colon H \to G$ such that $(x,y) \in \tau \implies \big(\psi(x),\psi(y)\big) \in \theta$. $\mathbf{H}$ is a subgraph of $\mathbf{G}$ if $H \subseteq G$ and the inclusion map is a morphism. If $\mathbf{G}_i = \langle G_i,\theta_i\rangle$, $i \in I$ is a family of digraphs, then the product graph is $\langle G,\theta\rangle$, where $G = \prod_{i\in I} G_i$ and $\theta = \{\,(x,y) \in G \times G : (\forall i \in I)\ (x_i,y_i) \in \theta_i\,\}$. Just as for algebras, we use the

operators **S** and **P** to denote closure under the formation of subgraph and product graph, respectively.

Given a digraph $\mathbf{G} = \langle G, \theta \rangle$, we now define an algebra $\mathbf{G}^*$ as follows. The universe of $\mathbf{G}^*$ is the set $G \cup \theta \cup \{u, v\}$ where u and v are points not appearing in either G or θ. $\mathbf{G}^* = \langle\, G^*, f_0, f_1 \,\rangle$, where f_0 and f_1 are unary operations defined by

$$\begin{array}{lll} \forall x \in G & f_0(x) = u, & f_1(x) = v; \\ \forall (x,y) \in \theta & f_0((x,y)) = x, & f_1((x,y)) = y; \\ & f_0(u) = v, & f_1(u) = u, \\ & f_0(v) = v, & f_1(v) = u. \end{array}$$

Furthermore, let $\psi\colon \mathbf{H} \to \mathbf{G}$ be a digraph morphism. We define a function $\psi^*\colon H^* \to G^*$ given by

$$\begin{array}{ll} \forall x \in H & \psi^*(x) = \psi(x); \\ \forall (x,y) \in \tau & \psi^*((x,y)) = (\psi(x),\, \psi(y)); \\ \psi^*(u^H) = u^G, & \psi^*(v^H) = v^G. \end{array}$$

LEMMA 5.3 (see Hedrlín and Pultr [11]). *The assignments $\mathbf{G} \mapsto \mathbf{G}^*$ and $\psi \mapsto \psi^*$ constitute a full and faithful functor from the category of digraphs to that of algebras with two unary operations. In other words, for each pair $\mathbf{H}$, $\mathbf{G}$ of digraphs, and each digraph morphism ψ, the function $\psi^*\colon \mathbf{H}^* \to \mathbf{G}^*$ is a homomorphism, and furthermore, the mapping $\psi \mapsto \psi^*$ is a bijection between the morphisms from $\mathbf{H}$ to $\mathbf{G}$ and the homomorphisms between $\mathbf{H}^*$ and $\mathbf{G}^*$. Also, ψ is an injective map if and only if ψ^* is injective.*

We will show **NP**-completeness of QVAR-MEM[1] by exhibiting a reduction from the problem CLIQUE, which is well known to be complete for **NP**; see [9, p. 194]. For a positive integer n, let $\mathbf{K}_n$ denote the digraph with vertex set $\{1, 2, \ldots, n\}$ and edges $\{\, (x,y) : x \neq y \,\}$. We define

$$\text{CLIQUE} = \{\, (\mathbf{G}, n) : \mathbf{G} \text{ a digraph, } n \geq 1 \text{ and } \mathbf{K}_n \in \mathbf{S}(\mathbf{G}) \,\}.$$

A subset of G isomorphic to some $\mathbf{K}_n$ is called a *clique*.

PROPOSITION 5.4. *Let $\mathbf{G}$ be a loopless digraph and n a positive integer. The following are equivalent.*

(i) $(\mathbf{G}, n) \in$ CLIQUE.
(ii) $\mathbf{K}_n \in \mathbf{SP}(\mathbf{G})$.
(iii) $\mathbf{K}_n^* \in \mathbf{SP}(\mathbf{G}^*)$.

Proof. That (i) implies (ii) is trivial. Now assume (ii), i.e., suppose that ψ is an embedding of $\mathbf{K}_n$ into a power $\mathbf{G}^I$. By Lemma 5.3, we have an injective homomorphism ψ^* from $\mathbf{K}_n^*$ to $(\mathbf{G}^I)^*$. Moreover, it is easy to see that $(\mathbf{G}^I)^*$ is isomorphic to a subalgebra of $(\mathbf{G}^*)^I$. Thus (iii) holds.

Finally, suppose that $\phi\colon \mathbf{K}_n^* \to (\mathbf{G}^*)^I$ is an embedding for some set I. Choose any $i \in I$. Then $\pi_i \circ \phi$ is a homomorphism from $\mathbf{K}_n^*$ to $\mathbf{G}^*$, which, by Lemma 5.3, must be of the form ψ^* for some digraph morphism $\psi\colon \mathbf{K}_n \to \mathbf{G}$. Suppose that x and y are distinct elements of $\mathbf{K}_n$. By definition, there is an edge from x to y; consequently, there must be an edge in $\mathbf{G}$ from $\psi(x)$ to $\psi(y)$. Since $\mathbf{G}$ is assumed to be loopless, it must be the case that $\psi(x) \neq \psi(y)$. Therefore, ψ is injective, so (iii) implies (i). □

It is convenient to introduce one more problem involving the relationship between two algebras. Let

$$\text{SUBALG} = \{\, (\mathbf{A}, \mathbf{B}) : \mathbf{A} \sim \mathbf{B} \;\&\; \mathbf{B} \in \mathbf{S}(\mathbf{A}) \,\}.$$

As with our other problems, SUBALG^1 will refer to the restriction of this problem to the case that $\mathbf{A}$ and $\mathbf{B}$ are unary algebras. It is easy to see that SUBALG and SUBALG^1 lie in **NP** (just guess an injective function and check to see if it is a homomorphism).

THEOREM 5.5. *The problems* QVAR-MEM^1, QVAR-MEM, SUBALG^1, SUBALG, *and* QVAR-EQUIV *are all complete for* **NP**.

Proof. We showed in Theorem 5.1, Corollary 5.2, and the comments just above that all of these problems lie in **NP**. The problem CLIQUE is **NP**-complete, and the transformation $(\mathbf{G}, n) \mapsto (\mathbf{G}^*, \mathbf{K}_n^*)$ can be done in polynomial time. Therefore, we deduce from Proposition 5.4 and Theorem 2.3 that QVAR-MEM^1 is also **NP**-complete. Clearly, QVAR-MEM$^1 \leq_m^p$ QVAR-MEM, so QVAR-MEM is **NP**-complete.

By Lemma 5.3, $(\mathbf{G}, n) \in$ CLIQUE if and only if $(\mathbf{G}^*, \mathbf{K}_n^*) \in$ SUBALG^1. Thus SUBALG^1 and also SUBALG are **NP**-complete. This fact is not new; it was first noted in [21].

To complete the proof, we will reduce SUBALG^1 to QVAR-EQUIV. Given a finite unary algebra $\mathbf{B} = \langle B, F\rangle$, let $\mathbf{B}^+ = \big\langle\, B \cup \{e\},\ F \cup \{p, d\}\,\big\rangle$, where $e \notin B$ and $p, d \notin F$. For each $f \in F$, $f(e) = e$. For all $x, y, z \in B \cup \{e\}$ we define

$$p(x) = e;$$

$$d(x, y, z) = \begin{cases} z & \text{if } x = y, \\ x & \text{if } x \neq y. \end{cases}$$

The ternary operation d is called the *discriminator operation*. It is an easy exercise to check that any algebra with a discriminator among its term operations is simple.

Now, given two finite unary algebras $\mathbf{A}$ and $\mathbf{B}$, we have the following equivalences.

$$\mathbf{B} \in \mathbf{S}(\mathbf{A}) \iff \mathbf{B}^+ \in \mathbf{S}(\mathbf{A}^+) \iff \mathbf{Q}(\mathbf{A}^+) = \mathbf{Q}(\mathbf{A}^+ \times \mathbf{B}^+). \tag{5.2}$$

Clearly, the equivalences in (5.2) imply that SUBALG$^1 \leq_m^p$ QVAR-EQUIV. The first equivalence is an easy verification. We check the second. First suppose that $\mathbf{B}^+ \in \mathbf{S}(\mathbf{A}^+)$. Then both $\mathbf{A}^+$ and $\mathbf{B}^+$ are members of $\mathbf{Q}(\mathbf{A}^+)$ which is closed under products. Therefore, $\mathbf{A}^+ \times \mathbf{B}^+ \in \mathbf{Q}(\mathbf{A}^+)$, so $\mathbf{Q}(\mathbf{A}^+ \times \mathbf{B}^+) \subseteq \mathbf{Q}(\mathbf{A}^+)$. On the other hand, there is an embedding of $\mathbf{A}^+$ into $\mathbf{A}^+ \times \mathbf{B}^+$ given by $x \mapsto (x, e)$. Hence $\mathbf{Q}(\mathbf{A}^+) \subseteq \mathbf{Q}(\mathbf{A}^+ \times \mathbf{B}^+)$.

Conversely, suppose that $\mathbf{Q}(\mathbf{A}^+) = \mathbf{Q}(\mathbf{A}^+ \times \mathbf{B}^+)$. There is an embedding of $\mathbf{B}^+$ into $\mathbf{A}^+ \times \mathbf{B}^+$ such that $x \mapsto (e, x)$. Thus $\mathbf{B}^+ \in \mathbf{Q}(\mathbf{A}^+) = \mathbf{SP}(\mathbf{A}^+)$. However, $\mathbf{B}^+$ is a simple algebra, so by Proposition 2.4, $\mathbf{B}^+ \in \mathbf{S}(\mathbf{A}^+)$. □

Remark. Referring back to the proof of Theorem 5.5, it is tempting to try to prove the **NP**-hardness of QVAR-EQUIV by using (5.1) together with the **NP**-completeness of QVAR-MEM and Corollary 5.2. However, there are difficulties with this line of argument. To take an analogous situation, the subgraph isomorphism problem is **NP**-complete, but the graph isomorphism problem is in **NP** but most probably is not **NP**-complete.

Unfortunately, the method used to reduce SUBALG^1 to QVAR-EQUIV does not produce a unary algebra, so we are not able to show that QVAR-EQUIV^1 is complete for **NP**. We leave it as an open problem.

PROBLEM 5.6. *Is* QVAR-EQUIV^1 *complete for* **NP**? *Is* QVAR-EQUIV$^1 \in$ **P**?

Now we turn to the problem QVAR-EQUIV$_2$. Suppose that $\mathbf{A}$ and $\mathbf{B}$ are two-element algebras. We claim that $\mathbf{B} \in \mathbf{Q}(\mathbf{A})$ if and only if $\mathbf{B} \cong \mathbf{A}$. To see this, note that every two-element algebra is simple. Therefore, by Proposition 2.4, Theorem 2.3, and the fact that $|B| = |A|$, we obtain

$$\mathbf{B} \in \mathbf{Q}(\mathbf{A}) = \mathbf{SP}(\mathbf{A}) \implies \mathbf{B} \in \mathbf{S}(\mathbf{A}) \implies \mathbf{B} \cong \mathbf{A}.$$

The converse, that $\mathbf{B} \cong \mathbf{A} \implies \mathbf{B} \in \mathbf{Q}(\mathbf{A})$, is trivial.

THEOREM 5.7. QVAR-EQUIV$_2$, QVAR-MEM$_2 \in \mathbf{L}$.

Proof. As we argued in the previous paragraph, $(\mathbf{B}, \mathbf{A}) \in$ QVAR-MEM$_2$ if and only if $\mathbf{B} \cong \mathbf{A}$. There are only two bijections from B to A, and each of these can be tested to see if it is a homomorphism. The testing requires just a couple of counters, each of which has a space bound that is logarithmic in the size of the input. Thus QVAR-MEM$_2 \in \mathbf{L}$. Now apply assertion (5.1) to deduce that QVAR-EQUIV$_2$ $\in \mathbf{L}$. □

6. The variety problems. The problem VAR-EQUIV asks: if $\mathbf{A}$ and $\mathbf{B}$ are two algebras of the same similarity type, is $\mathbf{V}(\mathbf{A}) = \mathbf{V}(\mathbf{B})$? As with quasi-varieties, it is convenient to introduce an auxiliary problem, VAR-MEM:

$$\text{VAR-MEM} = \{\, (\mathbf{A}, \mathbf{B}) : \mathbf{A} \sim \mathbf{B} \;\&\; \mathbf{B} \in \mathbf{V}(\mathbf{A}) \,\}.$$

Unlike the situation for quasi-varieties, the problems VAR-EQUIV and VAR-MEM are interchangeable from the perspective of complexity. We have

$$\begin{aligned} (\mathbf{A}, \mathbf{B}) \in \text{VAR-EQUIV} &\iff (\mathbf{A}, \mathbf{B}), (\mathbf{B}, \mathbf{A}) \in \text{VAR-MEM}, \\ (\mathbf{A}, \mathbf{B}) \in \text{VAR-MEM} &\iff (\mathbf{A}, \mathbf{A} \times \mathbf{B}) \in \text{VAR-EQUIV}. \end{aligned} \tag{6.1}$$

The second equivalence follows from the fact that both $\mathbf{A}$ and $\mathbf{B}$ are homomorphic images of $\mathbf{A} \times \mathbf{B}$.

We begin with the two-element problem. The crucial point is the following theorem.

THEOREM 6.1. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be two-element algebras of the same similarity type. Then* $\mathbf{V}(\mathbf{A}) = \mathbf{V}(\mathbf{B})$ *if and only if* $\mathbf{A} \cong \mathbf{B}$.

The proof of Theorem 6.1 requires considerably more universal algebra than does the remainder of this paper. For this reason, we have relegated the proof to the appendix.

THEOREM 6.2. VAR-EQUIV$_2 \in \mathbf{L}$.

Proof. From Theorem 6.1, testing whether $(\mathbf{A}, \mathbf{B}) \in$ VAR-EQUIV$_2$ is equivalent to testing $\mathbf{A} \cong \mathbf{B}$. Arguing as we did at the end of section 5, there are only two possible isomorphisms to test. This can be done deterministically in logarithmic space. □

In order to proceed to the remaining two problems, we need some more detailed information on the relationship between clones, terms, and varieties. Let $\mathbf{A} = \langle A, F \rangle$ be an algebra of cardinality n, and let m be a positive integer. An m-ary operation on A, being a function from A^m to A, can also be thought of as an element of the direct power $A^{(A^m)}$. Visualized this way, it is not hard to see that $\mathrm{Clo}_m(\mathbf{A})$ forms a subalgebra of $\mathbf{A}^{(A^m)}$. In fact, Theorem 2.1 can be viewed as asserting that $\mathbf{Clo}_m(\mathbf{A})$ is the subalgebra of $\mathbf{A}^{(A^m)}$ generated by the set $\{p_1^m, \ldots, p_m^m\}$. Notice that we follow our usual typographic convention and print "**Clo**" in boldface when it is to be used as an algebra. Since varieties are closed under the formation of both powers and subalgebras, it follows that both $\mathbf{A}^{(A^m)}$ and $\mathbf{Clo}_m(\mathbf{A})$ lie in $\mathbf{V}(\mathbf{A})$.

Let us be more precise about how this subalgebra could be constructed. For each natural number j, define a set X_j of m-ary operations on A recursively by

$$\begin{aligned} X_0 &= \{p_1^m, p_2^m, \ldots, p_m^m\}, \\ X_{j+1} &= X_j \cup \left\{\, f[g_1, \ldots, g_{\rho(f)}] : f \in F,\ g_1, \ldots, g_{\rho(f)} \in X_j \,\right\}. \end{aligned} \tag{6.2}$$

It is easy to see that $X_0 \subseteq X_1 \subseteq \cdots \subseteq \mathrm{Clo}_m(\mathbf{A})$. Since the total number of m-ary operations on A is $n^{(n^m)}$, there is some index $q < n^{(n^m)}$ such that $X_q = X_{q+1}$. But

the pair of conditions $X_j \supseteq X_0$ and $X_j = X_{j+1}$ are precisely those of Theorem 2.1. Therefore, $X_q = \mathrm{Clo}_m(\mathbf{A})$.

The case of a unary algebra deserves special consideration. Suppose that $\mathbf{A}$ is unary (i.e., has rank 1). While it is true that, technically speaking, $\mathbf{A}$ has term operations of arbitrarily large rank, these operations are trivial in the sense that they only depend on one of their variables. For example, if f is a basic operation of $\mathbf{A}$, then the term operation $f[p_1^2]$ has rank 2. However, this operation maps any pair (x, y) to $f(x)$. Thus $f[p_1^2]$ is "essentially unary." Since there are only n^n possible unary operations, one can easily show that for any m, we always have $X_{n^n-1} = \mathrm{Clo}_m(\mathbf{A})$.

Now one cannot help but notice the similarity between the definition of the sets X_j in (6.2) and that of a term given in Definition 2.2. Of course, if t is an m-ary term, then $t^{\mathbf{A}} \in \mathrm{Clo}_m(\mathbf{A})$, hence, for some $j < n^{(n^m)}$, $t^{\mathbf{A}} \in X_j$. Conversely, if we view each m-ary term as a tree, then every member of X_j is of the form $t^{\mathbf{A}}$ for some term t of height at most j. Let us write $\mathrm{ht}(t)$ to denote the height of the term t. Putting these two observations together, for every m-ary term t, there is a term t' such that $t^{\mathbf{A}} = (t')^{\mathbf{A}}$ and $\mathrm{ht}(t') < n^{(n^m)}$. Following up on our earlier observation, in the special case that $\mathbf{A}$ is unary, the bound on the height of t' can be reduced to n^n no matter what the value of m.

THEOREM 6.3. *Let $\mathbf{A}$ and $\mathbf{B}$ be finite algebras of the same similarity type. Assume that the cardinalities of $\mathbf{A}$ and $\mathbf{B}$ are n and m, respectively. Then the following are equivalent.*

(i) $\mathbf{B} \in \mathbf{V}(\mathbf{A})$.

(ii) *For every pair of terms s and t, each of height at most $n^{(n^m)}$, if $\mathbf{A}$ satisfies the identity $s \approx t$, then so does $\mathbf{B}$.*

(iii) $\mathbf{B}$ *is a homomorphic image of the algebra* $\mathbf{Clo}_m(\mathbf{A})$.

If $\mathbf{A}$ and $\mathbf{B}$ are unary algebras, then the bound $n^{(n^m)}$ in (ii) *can be reduced to n^n.*

Proof. That (i) implies (ii) follows from (2.2). So assume (ii). Enumerate the elements of B as $b_1, \ldots, b_m$. Define a function $\psi\colon \mathrm{Clo}_m(\mathbf{A}) \to B$ as follows. For each $g \in \mathrm{Clo}_m(\mathbf{A})$, choose a term u such that $\mathrm{ht}(u) < n^{(n^m)}$ and $u^{\mathbf{A}} = g$, and define $\psi(g) = u^{\mathbf{B}}(b_1, \ldots, b_m)$. To see that this is well defined, suppose that s is another term with the properties $s^{\mathbf{A}} = g$ and $\mathrm{ht}(s) < n^{(n^m)}$. Then $s^{\mathbf{A}} = g = u^{\mathbf{A}}$, so $\mathbf{A}$ satisfies the identity $s \approx u$. Therefore, by (ii), $\mathbf{B}$ also satisfies the identity $s \approx u$, hence $s^{\mathbf{B}} = u^{\mathbf{B}}$.

The function ψ is surjective since for every $i \leq m$, $b_i = \psi(p_i^m)$. In order to prove (iii), it remains to show that ψ is a homomorphism. So let $g_1, \ldots, g_k \in \mathrm{Clo}_m(\mathbf{A})$, and let f be a basic k-ary operation symbol. There are terms $s_1, s_2, \ldots, s_k$ and t, all of height less than $n^{(n^m)}$ such that $g_i = s_i^{\mathbf{A}}$ for $i = 1, 2, \ldots, k$, and $f^{\mathbf{A}}[g_1, \ldots, g_k] = t^{\mathbf{A}}$. Let r be the term $f(s_1, \ldots, s_k)$. We need to verify that $\psi\big(f^{\mathbf{A}}[g_1, \ldots, g_k]\big) = f^{\mathbf{B}}\big(\psi(g_1), \ldots, \psi(g_k)\big)$. Note that $\mathrm{ht}(r) = 1 + \max_i \mathrm{ht}(s_i) \leq n^{(n^m)}$. Also, $r^{\mathbf{A}} = t^{\mathbf{A}}$ since

$$r^{\mathbf{A}} = f^{\mathbf{A}}[s_1^{\mathbf{A}}, \ldots, s_k^{\mathbf{A}}] = f^{\mathbf{A}}[g_1, \ldots, g_k] = t^{\mathbf{A}}.$$

Therefore, the equation $r \approx t$ holds in $\mathbf{A}$, hence in $\mathbf{B}$, because of the bounds on the heights. Thus $r^{\mathbf{B}} = t^{\mathbf{B}}$. Now, writing $\mathbf{b} = (b_1, \ldots, b_m)$, we compute

$$\begin{aligned}\psi\big(f^{\mathbf{A}}[g_1, \ldots, g_k]\big) &= \psi(t^{\mathbf{A}}) = t^{\mathbf{B}}(\mathbf{b}) = r^{\mathbf{B}}(\mathbf{b}) \\ &= f^{\mathbf{B}}\big(s_1^{\mathbf{B}}(\mathbf{b}), \ldots, s_k^{\mathbf{B}}(\mathbf{b})\big) = f^{\mathbf{B}}\big(\psi(g_1), \ldots, \psi(g_k)\big).\end{aligned}$$

Finally, assume (iii). $\mathbf{Clo}_m(\mathbf{A})$ is a subalgebra of $\mathbf{A}^{A^m}$; consequently, it lies in $\mathbf{V}(\mathbf{A})$. Since every variety is closed under homomorphic images, $\mathbf{B} \in \mathbf{V}(\mathbf{A})$ as well. □

1. $s^A \leftarrow t^A \leftarrow f_0^{\mathbf{A}}$; $\quad s^B \leftarrow t^B \leftarrow f_0^{\mathbf{B}}$.
2. for $i = 1$ to n^n do
3. guess $j, \ell \in \{0, 1, \ldots, k\}$
4. $s^A \leftarrow f_j^{\mathbf{A}} \circ s^A$; $\quad s^B \leftarrow f_j^{\mathbf{B}} \circ s^B$; $\quad t^A \leftarrow f_\ell^{\mathbf{A}} \circ t^A$; $\quad t^B \leftarrow f_\ell^{\mathbf{B}} \circ t^B$
5. if $\big((\forall x, y \in A)\ (s^A(x) = t^A(y))$ and $(\exists x, y \in B)\ (s^B(x) \neq t^B(y))\big)$ or $\big((\forall x \in A)\ (s^A(x) = t^A(x))$ and $(\exists x \in B)\ (s^B(x) \neq t^B(x))\big)$ then **accept**.

ALGORITHM 6.1. *Testing* $(\mathbf{A}, \mathbf{B}) \notin$ VAR-MEM^1.

It is worthwhile to extract just a bit more information from the proof of Theorem 6.3. Notice that in the proof of (ii) implies (iii), we began with an arbitrary enumeration of the elements of B and constructed a homomorphism from $\mathbf{Clo}_m(\mathbf{A})$ onto $\mathbf{B}$. In the language of universal algebra, this is equivalent to the assertion that the algebra $\mathbf{Clo}_m(\mathbf{A})$ is *freely generated* by $X_0 = \{\, p_i^m : 1 \leq i \leq m \,\}$. For our purposes, we can express this property as the following corollary.

COROLLARY 6.4. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be algebras as in Theorem* 6.3. *The following are equivalent.*

(i) $\mathbf{B} \in \mathbf{V}(\mathbf{A})$.

(ii) *For some bijection* ψ_0 *of* X_0 *with* B, *there exists a homomorphism* ψ *from* $\mathbf{Clo}_m(\mathbf{A})$ *to* $\mathbf{B}$ *such that* $\psi{\restriction}_{X_0} = \psi_0$.

(iii) *For every bijection* ψ_0 *of* X_0 *with* B, *there exists a homomorphism* ψ *from* $\mathbf{Clo}_m(\mathbf{A})$ *to* $\mathbf{B}$ *such that* $\psi{\restriction}_{X_0} = \psi_0$.

Theorem 6.3 suggests an approach that can be used to test the condition $\mathbf{B} \notin \mathbf{V}(\mathbf{A})$: simply *guess* an identity ϵ and check to see whether $\mathbf{A}$ satisfies ϵ while $\mathbf{B}$ fails to satisfy ϵ. This approach seems to be quite effective—at least for unary algebras. For in this case, we have the improved bound n^n in part (ii) of the theorem.

Let us fix a set $F = \{f_1, \ldots, f_k\}$ of operation symbols, each of rank 1. Also, let us add an additional unary operation symbol f_0 which will always be interpreted as the identity operation. This has no effect on the algebras but will save us a subscript in our analysis. A typical term over F is of the form $f_{i_\ell} f_{i_{\ell-1}} \cdots f_{i_2} f_{i_1}(x)$, where $i_1, i_2, \ldots, i_\ell \in \{0, 1, \ldots, k\}$. The height of this term is ℓ. Since each term involves only one variable, every identity is of one of two possible forms:

$$s(x) \approx t(x) \quad \text{or} \quad s(x) \approx t(y).$$

Notice that the second of these is quite degenerate since it requires that the term operations corresponding to s and t both be constant and, in fact, the same constant. Nevertheless, it must be considered in the analysis.

Now suppose that $\mathbf{A}$ and $\mathbf{B}$ are algebras of type F and of cardinalities n and m, respectively. Algorithm 6.1 is a nondeterministic algorithm that accepts the pair $(\mathbf{A}, \mathbf{B})$ if and only if $\mathbf{B} \notin \mathbf{V}(\mathbf{A})$.

How much space is used by this algorithm? Let $p = \max(n, m)$. Each of the four unary operations can be represented as a vector of length p. Each such vector requires $p \log(p) \leq p^2$ bits. We also need space for the counter i, which ranges from 0 to n^n. Since $\log(n^n) = n \log(n) \leq p^2$, i requires another p^2 bits. It follows that the total amount of space required is on the order of p^2 bits.

What is the size of the input? The algebra $\mathbf{A}$ requires $\log(n) + kn \log(n) > n$ bits. Similarly $\mathbf{B}$ requires at least m bits. The total input size is at least $n + m > p$ bits. It follows that our algorithm's space requirements are bounded above by the square of the size of the input.

1. Create empty table for ψ. Fill in ψ_0.
2. for $j = 0$ to $n^{(n^m)} - 2$ do
3. for $f \in F$ do (*let* $k = \rho(f)$)
4. for $g_1, \ldots, g_k \in X_j$ do
5. $b \leftarrow f^{\mathbf{B}}(\psi(g_1), \ldots, \psi(g_k))$
6. $b' \leftarrow \psi(f^{\mathbf{A}}[g_1, \ldots, g_k])$
7. if $b' = \varnothing$ then insert b into the position for b'
8. else if $b' \neq b$ then
9. **reject**
10. **accept**

ALGORITHM 6.2. *Testing* $(\mathbf{A}, \mathbf{B}) \in$ VAR-MEM.

THEOREM 6.5. VAR-MEM^1 *and* VAR-EQUIV^1 *lie in* **PSPACE**.

Proof. Algorithm 6.1 can be used to test whether $(\mathbf{A}, \mathbf{B})$ lies in the *complement* of VAR-MEM^1. Since the algorithm is nondeterministic, we get VAR-MEM$^1 \in$ **co-NPSPACE**. But from Savitch's theorem [26], **NPSPACE** = **PSPACE**. Furthermore, every deterministic class is closed under complements, so **co-PSPACE** = **PSPACE**. Thus VAR-MEM$^1 \in$ **PSPACE**. Now it follows from the relationships in (6.1) that VAR-EQUIV^1 lies in **PSPACE** as well. □

It is not clear whether this is the best possible bound for these two problems. We leave it as an open question.

PROBLEM 6.6. *Are either* VAR-MEM^1 *or* VAR-EQUIV^1 **PSPACE***-complete?*

One might hope to apply the same techniques used above to the unrestricted problem, VAR-EQUIV. Unfortunately, the resources needed to evaluate an arbitrary term in a given algebra jump dramatically as soon as we allow a binary operation. Our approach instead is to try to construct the homomorphism guaranteed by Theorem 6.3 (iii). However, rather than use the construction given in the proof of that theorem, we will construct a homomorphism ψ recursively, based on (6.2). The best we seem to be able to do is the following hyperexponential bound.

THEOREM 6.7. VAR-EQUIV, VAR-MEM $\in$ **2-EXPTIME**.

Proof. Let $|A| = n > 1$ and $|B| = m$. Let F denote the set of basic operation symbols of $\mathbf{A}$ and $\mathbf{B}$, and let r be the maximum rank of the members of F. In light of Theorem 6.5, we shall assume that $r \geq 2$. It follows from Corollary 6.4 that we can test $\mathbf{B} \in \mathbf{V}(\mathbf{A})$ by choosing an arbitrary bijection $\psi_0 \colon X_0 \to B$ and extending it, if possible, to a homomorphism $\psi \colon \mathbf{Clo}_m(\mathbf{A}) \to \mathbf{B}$. So we fix any bijection ψ_0.

Let us sketch an algorithm for extending (if possible) ψ_0 to a homomorphism ψ. Create a table with two columns. In the left-hand column, list every m-ary operation of A. For each m-ary operation g, the right-hand column will contain $\psi(g)$ if and when it is defined. To begin with, leave every entry in the right-hand column blank, except that, for each $1 \leq i \leq m$, put $\psi_0(p_i^m)$ in the row containing p_i^m.

Now let j be an integer, $0 \leq j < n^{(n^m)} - 1$, and suppose we have successfully defined ψ on X_j. We attempt to extend ψ to X_{j+1}. For each $f \in F$ and each $g_1, \ldots, g_k \in X_j$ (with $k = \rho(f)$), we proceed as follows. Look up $\psi(g_1), \ldots, \psi(g_k)$ in the table and compute $b = f^{\mathbf{B}}(\psi(g_1), \ldots, \psi(g_k))$. Also, compute the operation $f^{\mathbf{A}}[g_1, \ldots, g_k]$. If the entry in the table corresponding to $f^{\mathbf{A}}[g_1, \ldots, g_k]$ is blank, then insert b and continue. If the entry is already equal to b, continue. But if the table contains some value $b' \neq b$, then we terminate the algorithm and reject. See the pseudocode in Algorithm 6.2.

If the algorithm runs to completion without a rejection, then we have successfully

found the extension of ψ_0. A rejection means that ψ does not exist.

We must analyze the time requirements of Algorithm 6.2. As we noted earlier, the algebra $\mathbf{A}$ will require at least $n^r \log n$ bits to represent. Let S denote the size of the input $(\mathbf{A}, \mathbf{B})$. Then $S \geq n^r + m^r > r$. Notice also that $S \geq |F|$ since each operation requires at least one bit.

To prove the theorem, we must show that the time required to run Algorithm 6.2 on the input $(\mathbf{A}, \mathbf{B})$ is bounded above by a function of the form $2^{2^{p(S)}}$ for some polynomial p. Since $O\big(2^{2^{\text{poly}}}\big)$ is closed under sums and products, it suffices to check that we can bound the time required by each step of the algorithm by a member of $O\big(2^{2^{\text{poly}}}\big)$. Observe first that

$$\log(n^{m+1}) = (m+1)\log n \leq (m+1)n \leq S,$$

and hence $n^{m+1} \leq 2^S$. Similarly,

$$\log n^{(n^m)} = n^m \log n \leq n^{m+1} \leq 2^S,$$

so

$$n^{(n^m)} \leq 2^{2^S}. \tag{6.3}$$

Let us go step-by-step through Algorithm 6.2. The initial setup (step 1) requires time on the order of n^m to fill in each entry. Since there are $n^{(n^m)}$ entries, the total time required for this step lies in $O\big(2^{2^{\text{poly}}}\big)$ by inequality (6.3). Steps 2–7 constitute a triply-nested loop. The total number of iterations is at most the product of the upper bounds on the counters in steps 2, 3, and 4. Step 2 requires $n^{(n^m)} \in O\big(2^{2^{\text{poly}}}\big)$ iterations. Step 3 runs from 1 to $|F|$, and we have already observed that $|F| \leq S$. For each value of j (in step 2), step 4 will loop $|X_j|^k$ times. We have $k = \rho(f) \leq r$ and $|X_j| \leq |\mathrm{Clo}_m(\mathbf{A})| \leq n^{(n^m)}$, so an upper bound on the counter is $(n^{(n^m)})^r \leq 2^{2^{rS}}$. Finally, the time required for each trip through the body of the triple-loop is the sum of the times for steps 5–7. Step 7 is negligible. Step 5 requires r lookups in the table for ψ and one lookup in the table for $f^{\mathbf{B}}$. The former takes time $r \cdot n^{(n^m)} \leq S \cdot 2^{2^S}$ which is clearly in $O\big(2^{2^{\text{poly}}}\big)$. The latter requires $m^r \leq S$ program steps. Step 6 is the reverse of step 5: one lookup in ψ (time $\leq n^{(n^m)}$) and $(r+1)n^m$ lookups in operation tables. Thus the running time for the entire algorithm lies in $O\big(2^{2^{\text{poly}}}\big)$. □

Theorem 6.7 provides a rather disappointing bound for VAR-EQUIV. It is also somewhat surprising that there seems to be such a dramatic (i.e., exponential) difference in the complexities of QVAR-EQUIV and VAR-EQUIV (see Table 4.1). Perhaps one can do better.

PROBLEM 6.8. *Is* VAR-EQUIV *complete for* **2-EXPTIME**? *Is* VAR-EQUIV *in* **EXPSPACE**?

As we mentioned earlier, Székely proved in [28] that VAR-EQUIV is **NP**-hard. This result can also be obtained using the construction in Theorem 5.5; see especially the equivalences in (5.2).

7. Term-equivalence. Recall that the algebras $\mathbf{A}$ and $\mathbf{B}$ are term-equivalent if $A = B$ and $\mathrm{Clo}(\mathbf{A}) = \mathrm{Clo}(\mathbf{B})$. From a universal algebraic standpoint, term-equivalent algebras are generally interchangeable. When considering the complexity of the problem TERM-EQUIV, it is convenient, once again, to consider a slightly different problem. Thus we define the problem CLO-MEM to consist of all pairs (F, g) in which $F \cup \{g\}$

is a set of operations on some finite set A and $g \in \mathrm{Clo}^A(F)$. The problem CLO-MEM^1 is similar, but we require all of the operations in $F \cup \{g\}$ to be unary. For the problem CLO-MEM$_2$, the set A has cardinality two.

Historically, CLO-MEM was the first problem, of those discussed in this paper, to be considered in the literature. Kozen proved in 1977 [18] that CLO-MEM^1 is complete for **PSPACE**. In 1982, Friedman proved that CLO-MEM is complete for **EXPTIME** [8]. However, that manuscript was never published. A different proof of this result, as well as a proof that TERM-EQUIV is complete for **EXPTIME**, appears in [1].

Let $\mathbf{A} = \langle A, F\rangle$ and $\mathbf{B} = \langle B, G\rangle$. Then

$$\begin{array}{c} (\mathbf{A}, \mathbf{B}) \in \text{TERM-EQUIV} \\ \Updownarrow \\ A = B \;\&\; (\forall g \in G)(\forall f \in F)\ \big((F, g), (G, f) \in \text{CLO-MEM}\big). \end{array} \tag{7.1}$$

Conversely, if $F \cup \{g\}$ is a set of operations on A, then

$$(F, g) \in \text{CLO-MEM} \iff \big(\langle A, F\rangle, \langle A, F \cup \{g\}\rangle\big) \in \text{TERM-EQUIV}. \tag{7.2}$$

Of course, similar relationships hold for the unary and two-element variants of these problems.

Now it follows easily from (7.2) that CLO-MEM is log-space reducible to TERM-EQUIV. This is true for the general, unary, and two-element variants of the problems. However, a reduction in the other direction is a bit problematic. Let us first consider the general case. Given a pair $(\mathbf{A}, \mathbf{B})$ of size S, equivalence (7.1) tells us that we can test $(\mathbf{A}, \mathbf{B}) \in$ TERM-EQUIV by making several calls to an algorithm for CLO-MEM. The input to each such call will certainly have size at most S hence will run in time at most $2^{p(S)}$ for some polynomial p. Furthermore, there will clearly be at most S such calls. Hence a bound on the running time for TERM-EQUIV will be $S \cdot 2^{p(S)}$ which is still exponential in S. Combining our observations, we conclude that the (general) problem TERM-EQUIV is complete for **EXPTIME** (see Table 4.1).

For CLO-MEM^1 and CLO-MEM$_2$, we need to argue a bit differently, since we will be interested in a space-bound. Let C denote one of these two problems, and let T denote the corresponding term-equivalence problem. Suppose we have an algorithm for C that runs in space $f(x)$ on an input of size x. As is commonplace, we assume that f is a monotonically increasing function. In applying (7.1) to test $(\mathbf{A}, \mathbf{B}) \in$ T, the first call to C will require space bounded above by $O(f(S))$. But subsequent calls to C can reuse the same space. Hence the total space requirement for T is on the order of $\log S + f(S)$. (The $\log S$ term accounts for some counters.)

For the specific case C = CLO-MEM^1, Kozen's result tells us that the function f is a polynomial, so we conclude that TERM-EQUIV^1 is complete for **PSPACE**, as we assert in Table 4.1. When $C =$ CLO-MEM$_2$, we can certainly make the following claim.

LEMMA 7.1. *If $f(S) \in O(\log S)$, then both* CLO-MEM$_2$ *and* TERM-EQUIV$_2$ *lie in* **NL**.

So our remaining task is to prove that $f(S)$ is indeed on the order of $\log S$. In other words, we must find a nondeterministic algorithm for CLO-MEM$_2$ that runs in log-space. To do this, we will take advantage of the very detailed description of the lattice of clones on $\{0, 1\}$ that was discovered by Post in 1941 [24]. This will allow us to check the condition $(F, g) \in$ CLO-MEM$_2$ using a finite number of nondeterministic tests, each of which uses very little space. There have been several more recent

treatments of Post's results. The reader might wish to consult the first chapter of Pippenger's book [23]. All we really need here is a description of the completely meet-irreducible members of the lattice of clones. For this we mostly follow the discussion given in Szendrei [30, pp. 36ff].

Let A be any set, and let k be a positive integer. A subset θ of A^k is called a k-ary *relation on* A. If f is an n-ary operation on A, we say that f *preserves* θ, ($f \mid\!: \theta$) if for all arrays $\langle\, a_{ij} : 1 \le i \le n,\ 1 \le j \le k \,\rangle$ we have

$$(\forall i \le n\ \ \mathbf{a}^i \in \theta) \implies \langle f(\mathbf{a}_1), \ldots, f(\mathbf{a}_k)\rangle \in \theta,$$

where $\mathbf{a}_j = \langle a_{1j}, \ldots, a_{nj}\rangle$ and $\mathbf{a}^i = \langle a_{i1}, \ldots, a_{ik}\rangle$. The relationship $f \mid\!: \theta$ is equivalent to asserting that θ is a subalgebra of the algebra $\langle A, f\rangle^k$. One way to visualize this is to think of the $\langle a_{ij}\rangle$ as forming an $n \times k$ matrix. If each row of the matrix constitutes an element of θ, then the row obtained by applying f to each column is again a member of θ. We extend this notation to multiple operations by defining $F \mid\!: \theta$ if and only if for every $f \in F$ we have $f \mid\!: \theta$.

For a fixed relation θ we define

$$\mathcal{F}^A(\theta) = \{\, f : f \text{ is an operation on } A \text{ and } f \mid\!: \theta \,\}.$$

More generally, if Θ is a set of relations on A (of various ranks), then

$$\mathcal{F}^A(\Theta) = \bigcap_{\theta \in \Theta} \mathcal{F}^A(\theta).$$

We usually omit the superscript "A" if no confusion will result. It is easy to check that for any set A and family Θ, $\mathcal{F}(\Theta)$ is always a clone on A. However, more to the point for us, for every *finite* set A, every clone on A is of the form $\mathcal{F}(\Theta)$ for some (not necessarily finite) family Θ. See [30, Corollary 1.4].

We now restrict our attention to $A = \{0, 1\}$. We define the following relations on A.

$$\begin{aligned}
\nu &= \{\, \langle 0,1\rangle, \langle 1,0\rangle \,\}, & \lambda &= \{\, \langle 0,0\rangle, \langle 0,1\rangle, \langle 1,1\rangle \,\},\\
\mu &= \{\, \langle x,y,z\rangle : z = x \wedge y \,\}, & \chi &= \{\, \langle x,y,z\rangle : z = x \vee y \,\},\\
\varepsilon &= \{\, \langle x,y,z\rangle : x = y \text{ or } y = z \,\}, & \sigma &= \{\, \langle x,y,z,w\rangle : x \oplus y = z \oplus w \,\},\\
\kappa_m &= \{0,1\}^m - \{\langle 1,1,\ldots,1\rangle\} \quad \text{for } m \ge 1,\\
\tilde{\kappa}_m &= \{0,1\}^m - \{\langle 0,0,\ldots,0\rangle\} \quad \text{for } m \ge 1.
\end{aligned}$$

Let $\Sigma_0 = \{\nu, \mu, \chi, \lambda, \varepsilon, \sigma\}$ and, for every $m > 0$, $\Sigma_m = \Sigma_0 \cup \{\, \kappa_j, \tilde{\kappa}_j : j \le m \,\}$. Set $\Sigma = \bigcup_{m \ge 1} \Sigma_m$. From Post's analysis, we get the following theorem.

THEOREM 7.2 (Post). *Let C be a clone on $\{0,1\}$. Then for some family $\Theta \subseteq \Sigma$, $C = \mathcal{F}(\Theta)$.*

Let θ be a fixed k-ary relation on $\{0,1\}$. We first show that there is a simple nondeterministic algorithm for the *complement* of the problem

$$\textsc{Pres}(\theta) = \{\, f : f \text{ an operation on } \{0,1\} \text{ and } f \mid\!: \theta \,\}.$$

Given an n-ary operation f, we guess an $n \times k$ matrix and accept if each row of the matrix lies in θ but the result of applying f to the columns fails to lie in θ. A more formal description is given in Algorithm 7.1.

LEMMA 7.3. *For any fixed relation θ on $\{0,1\}$, $\textsc{Pres}(\theta) \in \mathbf{NL}$.*

1. Let $n = \rho(f)$
2. Guess $a_{ij} \in \{0,1\}$, $1 \le i \le n$, $1 \le j \le k$
3. For $i = 1$ to n do
4. if $\mathbf{a}^i \notin \theta$ then **reject**
5. if $\langle f(\mathbf{a}_1), \ldots, f(\mathbf{a}_k)\rangle \notin \theta$ then **accept**

ALGORITHM 7.1. $f \notin \text{PRES}(\theta)$.

---

1. Let $n = \rho(f)$
2. $\mathbf{x} \leftarrow \langle 1, 1, \ldots, 1\rangle \in \{0,1\}^n$
3. for $i = 1$ to m do
4. guess $\mathbf{a} \in \{0,1\}^n$
5. if $f(\mathbf{a}) = 1$ then $\mathbf{x} \leftarrow \mathbf{x} \wedge \mathbf{a}$ *(coordinatewise conjunction)*
6. else **reject**
7. if $\mathbf{x} = \langle 0, 0, \ldots, 0\rangle$ then **accept**
8. else **reject**

ALGORITHM 7.2. $(f, m) \notin \text{KAPPA}$.

Proof. Algorithm 7.1 provides a test for the complement of PRES(θ). Since **NL** is closed under complements (see [12, 29]), it suffices to show that this algorithm requires only log-space. Since the size of the input (an n-ary operation) is 2^n bits, we must verify that the space requirement of the algorithm lies in $O(n)$. However, the algorithm only requires space for the counters i and j, the Boolean matrix $\langle a_{ij}\rangle$, and k pointers into the table of values for f. Since k is a constant, the total space required is indeed in $O(n)$. □

We will also need a variation of Algorithm 7.1 to handle the κ_m and $\tilde{\kappa}_m$. Algorithm 7.2 gives a procedure that takes as input an operation f and a positive integer m and accepts (nondeterministically) if and only if f does not preserve κ_m. Using the same argument as in Lemma 7.3, we can conclude that

$$\text{KAPPA} = \{\, (f, m) : f \not\colon \kappa_m \,\} \in \mathbf{NL},$$

with an analogous result for $\tilde{\kappa}_m$. Let us remark here that it is not hard to modify Algorithm 7.1 to run in deterministic logarithmic space. However, that modification does not seem to work for KAPPA, so there does not appear to be anything to gain by including the argument here.

Our next task is to put a bound on the size of the family Θ that we need to consider in Theorem 7.2. Notice that, except for the six members of Σ_0, the members of Σ form two infinite sequences, and, furthermore, these two sequences are dual to each other.

LEMMA 7.4. *Let g be an n-ary operation on $\{0,1\}$.*

(i) *For every $m > 1$, if g preserves κ_m, then g preserves κ_{m-1}.*

(ii) *If g preserves κ_n, then g preserves κ_m for all $m > n$.*

The same results hold with $\tilde{\kappa}$ in place of κ.

Proof. Suppose $g \not\colon \kappa_m$. Let $\langle a_{ij}\rangle$ be an $n \times (m-1)$ matrix in which every row is a member of κ_{m-1}. This simply means that no row consists of all 1s. We wish to argue that $\langle g(\mathbf{a}_1), \ldots g(\mathbf{a}_{m-1})\rangle \in \kappa_{m-1}$, i.e., is not all 1s. Create an $n \times m$ matrix by repeating the last column of $\langle a_{ij}\rangle$. Then each row of this new matrix lies in κ_m, so by assumption, the m-tuple $\langle g(\mathbf{a}_1), \ldots, g(\mathbf{a}_{m-1}), g(\mathbf{a}_{m-1})\rangle \in \kappa_m$. Therefore, $\langle g(\mathbf{a}_1), \ldots, g(\mathbf{a}_{m-1})\rangle$ is not equal to $\langle 1, 1, \ldots, 1\rangle$, as desired.

For the second claim, assume that $g \not\colon \kappa_n$ and let $m > n$. Let $\langle a_{ij}\rangle$ be an $n \times m$

```
1. n ← ρ(g)
2. for θ ∈ Σ_n do
3.     if F |: θ then
4.         if not(g |: θ) then reject
5. accept
```

ALGORITHM 7.3. $g \in \mathrm{Clo}(F)$.

matrix in which every row is a member of κ_m. In other words, each row contains a "0." Therefore we can find $1 \le j_1 < j_2 < \cdots < j_n \le m$ such that the $n \times n$ submatrix $\langle a_{ij_k} : 1 \le i \le n,\ 1 \le k \le n\rangle$ retains the property that every row contains a "0." Since $g \mid\!: \kappa_n$, we get $\langle g(\mathbf{a}_{j_1}), \ldots, g(\mathbf{a}_{j_n})\rangle \in \kappa_n$, i.e., for some k, $g(\mathbf{a}_{j_k}) = 0$. Therefore, $\langle g(\mathbf{a}_1), \ldots, g(\mathbf{a}_n)\rangle \in \kappa_m$. □

We can now combine Theorem 7.2 and Lemmas 7.3 and 7.4 to put a finite bound on the work needed in order to test the relationship $g \in \mathrm{Clo}(F)$.

THEOREM 7.5. *Let F be a set of operations on $\{0,1\}$, and let g be an n-ary operation on $\{0,1\}$. Then $g \in \mathrm{Clo}(F)$ if and only if for every $\theta \in \Sigma_n$, $F \mid\!: \theta \implies g \mid\!: \theta$.*

Proof. Suppose first that $g \in \mathrm{Clo}(F)$. Let θ be any relation such that $F \mid\!: \theta$. Since $\mathcal{F}(\theta)$ is a clone, $F \subseteq \mathcal{F}(\theta)$, and $\mathrm{Clo}(F)$ is, by definition, the *smallest* clone containing F, we obtain $\mathrm{Clo}(F) \subseteq \mathcal{F}(\theta)$; hence $g \mid\!: \theta$.

Conversely, suppose the condition holds. By Theorem 7.2, there is some subset Θ of Σ such that $\mathrm{Clo}(F) = \mathcal{F}(\Theta)$. We need to show that for every $\theta \in \Theta$, $g \mid\!: \theta$. If $\theta \in \Sigma_n$, then, since $F \subseteq \mathrm{Clo}(F)$, we have $F \mid\!: \theta$, so $g \mid\!: \theta$ by assumption. So suppose that $\theta \in \Sigma - \Sigma_n$. Then $\theta = \kappa_m$ or $\theta = \tilde{\kappa}_m$ for some $m > n$. Without loss of generality, assume the former. Again, since $F \subseteq \mathrm{Clo}(F) = \mathcal{F}(\Theta)$, we get $F \mid\!: \kappa_m$. By Lemma 7.4(i), $F \mid\!: \kappa_n$; hence $g \mid\!: \kappa_n$ by assumption. Therefore, by Lemma 7.4(ii), $g \mid\!: \kappa_m$. □

Since the family Σ_n is finite, Theorem 7.5 provides an algorithm for testing $g \in \mathrm{Clo}(F)$. Pseudocode is given in Algorithm 7.3. Let us analyze the space requirements for this algorithm. The size of the input is at least 2^n since that is the size of g, and also at least $|F|$ since each operation takes up at least 1 bit. Line 1 can be implemented by setting θ first to each member of Σ_0, followed by $\kappa_1, \tilde{\kappa}_1, \kappa_2, \ldots, \tilde{\kappa}_n$ using a loop. The amount of space needed to manage the loop is in $O(\log n)$.

The notation "$F \mid\!: \theta$" is shorthand for the following code fragment.

```
x ← True
for f ∈ F do
    if not(f |: θ) then x ← False
return x
```

When $\theta \in \Sigma_0$, the test $f \mid\!: \theta$ is implemented using an algorithm for PRES(θ). When $\theta = \kappa_m$ or $\tilde{\kappa}_m$, we use KAPPA or its dual. Clearly, the entire algorithm requires only a couple of counters besides the calls to PRES and KAPPA and hence lies in **NL**. Combining this with Lemma 7.1, we have proved the following theorem.

THEOREM 7.6. *Both* CLO-MEM$_2$ *and* TERM-EQUIV$_2$ *lie in* **NL**.

Once again, it is not clear that these are optimal results, so we pose a problem.

PROBLEM 7.7. *Is* TERM-EQUIV$_2$ *complete for* **NL**? *Does it lie in* **L**?

Appendix. Proof of Theorem 6.1. This appendix provides a proof of Theorem 6.1. In fact, we shall prove a slightly stronger theorem which allows us to conclude that both VAR-EQUIV$_2$ and VAR-MEM$_2$ are members of **L**. The line of argument we

follow was suggested to us by J. Berman. An alternate proof can be obtained from [15, Corollary 2.5], using the fact that every two-element algebra is both term-minimal and strictly simple.

An algebra is called *congruence distributive* if its lattice of congruences is distributive. The algebra is *congruence permutable* if for every pair of congruences α and β we have $\alpha \circ \beta = \beta \circ \alpha$. Here,

$$\alpha \circ \beta = \{\, (x, z) : (\exists y)\ (x, y) \in \alpha \ \&\ (y, z) \in \beta \,\}.$$

A variety is called congruence distributive (respectively, permutable) if every member is congruence distributive (permutable). See [5, Definition 5.8] for a discussion of these two important properties. We also require the following result.

LEMMA A.1 (see Berman [2, Lemma 1]). *Let* $\mathbf{A}$ *be an algebra with universe* $\{0, 1\}$. *Then at least one of the following hold.*

(i) $\mathbf{V}(\mathbf{A})$ *is congruence distributive.*

(ii) $\mathbf{V}(\mathbf{A})$ *is congruence permutable.*

(iii) *Every basic operation of* $\mathbf{A}$ *is one of the following: a conjunction of variables, a disjunction of variables, negation of a variable, a projection operation, or a constant operation.*

An n-ary operation f is "a conjunction of variables" if $f(x_1, \dots, x_n) = x_{i_1} \wedge x_{i_2} \wedge \cdots \wedge x_{i_k}$, where $2 \le k \le n$, $1 \le i_1 < i_2 < \cdots < i_k \le n$ and "$\wedge$" denotes the usual logical "and" operation on $\{0, 1\}$. A disjunction of variables is similar. We write $0' = 1$ and $1' = 0$. The function $f(x_1, \dots, x_n) = x_i'$ is the negation of the variable x_i.

Let Δ denote the equivalence relation on $\{0, 1\}^2$ that identifies the pairs $(0, 0)$ and $(1, 1)$ and the pairs $(0, 1)$ and $(1, 0)$. One way to think of Δ is as the kernel of the exclusive-or function $\{0, 1\}^2 \to \{0, 1\}$. For a fixed algebra $\mathbf{A}$ on $\{0, 1\}$, Δ may or may not be a congruence relation on $\mathbf{A}^2$. By using Lemma A.1, it is not hard to see that Δ is a congruence if and only if $\mathbf{A}$ is Abelian. (See [20, section 4.13], especially Theorem 4.152.) In the case that Δ is indeed a congruence on $\mathbf{A}^2$, we define $\mathbf{A}_\nabla$ to be the algebra $\mathbf{A}^2/\Delta$. Note that it is possible for $\mathbf{A}$ and $\mathbf{A}_\nabla$ to be isomorphic. In fact, $\mathbf{A} \cong \mathbf{A}_\nabla$ if and only if there is an element $e \in \{0, 1\}$ such that $\{e\}$ is a subalgebra of $\mathbf{A}$. (To see this, show that the function $x \mapsto (x, e)/\Delta$ is an isomorphism.) As we shall see, in the case that $\mathbf{A}_\nabla$ exists and is not isomorphic to $\mathbf{A}$, $\mathbf{V}(\mathbf{A}_\nabla)$ is the unique proper, nontrivial subvariety of $\mathbf{V}(\mathbf{A})$.

THEOREM A.2. *Let* $\mathbf{A}$ *and* $\mathbf{B}$ *be algebras on* $\{0, 1\}$ *of the same similarity type, and suppose that* $\mathbf{B} \in \mathbf{V}(\mathbf{A})$. *Then either* $\mathbf{B} \cong \mathbf{A}$ *or* $\mathbf{B} \cong \mathbf{A}_\nabla$.

Proof. Since both $\mathbf{A}$ and $\mathbf{B}$ have cardinality two, they are *strictly simple* algebras, i.e., they are simple and have no proper, nontrivial subalgebras. Suppose first that $\mathcal{V} = \mathbf{V}(\mathbf{A})$ is congruence distributive. Since $\mathbf{B} \in \mathbf{V}(\mathbf{A})$ and $\mathbf{B}$ is simple, we obtain $\mathbf{B} \in \mathbf{HS}(\mathbf{A})$ from Jónsson's lemma [5, Theorem 6.8]. From the strict simplicity of $\mathbf{A}$ we get $\mathbf{B} \cong \mathbf{A}$. Notice that we never obtain the conclusion $\mathbf{B} \cong \mathbf{A}_\nabla$ here. This is not surprising since a congruence distributive variety contains no nontrivial Abelian algebras.

For the remainder of the proof, we assume that $\mathcal{V}$ is not congruence distributive. Suppose that $\mathcal{V}$ is congruence permutable. Then, in particular, $\mathcal{V}$ is congruence modular [5, Theorem 5.10], so by [7, Theorem 12.1] and the strict simplicity of $\mathbf{A}$, $\mathcal{V}$ is an Abelian variety, and hence $\mathbf{A}$ is an Abelian algebra. Therefore, since $\mathbf{B} \in \mathbf{V}(\mathbf{A})$, from [7, Theorem 12.4] we deduce that either $\mathbf{B} \cong \mathbf{A}$ or $\mathbf{B} \cong \mathbf{A}_\nabla$.

So now we are reduced to the case that $\mathcal{V}$ is neither congruence distributive nor congruence permutable. This forces us into case (iii) of Lemma A.1. Each of the

TABLE A.1

$\mathbf{A}$	$\mathbf{A}_\nabla$
$\langle\{0,1\}, '\rangle$	$\langle\{0,1\}, p_1^1\rangle$
$\langle\{0,1\}, ', 0\rangle$	$\langle\{0,1\}, p_1^1, 0\rangle$
$\langle\{0,1\}\rangle$	
$\langle\{0,1\}, 0\rangle$	
$\langle\{0,1\}, 1\rangle$	
$\langle\{0,1\}, 0, 1\rangle$	$\langle\{0,1\}, 0, 0\rangle$

basic operations of $\mathbf{A}$ and of $\mathbf{B}$ must be of one of the types described there. Suppose that $\mathbf{A}$ contains a conjunction of variables among its basic operations. If $\mathbf{A}$ also contains either a disjunction of variables or a negation operation, then $\mathbf{A}$ will have a lattice reduct and consequently $\mathcal{V}$ will be congruence distributive, which contradicts our assumption. Therefore, if $\mathbf{A}$ contains a conjunction of variables, then $\mathbf{A}$ is term-equivalent to a semilattice, possibly with one or two constant operations. However, it is well known that the variety generated by such an algebra has, up to isomorphism, a unique two-element member. Thus we conclude that in this case too, $\mathbf{B} \cong \mathbf{A}$. Obviously, we arrive at the same conclusion if we assume at the outset that $\mathbf{A}$ has a basic operation that is a disjunction of variables.

Now suppose that neither $\mathbf{A}$ nor $\mathbf{B}$ has a conjunction or a disjunction of variables among its basic operations. By looking at the available operations in Lemma A.1 (iii), we conclude that $\mathbf{A}$ is term-equivalent to one of the algebras listed in Table A.1. It is easy to check that for each algebra $\mathbf{A}$ in the table, $\mathbf{A}_\nabla$ exists. Furthermore, in each case, it is obvious that $\mathbf{A}$ and $\mathbf{A}_\nabla$ are the only two-element algebras in $\mathbf{V}(\mathbf{A})$. Using these observations, we obtain the conclusion in the remaining cases. □

Theorem 6.1 follows immediately from Theorem A.2. For if $\mathbf{B}$ and $\mathbf{A}$ generate the same variety, then we certainly have $\mathbf{B} \in \mathbf{V}(\mathbf{A})$, so by Theorem A.2, either $\mathbf{B} \cong \mathbf{A}$ or $\mathbf{B} \cong \mathbf{A}_\nabla$. However, we claim that if $\mathbf{A} \ncong \mathbf{A}_\nabla$, then $\mathbf{A}_\nabla$ always generates a proper subvariety of $\mathbf{V}(\mathbf{A})$, contrary to the assumption that $\mathbf{V}(\mathbf{B}) = \mathbf{V}(\mathbf{A})$. The easiest way to see this is probably just to treat the various cases that arose in Theorem A.2 individually. The case that $\mathbf{V}(\mathbf{A})$ is congruence permutable is discussed in [7]. For the algebras listed in Table A.1, it is a simple matter to find an identity satisfied by $\mathbf{A}_\nabla$ that fails in $\mathbf{A}$. For example, consider the first line of the table. The similarity type consists of a single unary operation symbol, f. Then the identity $f(x) \approx x$ separates $\mathbf{A}$ from $\mathbf{A}_\nabla$.

COROLLARY A.3. VAR-MEM$_2 \in \mathbf{L}$.

Proof. From Theorem A.2, $(\mathbf{A}, \mathbf{B}) \in$ VAR-MEM$_2$ if and only if either $\mathbf{B} \cong \mathbf{A}$ or $\mathbf{B} \cong \mathbf{A}_\nabla$. We have already observed that the former condition can be checked in log-space. To check the latter, it is enough to check the following two maps to see if either is a homomorphism.

$$\mathbf{A}^2 \to \mathbf{B} \qquad (x, y) \mapsto x \oplus y,$$
$$\mathbf{A}^2 \to \mathbf{B} \qquad (x, y) \mapsto x \oplus y \oplus 1.$$

Here $x \oplus y$ is the exclusive-or of x and y. □

Acknowledgments. Finally, we would like to thank Joel Berman, Gary Leavens, and Ross Willard for many helpful discussions on this and related topics.

REFERENCES

[1] C. Bergman, D. Juedes, and G. Slutzki, *Computational complexity of term-equivalence*, Internat. J. Algebra Comput., 9 (1999), pp. 113–128.

[2] J. Berman, *A proof of Lyndon's finite basis theorem*, Discrete Math., 29 (1980), pp. 229–233.

[3] G. Birkhoff, *On the structure of abstract algebras*, Proc. Cambr. Philos. Soc., 31 (1935), pp. 433–454.

[4] S. Burris, *Computers and universal algebra: some directions*, Algebra Universalis, 34 (1995), pp. 61–71.

[5] S. Burris and H. P. Sankappanavar, *A Course in Universal Algebra*, Springer-Verlag, New York, 1981.

[6] H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification* 1, EATCS Monographs on Theoretical Computer Science 6, Springer-Verlag, Berlin, 1985.

[7] R. Freese and R. McKenzie, *Commutator Theory for Congruence Modular Varieties*, London Math. Soc. Lecture Notes Ser. 125, Cambridge University Press, Cambridge, UK, 1987.

[8] H. Friedman, *Function Composition and Intractability* I, Manuscript, 1982.

[9] M. Garey and D. Johnson, *Computers and Intractability—A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co., San Francisco, CA, 1979.

[10] G. Grätzer, *Universal Algebra*, 2nd ed., Springer-Verlag, New York, 1979.

[11] Z. Hedrlín and A. Pultr, *On full embeddings of categories of algebras*, Illinois J. Math., 10 (1966), pp. 392–405.

[12] N. Immerman, *Nondeterministic space is closed under complementation*, SIAM J. Comput., 17 (1988), pp. 935–938.

[13] D. Johnson, *A catalog of complexity classes*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 69–161.

[14] J. Kalicki, *On comparison of finite algebras*, Proc. Amer. Math. Soc., 3 (1952), pp. 36–40.

[15] K. Kearnes and Á. Szendrei, *A characterization of minimal locally finite varieties*, Trans. Amer. Math. Soc., 349 (1997), pp. 1749–1768.

[16] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser Boston, Cambridge, MA, 1993.

[17] D. Kozen, *Complexity of finitely presented algebras*, in Proceedings of the Ninth Annual Symposium on the Theory of Computing, ACM, New York, 1977, pp. 164–177.

[18] D. Kozen, *Lower bounds for natural proof systems*, in 18th Annual Symposium on Foundations of Computer Science, Providence, RI, IEEE Computer Society, Los Alamitos, CA, 1977, pp. 254–266.

[19] L. Kučera and V. Trnková, *The computational complexity of some problems in universal algebra*, in Universal Algebra and its Links with Logic, Algebra, Combinatorics and Computer Science, P. Burmeister et al., eds., Heldermann Verlag, Berlin, 1984, pp. 261–289.

[20] R. McKenzie, G. McNulty, and W. Taylor, *Algebras, Lattices, Varieties*, I, Wadsworth and Brooks/Cole, Monterey, CA, 1987.

[21] G. L. Miller, *Graph isomorphism: General remarks*, J. Comput. System Sci., 18 (1979), pp. 128–142.

[22] C. H. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.

[23] N. Pippenger, *Theories of Computability*, Cambridge University Press, Cambridge, UK, 1997.

[24] E. Post, *Two-Valued Iterative Systems of Mathematical Logic*, Princeton University Press, Princeton, NJ, 1941.

[25] D. Sanella and A. Tarlecki, *On observational equivalence and algebraic specification*, J. Comput. System Sci., 34 (1987), pp. 150–178.

[26] W. J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. System Sci., 4 (1970), pp. 177–192.

[27] M. Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, Boston, MA, 1997.

[28] Z. Székely, *Complexity of the Finite Algebra Membership Problem for Varieties*, Ph.D. thesis, University of South Carolina, Columbia, SC, 1998.

[29] R. Szelepcsényi, *The method of forced enumeration for nondeterministic automata*, Acta Inform., 26 (1988), pp. 279–284.

[30] Á. Szendrei, *Clones in Universal Algebra*, Séminaire de Mathématiques Supérieures, L'Université de Montréal, Montréal, Canada, 1986.

[31] W. Taylor, *Equational logic*, Houston J. Math. Survey, (1979), pp. iii+83.

[32] I. Wegener, *The Complexity of Boolean Functions*, John Wiley and Sons, Chichester, UK, 1987.

[33] J. Wing, *A specifier's introduction to formal methods*, Computer, 23 (1990), pp. 8–24.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 383–390

AN ALGORITHM FOR HEILBRONN'S PROBLEM*

CLAUDIA BERTRAM–KRETZBERG†, THOMAS HOFMEISTER†, AND HANNO LEFMANN†

Abstract. Heilbronn conjectured that given arbitrary n points from the 2-dimensional unit square, there must be three points which form a triangle of area at most $O(1/n^2)$. This conjecture was disproved by a nonconstructive argument of Komlós, Pintz, and Szemerédi [*J. London Math. Soc.*, 25 (1982), pp. 13–24] who showed that for every n there is a configuration of n points in the unit square where all triangles have area at least $\Omega(\log n/n^2)$. Considering a discretization of Heilbronn's problem, we give an alternative proof of the result from [*J. London Math. Soc.*, 25 (1982), pp. 13–24]. Our approach has two advantages: First, it yields a polynomial-time algorithm which for every n computes a configuration of n points where all triangles have area $\Omega(\log n/n^2)$. Second, it allows us to consider a generalization of Heilbronn's problem to convex hulls of k points where we can show that an algorithmic solution is also available.

Key words. hypergraphs, independent sets, Heilbronn, triangles

AMS subject classifications. 68Q25, 05C65, 52C99

PII. S0097539798348870

1. Introduction.

Given a configuration of n points in the 2-dimensional unit square, we can consider the minimum triangle area formed by any three of the n points. Heilbronn's problem consists in finding a configuration of n points which maximizes this minimum area. Denote the maximal possible area by $\Delta^{(3)}(n)$. Heilbronn conjectured that $\Delta^{(3)}(n) = O(1/n^2)$, and the earliest reference mentioning the conjecture is a paper by Roth [12] from 1951; see also the 1976 survey paper [16].

In 1982, Komlós, Pintz, and Szemerédi [9] showed nonconstructively that Heilbronn's conjecture is false. They proved the lower bound $\Delta^{(3)}(n) = \Omega(\log n/n^2)$. The exact order of $\Delta^{(3)}(n)$ is still unknown, but some upper bounds are known. The best one is by Komlós, Pintz, and Szemerédi [8] who showed that $\Delta^{(3)}(n) = O(e^{c\sqrt{\log n}}/n^{8/7})$ for some constant $c > 0$, i.e., $\Delta^{(3)}(n) = O(n^{-8/7+\epsilon})$ for any fixed $\varepsilon > 0$. This result improved upon earlier upper bounds due to Roth [12], [13], [14], [15], [16] and Schmidt [17].

In this paper, we provide a constructive proof of the result by Komlós, Pintz, and Szemerédi. We provide a polynomial-time algorithm which for every n computes a configuration of n points in the unit square where the minimum triangle area is at least $\Omega(\log n/n^2)$. This is done by first discretizing the problem and then transforming it into an independent set problem on hypergraphs.

Designing an algorithm for Heilbronn's problem is interesting because the existence proof of [9] applies a probabilistic argument which uses a continuous distribution for choosing the points in the unit disc (or square) and it involves the evaluation of

*Received by the editors December 10, 1998; accepted for publication (in revised form) October 10, 1999; published electronically June 3, 2000. This research was supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center *Computational Intelligence* (SFB 531). A preliminary version of this paper appeared in *Proceedings of the* 3*rd Annual International Computing and Combinatorics Conference (COCOON'97)*, Lecture Notes in Comput. Sci. 1276, pp. 23–31.
http://www.siam.org/journals/sicomp/30-2/34887.html

†Lehrstuhl Informatik 2, Universität Dortmund, Dortmund 44221, Germany (c.bertram-kretzberg@douglas-informatik.de, hofmeist@ls2.informatik.uni-dortmund.de, lefmann@ls2.informatik.uni-dortmund.de).

integrals. This makes a direct derandomization of the probabilistic argument difficult. To overcome this difficulty, we first discretize the problem. More precisely, we will consider a $T\times T$ grid, where T is of the order of n^α for some $\alpha > 1$. One of the problems introduced by this discretization is that now many triples of grid points lie on a line and form a degenerated triangle (i.e., one of area zero), whereas in the continuous case, the probability of this happening for randomly chosen points is zero.

For every n, our algorithm will find n points in the $T\times T$ grid where every triangle has area at least $\Omega(T^2 \log n/n^2)$. This yields a solution for the problem in the unit square by scaling with a factor of $\Theta(1/T^2)$.

Furthermore, we demonstrate that our approach can be used for a more general problem, first investigated for the special case $k = 4$ by Schmidt [17]. Given a configuration of n points in the unit square, what is the minimum area of the convex hull of k points? Choose n points to maximize this minimum area which we denote by $\Delta^{(k)}(n)$.

It seems that the general case of fixed $k \geq 4$ has not been investigated in the existing literature. We remark that in [5, p. 861] (without proof) the lower bound $\Delta^{(k)}(n) = \Omega(n^{-1-1/k})$ is attributed to Schmidt. However, Schmidt considered in [17] only the case $k = 4$ and proved by analytic arguments that $\Delta^{(4)}(n) = \Omega(1/n^{3/2})$. Here we give a simpler proof of this lower bound. Moreover, our arguments yield a polynomial-time algorithm which for arbitrary fixed $k \geq 4$ computes n points in the unit square where the minimum area of the convex hull of any k points is $\Omega(1/n^{(k-1)/(k-2)})$, i.e., $\Delta^{(k)}(n) = \Omega(1/n^{(k-1)/(k-2)})$. It should be noted (see [17]) that for $k \geq 4$ it is not known whether $\Delta^{(k)}(n) = o(1/n)$.

A generalization of Heilbronn's problem to d dimensions which considers the volumes of $(d + 1)$-simplices has recently been investigated in [2] and [10].

2. Hypergraphs. Heilbronn's problem can be transformed into a problem on hypergraphs. We first recall some basic definitions.

DEFINITION 1. *A hypergraph $\mathcal{G} = (V, \mathcal{E})$ is given by a finite nonempty set V of vertices and a set $\mathcal{E} = \{E_1, \ldots, E_r\}$ of hyperedges $E_i \subseteq V$ with $|E_i| \geq 2$ for $i = 1, \ldots, r$. A hypergraph is called k-uniform if every hyperedge has cardinality k. A subset $V' \subseteq V$ is called independent if no hyperedge E_i is a subset of V'. The average degree of a k-uniform hypergraph is defined by $d_{avg} = |\mathcal{E}| \cdot k/|V|$.*

2-uniform hypergraphs capture the usual concept of undirected simple graphs. For certain types of graphs, e.g., graphs without cycles of small length, very often better estimations on parameters and better algorithms are available than for arbitrary graphs. For hypergraphs, the same is sometimes true. We use the following cycle definition in hypergraphs.

DEFINITION 2. *A 2-cycle in a hypergraph is a pair of hyperedges $E_1 \neq E_2$ such that $|E_1 \cap E_2| \geq 2$.*

Such a pair E_1, E_2 can be seen as a cycle since if $\{v, w\} \subseteq E_1 \cap E_2$, then one can reach w from v via E_1 and return to v via E_2.

We consider the points from the $T\times T$ grid as the vertices of a 3-uniform hypergraph $\mathcal{H}^{(3)}_{T,L}$ whose hyperedges are determined by the parameters T and L as follows. A triple $\{p, q, r\}$ is a hyperedge of $\mathcal{H}^{(3)}_{T,L}$ if the (possibly degenerated) triangle on p, q, r in the $T\times T$ grid has area at most L. We will later choose some $L = \Theta(T^2 \log n/n^2)$.

If V' is an independent set in the hypergraph $\mathcal{H}^{(3)}_{T,L}$, then the corresponding points in the $T\times T$ grid have the property that every triangle has an area bigger than L. It remains to find an independent set V' in $\mathcal{H}^{(3)}_{T,L}$ of size at least $\Omega(n)$.

Computing maximum independent sets in graphs is $\mathcal{NP}$-hard, and it is also hard to find an approximate solution in polynomial time, unless $\mathcal{P}$=$\mathcal{NP}$; see Håstad [6]. The same holds for hypergraphs; see, e.g., [7]. Nevertheless, polynomial-time algorithms for computing independent sets of a guaranteed size are available. We apply such an algorithm, which is based on derandomization through potential functions. For this approach to work, it is crucial to determine the number of hyperedges, i.e., the number of triangles of area at most L in the $T\times T$ grid. Using nothing more but such an upper bound on the number of hyperedges, this approach could be used for a choice of $L = O(T^2/n^2)$. In order to make the approach work for $L = O(T^2 \log n/n^2)$, we have to work harder by counting the number of 2-cycles in the hypergraph. The reason is that for hypergraphs with not too many 2-cycles, the existence of larger independent sets can be shown, and algorithms for finding them are also known.

3. Counting polygons of bounded area. Let $\gcd(h, s)$ denote the greatest common divisor of h and s. The following is a simple fact.

LEMMA 3. *For integers $s \geq 1$, let $F(s) := \sum_{h=1}^{s} \gcd(h, s)$. Then for every $\varepsilon > 0$ there is a constant $c_\varepsilon > 0$ such that $F(s) \leq c_\varepsilon \cdot s^{1+\varepsilon}$.*

Proof. For a given divisor d of s, there are at most s/d numbers $h \in \{1, \ldots, s\}$ such that $\gcd(h, s) = d$. Hence,

$$F(s) \leq \sum_{d \text{ divisor of } s} (s/d) \cdot d = s \cdot \#\text{divisors of s.}$$

The number of divisors of each $s \geq 4$ is at most $s^{c'/(\log\log s)}$ for some positive constant c' (see, for example, [11]). Thus, $F(s) \leq c_\varepsilon \cdot s^{1+\varepsilon}$ for some positive constant c_ε and all $s \geq 1$. □

Given a point p from the $T\times T$ grid, we denote by p_x and p_y its x- and y- coordinate. Define a lexicographic order on the points of the $T\times T$ grid by

$$p <_{lex} q \quad \Leftrightarrow \quad (p_x < q_x) \text{ or } (p_x = q_x \text{ and } p_y < q_y).$$

LEMMA 4. *Let $\varepsilon > 0$ be fixed. The number of degenerated triangles in the $T\times T$ grid is at most $O(T^{4+\varepsilon})$.*

Proof. Let $p_1 <_{lex} p_2 <_{lex} p_3$ be the points of a degenerated triangle. Let $s = (p_3)_x - (p_1)_x$ and $h = (p_3)_y - (p_1)_y$. The number of degenerated triangles with $s = 0$ or $h = 0$ is at most $O(T^4)$, since we can choose one of $O(T)$ lines and one of $O(T^3)$ point triples from that line.

By symmetry, neglecting constant factors, it is now enough to count degenerated triangles with $s \geq h > 0$. If we fix p_1, s, and h (and therefore also p_3), there are exactly $\gcd(h, s) - 1$ many grid points on the line segment between p_1 and p_3. Thus, the number of degenerated triangles is bounded from above by

$$c \cdot T^2 \cdot \sum_{s=1}^{T} \sum_{h=1}^{s} \gcd(h, s) = c \cdot T^2 \cdot \sum_{s=1}^{T} F(s),$$

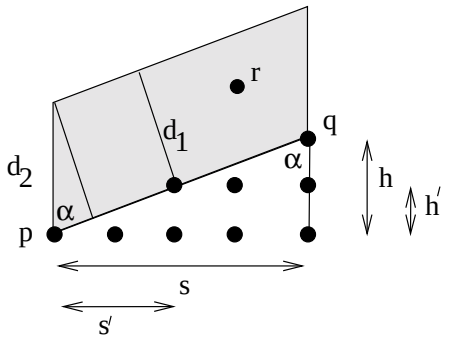
which, by Lemma 3, is at most $c'_\varepsilon \cdot T^2 \cdot \sum_{s=1}^{T} s^{1+\varepsilon} = O(T^{4+\varepsilon})$. □

LEMMA 5. *Let $p <_{lex} q$ be two points from the $T\times T$ grid. Define $s := q_x - p_x$. For every $L > 0$, the following hold.*

(a) *There are at most $4L$ grid points r such that the following three conditions hold simultaneously:*
   (1) $p <_{lex} r <_{lex} q$.
   (2) *The triangle (p, q, r) is nondegenerated.*
   (3) *The area of the triangle (p, q, r) is at most L.*

(b) *The number of grid points r which fulfill conditions* (2) *and* (3) *from* (a) *is at most $12LT/s$ if $s > 0$ and at most $4LT$ if $s = 0$.*

Proof. We first prove (a). By (1) and (2), we have $s > 0$. Assume without loss of generality (w.l.o.g.) that $h := q_y - p_y \geq 0$, since the other case can be dealt with analogously. The situation can be depicted as follows:



Since the area of the triangle (p, q, r) is at most L, and since $p <_{lex} r <_{lex} q$, we know that the point r must lie in some bounded strip above or below the straight line segment between p and q. (The strip is shaded in the above figure.)

Let the side lengths of the strip be d_2 and $\sqrt{s^2 + h^2}$. The width of the strip is given by $d_1 = 2L/\sqrt{s^2 + h^2}$. By a simple geometric argument, we have $d_1/d_2 = s/\sqrt{s^2 + h^2}$, hence $d_2 = 2L/s$.

If $h = 0$, then the number of points r is trivially bounded by $2 \cdot d_2 \cdot s = 4L$. Otherwise, we carefully count the number of grid points in the shaded strip. Assume w.l.o.g. that $p = (0, 0)$. Let $s' = s/\gcd(s, h)$ and $h' = h/\gcd(s, h)$. We count the number of grid points in the shaded strip above the line segment for the x-coordinates $0, \ldots, s'-1$.

For $x = 0$, the integer points from the y-interval $[0, d_2]$ are in the shaded strip. In general, for $x = i$, the integer points from the y-interval

$$\left[\frac{ih'}{s'}, \frac{ih'}{s'} + d_2\right] = \left[\frac{ih'}{s'}, \frac{ih'}{s'} + \frac{2L/\gcd(s, h)}{s'}\right]$$

are in the shaded strip. For every i, this interval contains exactly $D + 1$ numbers of the form m/s', where $D := \lfloor 2L/\gcd(s, h)\rfloor$.

Since h' and s' are relatively prime, the values ih', $i = 0, \ldots, s'-1$, run through all elements modulo s' exactly once. Hence, for every $t \in \{0, \ldots, s'-1\}$, the numbers of points of the form t^*/s' in the shaded strip such that $t^* \equiv t \bmod s'$ are equal.

In general, we have counted $(D+1)s'$ many points; hence there are $(D+1)s'/s' = (D + 1)$ many points of the form t/s' with $t \equiv 0 \bmod s'$ in the shaded strip. This is equal to the number of grid points in that strip. We are allowed to subtract one since the point p would lead to a degenerated triangle.

Since we are interested in the number of grid points in the shaded strip for $x = 0$ to $x = s - 1$, we thus count $D \cdot s/s'$ points. Finally, we have to multiply by two since the strip below the line segment between p and q also needs to be considered. There are some points at the x-coordinate of p which we have unnecessarily taken into account, but this is compensated for by the same number of points which we have not yet counted at the x-coordinate of q. Altogether, we have at most $2D \cdot s/s' = 2 \cdot \gcd(s,h) \cdot \lfloor 2L/\gcd(s,h) \rfloor \leq 4L$ possibilities for the choice of r.

For statement (b) and $s > 0$, we extend the shaded strip over all x-coordinates. Thus, we get a total number which is at most $4L \cdot (T/s + 2) \leq 12LT/s$. For $s = 0$, the bound $4LT$ is trivially obtained. □

LEMMA 6. *Let $\varepsilon > 0$ and $k \geq 3$ be fixed. For $L \geq T^{\frac{k-3}{k-2}+\varepsilon}$, there are at most $O(L^{k-2} \cdot T^4)$ configurations of k points $p_1 <_{lex} \cdots <_{lex} p_k$ in the $T \times T$ grid with a convex hull of area at most L.*

Proof. We divide the configurations into two classes. The first class contains the configurations where all triangles (p_1, p_i, p_k), $i = 2, \ldots, k-1$, are nondegenerated. After p_1 and p_k are chosen, we know by Lemma 5 that there are at most $4L$ candidates for each of $p_2, \ldots, p_{k-1}$ to choose from; hence this class contains at most $O(L^{k-2} \cdot T^4)$ configurations.

A configuration which has a degenerated triangle (p_1, p_i, p_k) can be constructed by first choosing some degenerated triangle (p_1, p_i, p_k) and then choosing $k-3$ more points q with $p_1 <_{lex} q <_{lex} p_k$. For each of those points, we have at most $4L+T$ many candidates. We obtain that there are at most $O(T^{4+\varepsilon} \cdot (L+T)^{k-3})$ many configurations in this class.

If $L \geq T$, the bound from the lemma follows easily. Otherwise, we have at most $O(T^{k+1+\varepsilon})$ many such configurations which meets the bound of the lemma since $L \geq T^{\frac{k-3}{k-2}+\varepsilon}$. □

4. An algorithm for $k = 3$. By Lemma 6, we know that for $L \geq T^{\varepsilon}$, the hypergraph $\mathcal{H}^{(3)}_{T,L}$ contains at most $O(L \cdot T^4)$ many hyperedges and thus its average degree satisfies $d_{avg} = O(L \cdot T^2)$. Just knowing this bound, one is able to provide an algorithm which computes an independent set of size at least $\Omega(T/\sqrt{L})$. We will use such an approach for the case $k \geq 4$ in section 5. For $k = 3$, we can exploit that the number of 2-cycles in $\mathcal{H}^{(3)}_{T,L}$ is not too large. For such hypergraphs, an algorithm which computes larger independent sets is known (see Theorem 8).

LEMMA 7. *Let $\varepsilon > 0$. The number of 2-cycles in the hypergraph $\mathcal{H}^{(3)}_{T,L}$ is at most*

$$O(L^2 \cdot T^4 \cdot \log T + T^{5+\varepsilon} + L \cdot T^5).$$

Proof. Consider a 2-cycle E_1, E_2 and let $\{v, w\} = E_1 \cap E_2$. This corresponds to two triangles which share two points v and w and which both have an area bounded by L. Let $s = w_x - v_x$ and $h = w_y - v_y$. By rotation symmetry (which we account for by an extra constant factor), we can assume that $s > 0$ and $0 \leq h \leq s$.

Let us first assume that the triangles corresponding to E_1 and E_2 both are non-degenerated. Using Lemma 5(b), the number of such pairs of triangles with area at most L can now be bounded from above by

$$C \cdot T^2 \cdot \sum_{s=1}^{T} \sum_{h=0}^{s} \left(\frac{LT}{s}\right)^2 \leq C' \cdot L^2 \cdot T^4 \cdot \sum_{s=1}^{T} \frac{1}{s} = O(L^2 \cdot T^4 \cdot \log T).$$

If both triangles are degenerated, we have four points on a line. By Lemma 4 there are at most $O(T^{5+\varepsilon})$ such configurations. If one of the triangles is nondegenerated and one is degenerated, then we can construct this situation by choosing one of the $O(L \cdot T^4)$ many nondegenerated triangles and choosing one out of at most $O(T)$ possible positions for the fourth point. This yields the bound of the lemma. ∎

The next result is taken from [3] and describes an algorithm for and an extension of a powerful result by Ajtai et al. [1]; compare also Fundia [4].

THEOREM 8 (see [3]). *Let $\mathcal{G} = (V, \mathcal{E})$ be a 3-uniform hypergraph on N vertices with average degree $d_{avg} = 3 \cdot |\mathcal{E}|/N$, where $d_{avg} \to \infty$. Let $t := \sqrt{d_{avg}}$. If the number $s_2(\mathcal{G})$ of 2-cycles in $\mathcal{G}$ satisfies $s_2(\mathcal{G}) \le N \cdot t^{3-\gamma}$ for some constant $\gamma > 0$, then one can find in polynomial time an independent set of size at least $\Omega((N/t) \cdot (\log t)^{1/2})$.*

We are now ready to prove our main theorem for the case $k = 3$.

THEOREM 9. *One can find in polynomial time n points in the unit square such that all triangles have area at least $\Omega(\log n/n^2)$.*

Proof. First, fix some $0 < \gamma < 1/4$ and let $\alpha := 1/(1-2\gamma)$; thus $1 < \alpha < 2$. Let

$$T := \lceil n^\alpha \rceil \quad \text{and} \quad L := \frac{\log n}{n^2} \cdot T^2.$$

From this choice, it follows that $L \ge T^\varepsilon$ for some fixed $\varepsilon > 0$.

Our earlier considerations have shown that the hypergraph $\mathcal{H}^{(3)}_{T,L}$ on $N{=}T^2$ vertices has average degree $d_{avg}{=}O(L \cdot T^2)$, i.e., $t := \sqrt{d_{avg}} = O(L^{1/2} \cdot T)$. However, this is only an upper bound for t. It is useful to distinguish two cases.

Case 1. $t \ge L^{1/2} \cdot T/\log n$.

We show that the conditions of Theorem 8 are fulfilled for the chosen γ. Since $\alpha < 2$, we have $L \cdot \log T/T = \log n \cdot \log T \cdot T/n^2 = o(1)$, and thus the dominating term in Lemma 7 for the number of 2-cycles in $\mathcal{H}^{(3)}_{T,L}$ is $O(L \cdot T^5)$. For the application of Theorem 8, it is enough to show now that $L \cdot T^5/B = o(1)$, where $B := T^2 \cdot (L^{1/2} \cdot T/\log n)^{3-\gamma}$:

$$\begin{aligned} \frac{L \cdot T^5}{T^2 \cdot (L^{1/2} \cdot T/\log n)^{3-\gamma}} &= \frac{T^\gamma}{L^{1/2-\gamma/2}} \cdot (\log n)^{3-\gamma} \\ &= \frac{1}{(\log n)^{1/2-\gamma/2}} \cdot \frac{n^{1-\gamma}}{T^{1-2\gamma}} \cdot (\log n)^{3-\gamma} \\ &= o(1), \end{aligned}$$

since $T^{1-2\gamma} \ge n$. Hence, we can apply Theorem 8. We now use the upper bound $t = O(L^{1/2} \cdot T)$ and obtain in polynomial time an independent set in $\mathcal{H}^{(3)}_{T,L}$ of size at least

$$\Omega\left(\frac{T^2}{t} \cdot \log^{1/2} t\right) = \Omega\left(\frac{T^2}{L^{1/2} \cdot T} \cdot \log^{1/2} n\right) = \Omega\left(\frac{T}{L^{1/2}} \cdot \log^{1/2} n\right) = \Omega(n).$$

Case 2. $t < L^{1/2} \cdot T/\log n$.

We apply the algorithm from Theorem 10 (see section 5) and obtain in polynomial time an independent set in $\mathcal{H}^{(3)}_{T,L}$ of size at least

$$\Omega\left(\frac{T^2}{L^{1/2} \cdot T/\log n}\right) = \Omega\left(\frac{T \cdot \log n}{L^{1/2}}\right) = \Omega\left(n \cdot \log n/(\log n)^{1/2}\right) = \Omega(n).$$

In both cases, the algorithm computes $c \cdot n$ points for some $c > 0$. If $c < 1$, we can adapt the constants and apply the above arguments to some $N = \Theta(n)$ replacing n such that the algorithm indeed finds at least n points where the condition on the triangles also holds. To summarize, we obtain in polynomial time at least n points in the $T \times T$ grid such that every triangle has area at least $c' \cdot T^2 \cdot \log n/n^2$ for some $c' > 0$. By rescaling, i.e., considering the $T \times T$ grid as located in the unit square, we have finished the proof. □

5. An algorithm for $k \geq 4$. For general k, one may consider the following problem: Find n points in the unit square such that the convex hull of every choice of k points among them has an area as large as possible.

Schmidt [17] investigated this problem for the case $k = 4$ and he proved the existence of a configuration on n points which achieves the bound given in Theorem 11 below.

Theorem 10 (Turán's theorem for hypergraphs) gives a lower bound for the independence number $\alpha(\mathcal{G})$ of a hypergraph $\mathcal{G}$ (cf. Spencer [18]), and a linear–time algorithm for achieving this bound is also available. We show that this algorithm can be exploited for Heilbronn's problem if $k \geq 4$.

THEOREM 10 (see [18]). *Let $\mathcal{G} = (V, \mathcal{E})$ be a k-uniform hypergraph, $k \geq 2$, with average degree d_{avg} and $|V| = N$. Let $t := \max\{1, (d_{avg})^{1/(k-1)}\}$. Then one can find in $\mathcal{G}$ in time $O(N + |\mathcal{E}|)$ an independent set of size at least $\Omega(N/t)$.*

We sketch the algorithm behind Theorem 10 for the sake of completeness.

Proof. Let $V = \{v_1, \ldots, v_N\}$. Assign a weight $p_i \in [0, 1]$ to every vertex v_i. Define a potential function F by $F(p_1, \ldots, p_N) = \sum_{i=1}^{N} p_i - \sum_{E \in \mathcal{E}} \prod_{v_i \in E} p_i$.

We have $F(p_1 := 1/t, \ldots, p_N := 1/t) = \frac{N}{t} - \frac{|\mathcal{E}|}{t^k} \geq \frac{k-1}{k} \cdot \frac{N}{t}$. Since F is linear in every p_i, we can set each p_i one after the other to either 0 or 1 without decreasing the potential. Set $V' = \{v_i \in V \mid p_i = 1\}$. If there is still some hyperedge E contained in V', then set $p_j = 0$ for some vertex $v_j \in E$ in this hyperedge. The potential does not decrease. Finally, we obtain an independent set of size at least $\frac{k-1}{k} \cdot \frac{N}{t}$. The running time is $O(N + |\mathcal{E}|)$. □

Theorem 10 and the counting result from Lemma 6 are enough to obtain our result for the generalization of Heilbronn's problem to convex hulls of k points. For the proof of the next theorem, we define a generalization of the hypergraph $\mathcal{H}^{(3)}_{T,L}$: let the vertices of the k-uniform hypergraph $\mathcal{H}^{(k)}_{T,L}$ be the grid points of the $T \times T$ grid and let a hyperedge be in $\mathcal{H}^{(k)}_{T,L}$ iff the area of the convex hull of the corresponding k points is at most L.

THEOREM 11. *For every fixed $k \geq 3$, there is a polynomial-time algorithm which on input n computes a configuration of n points in the unit square such that the convex hull of any k points has area at least $\Omega(1/n^{(k-1)/(k-2)})$.*

Proof. Choose some $0 < \alpha < \alpha' < 2\alpha$. Then for n large enough, there is some integer T with $n^{1+\alpha} \leq T \leq n^{1+\alpha'}$. Let $L := T^2/n^{(k-1)/(k-2)}$. It now holds that

$$L \geq n^{2\alpha + \frac{k-3}{k-2}} \qquad \text{and} \qquad T^{\frac{k-3}{k-2}+\varepsilon} \leq n^{(1+\alpha')(\frac{k-3}{k-2}+\varepsilon)} < n^{\frac{k-3}{k-2}+\alpha'+\varepsilon+\varepsilon\alpha'}.$$

Thus, we can choose some small $\varepsilon > 0$ such that $L \geq T^{\frac{k-3}{k-2}+\varepsilon}$; hence the conditions of Lemma 6 are fulfilled. By choice of L we infer that the average degree of the hypergraph $\mathcal{H}^{(k)}_{T,L}$ satisfies $d_{avg} = O(L^{k-2} \cdot T^2) = O((T^2/n)^{k-1})$.

Applying Theorem 10 with $t = O(T^2/n)$, we obtain in polynomial time an independent set of size at least $\Omega(n)$. For any k of these $\Omega(n)$ points from the $T \times T$ grid,

the area of their convex hull is at least $L = T^2/n^{(k-1)/(k-2)}$. Rescaling yields the desired result. □

6. Open problems. Of course, one would like to know the exact growth rate of $\Delta^{(k)}(n)$ for fixed $k \geq 3$. Hence, it would also be interesting to resolve whether for $k = 4$ or in general for $k \geq 4$ the lower bound on $\Delta^{(k)}(n)$ can also be improved by a logarithmic factor, e.g., by the factor $(\log n)^{1/(k-2)}$. It is also an open problem to determine whether $\Delta^{(k)}(n) = o(1/n)$ holds for fixed $k \geq 4$.

REFERENCES

[1] M. Ajtai, J. Komlós, J. Pintz, J. Spencer, and E. Szemerédi, *Extremal uncrowded hypergraphs*, J. Combin. Theory Ser. A, 32 (1982), pp. 321–335.

[2] G. Barequet, *A lower bound for Heilbronn's triangle problem in d dimensions*, in Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, PA, 1999, pp. 76–81.

[3] C. Bertram–Kretzberg and H. Lefmann, *The algorithmic aspects of uncrowded hypergraphs*, SIAM J. Comput., 29 (1999), pp. 201–230.

[4] A. Fundia, *Derandomizing Chebychev's inequality to find independent sets in uncrowded hypergraphs*, Random Structures Algorithms, 8 (1996), pp. 131–147.

[5] R. L. Graham, M. Grötschel, and L. Lovász, *Handbook of Combinatorics,* Vol. I, North–Holland, Amsterdam, 1995.

[6] J. Håstad, *Clique is hard to approximate within* $n^{1-\varepsilon}$, in Proceedings of the 37th IEEE Symposium on the Foundations of Computer Science (FOCS), 1996, pp. 627–636.

[7] T. Hofmeister and H. Lefmann, *Approximating maximum independent sets in uniform hypergraphs,* in Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science (MFCS), Lecture Notes in Comput. Sci. 1450, Springer, Berlin, 1998, pp. 562–570.

[8] J. Komlós, J. Pintz, and E. Szemerédi, *On Heilbronn's triangle problem,* J. London Math. Soc., 24 (1981), pp. 385–396.

[9] J. Komlós, J. Pintz, and E. Szemerédi, *A lower bound for Heilbronn's problem*, J. London Math. Soc., 25 (1982), pp. 13–24.

[10] H. Lefmann, *On Heilbronn's problem in higher dimension,* in Proceedings of the 11th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA), SIAM, Philadelphia, PA, 2000, pp. 60–64.

[11] I. Niven, H. S. Zuckerman, and H. L. Montgomery, *An Introduction to the Theory of Numbers,* 5th ed., John Wiley, New York, 1991.

[12] K. F. Roth, *On a problem of Heilbronn*, J. London Math. Soc., 26 (1951), pp. 198–204.

[13] K. F. Roth, *On a problem of Heilbronn,* II, Proc. London Math. Soc. (3), 25 (1972), pp. 193–212.

[14] K. F. Roth, *On a problem of Heilbronn,* III, Proc. London Math. Soc. (3), 25 (1972), pp. 543–549.

[15] K. F. Roth, *Estimation of the area of the smallest triangle obtained by selecting three out of n points in a disc of unit area,* in Proc. Symposia Pure Math. 24, AMS, Providence, RI, 1973, pp. 251–262.

[16] K. F. Roth, *Developments in Heilbronn's triangle problem,* Adv. Math., 22 (1976), pp. 364–385.

[17] W. M. Schmidt, *On a problem of Heilbronn,* J. London Math. Soc. (2), 4 (1972), pp. 545–550.

[18] J. Spencer, *Turán's theorem for k-graphs*, Discrete Math., 2 (1972), pp. 183–186.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 391–437

NONMALLEABLE CRYPTOGRAPHY*

DANNY DOLEV†, CYNTHIA DWORK‡, AND MONI NAOR§

Abstract. The notion of *nonmalleable* cryptography, an extension of semantically secure cryptography, is defined. Informally, in the context of encryption the additional requirement is that given the ciphertext it is impossible to generate a *different* ciphertext so that the respective plaintexts are related. The same concept makes sense in the contexts of string commitment and zero-knowledge proofs of possession of knowledge. Nonmalleable schemes for each of these three problems are presented. The schemes do not assume a trusted center; a user need not know anything about the number or identity of other system users.

Our cryptosystem is the first proven to be secure against a strong type of chosen ciphertext attack proposed by Rackoff and Simon, in which the attacker knows the ciphertext she wishes to break and can query the decryption oracle on any ciphertext other than the target.

Key words. cryptography, cryptanalysis, encryption, authentication, randomized algorithms, nonmalleability, chosen ciphertext security, auction protocols, commitment schemes, zero-knowledge

AMS subject classifications. 68M10, 68Q20, 68Q22, 68R05, 68R10

PII. S0097539795291562

1. Introduction.

The notion of *nonmalleable* cryptography is an extension of semantically secure cryptography. Informally, in the context of encryption the additional requirement is that given the ciphertext it is impossible to generate a *different* ciphertext so that the respective plaintexts are related. For example, consider the problem of *contract bidding*: municipality M has voted to construct a new elementary school, has chosen a design, and advertises in the appropriate trade journals, inviting construction companies to bid for the contract. The advertisement contains a public key E to be used for encrypting bids and a fax number to which encrypted bids should be sent. Company A places its bid of $\$1,500,000$ by faxing $E(15,000,000)$ to the published number over an insecure line. Intuitively, the public-key cryptosystem is *malleable* if, having access to $E(15,000,000)$, company B is more likely to generate a bid $E(\beta)$ such that $\beta \leq 15,000,000$ than company B would be able to do without the ciphertext. Note that company B *need not* be able to decrypt the bid of company A in order to consistently just underbid. In this paper we describe a nonmalleable public-key cryptosystem that prevents such underbidding. Our system does not even require company A to know of the existence of company B. It also does not require the municipality M to know of A or B before the companies bid, nor does it require A or B to have any kind of public key. The system remains nonmalleable even under a very strong type of chosen ciphertext attack, in which the attacker knows the ciphertext she wishes to break (or maul) and can query the decryption oracle on any ciphertext other than the target.

*Received by the editors September 11, 1995; accepted for publication (in revised form) May 27, 1999; published electronically June 3, 2000. A preliminary version of this work appeared in STOC'91.
http://www.siam.org/journals/sicomp/30-2/29156.html

†Dept. of Computer Science, Hebrew University, Jerusalem 91904, Israel (dolev@cs.huji.ac.il).

‡IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, CA 95120 (dwork@almaden.ibm.com). The research of this author was supported by BSF grant 32-00032-1.

§Dept. of Computer Science and Applied Mathematics, Weizmann Institute of Science, Rehovot 76100, Israel (naor@wisdom.weizmann.ac.il). Most of this work of this author was performed while at the IBM Almaden Research Center. The research of this author was supported by BSF grant 32-00032-1.

A well-established, albeit implicit, notion of nonmalleability is existential unforgeability of signature schemes [45]. Informally, a signature scheme is existentially unforgeable if, given access to $(m_1, S(m_1)), \ldots, (m_k, S(m_k))$, where $S(m_i)$ denotes a signature on message m_i, the adversary cannot construct a single valid $(m, S(m))$ pair for any *new* message m—even a nonsense message or a function of $m_1, \ldots, m_k$. Thus, existential unforgeability for signature schemes is the "moral equivalent" of nonmalleability for cryptography. We do not construct signature schemes in this paper. However, we introduce the related notion of *public-key authentication* and present a simple method of constructing a provably secure public-key authentication scheme based on any nonmalleable public-key cryptosystem.[1]

Nonmalleability is also important in private-key cryptography. Many common protocols, such as Kerberos or the Andrew Secure Handshake, use private-key encryption as a sort of authentication mechanism: parties A and B share a key K_{AB}. A sends to B the encryption of a nonce N under K_{AB}, and the protocol requires B to respond with the encryption under K_{AB} of $f(N)$, where f is some simple function such as $f(x) = x - 1$. The *unproved and unstated* assumption (see, e.g., [16]) is that seeing $K_{AB}(N)$ doesn't help an imposter falsely claiming to be B to compute $K_{AB}(f(N))$. As we shall see, this is *precisely* the guarantee provided by nonmalleability.

Nonmalleability is a desirable property in many cryptographic primitives other than encryption. For example, suppose researcher A has obtained a proof that $P \neq NP$ and wishes to communicate this fact to professor B. Suppose that, to protect herself, A proves her claim to B in a zero-knowledge fashion. Is zero-knowledge sufficient protection? Professor B may try to steal credit for this result by calling eminent professor E and acting as a *transparent prover*. Any questions posed by professor E to professor B are relayed by the latter to A, and A's answers to professor B are then relayed in turn to professor E. We solve this problem with a *nonmalleable zero-knowledge proof of knowledge.* Researcher A will get proper credit even without knowing of the existence of professor E, and even if professor E is (initially) unaware of researcher A.

Our work on nonmalleability was inspired by early attempts to solve the distributed coin flipping problem. Although $t+1$ rounds are necessary for solving Byzantine agreement in the presence of t faulty processors [33], in the presence of a global source of randomness the problem can be solved in constant expected time [62]. Thus, in the mid 1980s several attempts were made to construct a global coin by combining the individual sources of randomness available to each of the participants in the system. At a very high level, the original attempts involved commitment to coins by all processors, followed by a revelation of the committed values. The idea was that the global coin would be the exclusive-or (or some other function) of the individual committed values. Disregarding the question of how to force faulty processors to reveal their committed values, the original attempts erred because *secrecy was confused with independence*. In other words, the issue was malleability: even though the faulty processors could not know the committed values of the nonfaulty processors, they could potentially force a desired outcome by arranging to commit to a specific function of these (unknown) values.

As the examples show, *secrecy* does not imply *independence*. The goal of nonmalleable cryptography is to *force* this implication.

[1] For more on existentially unforgeable signature schemes, see [27, 45, 60].

1.1. Description of principal results.

Nonmalleable public-key cryptography. Goldwasser and Micali define a cryptosystem to be *semantically secure* if anything computable about the cleartext from the ciphertext is computable without the ciphertext [43]. This powerful type of security may be insufficient in the context of a distributed system, in which the mutual independence of messages sent by distinct parties often plays a critical role. For example, a semantically secure cryptosystem may not solve the contract bidding problem. Informally, a cryptosystem is *nonmalleable* if the ciphertext doesn't help: given the ciphertext it is no easier to generate a *different* ciphertext so that the respective plaintexts are related than it is to do so without access to the ciphertext. In other words, a system is nonmalleable if, for every relation R, given a ciphertext $E(\alpha)$, one cannot generate a different ciphertext $E(\beta)$ such that $R(\alpha, \beta)$ holds any more easily than can be done without access to $E(\alpha)$.[2] We present a public-key cryptosystem that is nonmalleable even against what we call a chosen ciphertext attack in the postprocessing mode (CCA-post) (defined informally in section 2.1 and formally in section 3). Since nonmalleability is an extension of semantic security, this yields the first public-key cryptosystem that is semantically secure against this strong type of chosen ciphertext attack.[3]

Our cryptosystem does not assume a trusted center, nor does it assume that any given collection of users knows the identities of other users in the system. In contrast, all other research touching on this problem of which we are aware requires at least one of these assumptions (e.g., [20, 21, 63]).

Nonmalleable string commitment. A second important scenario for nonmalleability is string commitment. Let A and B run a string commitment protocol. Assume that A is nonfaulty, and that A commits to the string α. Assume that, concurrently, C and D are also running a commitment protocol in which C commits to a string β. If B and C are both faulty, then even though neither of these players knows α, it is conceivable that β may depend on α. The goal of a nonmalleable string commitment scheme is to prevent this.

We present a nonmalleable string commitment scheme with the property that if the players have names (from a possibly unbounded universe), then for all polynomial-time computable relations R our scheme ensures that C is no more likely to be able to arrange that $R(\alpha, \beta)$ holds than it could do without access to the (A, B) interaction. Again, the scheme works even if A is unaware of the existence of C and D. If the players are anonymous, or the names they claim cannot be verified, then again if $\beta \neq \alpha$, then the two strings are no more likely to be related by R.

Intuitively, it is sufficient to require that C know the value to which it is committing in order to guarantee that α and β are unrelated. To see this, suppose C knows β and C also knows that $R(\alpha, \beta)$ holds. Then C knows "something about" α, thus violating the semantic security of the (A, B) string commitment. Proving possession of knowledge requires specifying a *knowledge extractor*, which, given the internal state of C, outputs β. In our case, the extractor has access to the (A, B) interaction, but it cannot rewind A. Otherwise it would only be a proof that *someone* (perhaps A) knows β, but not necessarily that C does.

[2] Clearly, there are certain kinds of relations R that we cannot rule out. For example, if $R(\alpha, \beta)$ holds precisely when $\beta \in E(\alpha)$, then from $E(\alpha)$ it is trivial to compute β, and hence $E(\beta)$, such that $R(\alpha, \beta)$ is satisfied. For formal definitions and specifications see section 2.

[3] For this type of attack it turns out that semantic and nonmalleable security are equivalent, which is not the case for weaker attacks. See section 3.4.2.

Nonmalleable zero-knowledge protocols. Using nonmalleable string commitment as a building block, we can convert any zero-knowledge interaction into a nonmalleable one. In particular we obtain nonmalleable zero-knowledge proofs of possession of knowledge, in the sense of Feige, Fiat, and Shamir [31]. Zero-knowledge protocols [44, 40] may compose in an unexpectedly malleable fashion. A classic example is the so-called "man-in-the-middle" attack (also known as the "intruder-in-the-middle," "Mafia scam," and "chess-masters problem") [24] on an identification scheme, similar in spirit to the transparent intermediary problem described above. Let A and D be nonfaulty parties, and let B and C be cooperating faulty parties (they could even be the same party). Consider two zero-knowledge interactive proof systems, in one A is proving to B knowledge of some string α and in the other C is proving to D knowledge of some string β. The two proof systems may be operating concurrently; since B and C are cooperating, the executions of the (A, B) and (C, D) proof systems may not be independent. Intuitively, nonmalleability says that if C can prove knowledge of β to D while A proves knowledge of α to B, then C could prove knowledge of β without access to the (A, B) interaction. The construction in section 5 yields a nonmalleable scheme for zero-knowledge proof of possession of knowledge.

1.2. Some technical remarks.

Nonmalleability in context. In the scenarios we have been describing, there are (at least) two protocol executions involved: the (A, B) interaction and the (C, D) interaction. Even if both pairs of players are, say, running string commitment protocols, the protocols need not be the same. Similar observations apply to the cases of nonmalleable public-key cryptosystems and nonmalleable zero-knowledge proofs of knowledge. Thus nonmalleability of a protocol really only makes sense with respect to another protocol. All our nonmalleable protocols are nonmalleable with respect to themselves. A more general result is mentioned briefly in section 5.

Identities. One delicate issue is the question of identities. Let α and β be as above. If the players have names, then our commitment and zero-knowledge interaction protocols guarantee that β is independent of α. The names may come from an unbounded universe. Note that there are many possibilities for names: timestamps, locations, message histories, and so on. If the players are anonymous, or the names they claim cannot be verified, then it is impossible to solve the *transparent prover* problem described earlier. However, the faulty prover must be completely transparent: if $\beta \neq \alpha$ then the two strings are unrelated by any relation R. In particular, recall the scenario described above in which (relatively unknown) researcher A seeks credit for the $P \neq NP$ result and at the same time needs protection against the transparent prover attack. Instead of proving knowledge of a witness s that $P \neq NP$, researcher A can prove knowledge of a statement $\alpha = A \circ s$. In this case the only dependent statement provable by professor B is α, which contains the name A. Note that we do not assume any type of authenticated channels.

Computational complexity assumptions. We assume the existence of trapdoor functions in constructing our public-key cryptosystems. The string commitment protocols and the compiler for zero-knowledge interactions require only one-way functions.

2. Definitions and system model. Since nonmalleability is a concept of interest in at least the three contexts of encryption, bit/string commitment, and zero-knowledge proofs, we give a single general definition that applies to all of these. Thus, when we speak of a *primitive* $\mathcal{P}$ we can instantiate any of these three primitives. We

start in section 2.1 by providing definitions for the primitives, as well as for some of the tools we use. Our presentation of the notion of security is nonstandard and we call it *semantic security with respect to relations*. In Theorem 2.4, we show that our version is equivalent to the "traditional" definition. We prefer this version for several reasons:

- It provides a uniform way of treating the security of all the primitives, i.e., the definition of zero-knowledge and semantic security do not *seem* different.
- It generalizes to the nonmalleable case in a natural way, whereas the usual notion of semantic security (provably) does not.

In section 2.2 we provide the definition of nonmalleable security. In section 2.3 we define the system model which is most relevant to those primitives which involve a lot of interaction.

The following definitions and notation are common to all the sections. We use $X \in_R B$ to mean that X is chosen from B at random. If B is a set, then X is simply chosen uniformly at random from the elements of B. If B is a distribution, then $X \in_R B$ means that X is chosen according to B from the support of B.

An *interactive* protocol $(A, B)[c, a, b]$ is an ordered pair of polynomial-time probabilistic algorithms A and B to be run on a pair of interactive Turing machines with common input c and with private inputs a and b, respectively, where any of a, b, c might be null.

We distinguish between the algorithm A and the agent $\psi(A)$ that executes it. We also use $\psi(A)$ to denote a faulty agent that is "supposed" to be running A (that is, that the nonfaulty participants expect to be running A) but has deviated from the protocol. Thus A is the protocol, and $\psi(A)$ is the player.

2.1. Definitions of primitives. In this section, we review the definitions from the literature of probabilistic public-key cryptosystems, string commitment, zero-knowledge interaction, and noninteractive zero-knowledge proof systems, all of which are used as primitives in our constructions. As mentioned above, we provide a unifying treatment of the security of all the primitives.

Probabilistic public-key encryption. A *probabilistic public-key encryption* scheme (see [43]) consists of the following:

- GP, the *key generator*, a probabilistic machine that on unary input 1^n, where n is the security parameter, outputs a pair of strings (e, d) (e is the *public key* and d is the *secret key*);
- E, the encryption function, gets three inputs: the public key e, $b \in \{0, 1\}$, and a random string r of length $p(n)$, for some polynomial p; $E_e(b, r)$ is computable in polynomial time;
- D, the decryption function, gets two inputs: c which is a ciphertext and the private key d which was produced by GP; $D_d(c)$ is computable in expected polynomial time;
- if GP outputs (e, d), then

$$\forall b \in \{0, 1\}\ \forall r \in \{0, 1\}^{p(n)} \quad D_d(E_e(b, r)) = b;$$

- The system has the property of *indistinguishability*: for all polynomial-time machines M, for all $c > 0$ $\exists n_c$ such that (s.t.) for $n > n_c$,

$$|\mathrm{Prob}[M(e, E_e(0, r)) = 1] - \mathrm{Prob}[M(e, E_e(1, r)) = 1]| < \frac{1}{n^c},$$

where the probability is taken over the coin-flips of GP, M and the choice of r.

This definition is for *bit* encryption and the existence of such a method suffices for our constructions. To encrypt longer messages one can concatenate several bit encryptions or use some other method. The definition of indistinguishability in this case becomes that with overwhelming probability over choice of encryption keys e, M cannot find two messages (m_0, m_1) for which it can distinguish with polynomial advantage between encryptions of m_0 and m_1. Formally, we give the following definition.

DEFINITION 2.1. *Let (GP, E, D) be a probabilistic public-key cryptosystem. We say that the system has the property of indistinguishability of encryptions if for all pairs of probabilistic polynomial time machines $(\mathcal{F}, \mathcal{T})$, for all $c > 0$ $\exists n_c$* s.t. *for $n > n_c$,*

$$\Pr\left[|\Pr[\mathcal{T}(e, m_0, m_1, E_e(m_0, r)) = 1] - \Pr[\mathcal{T}(e, m_0, m_1, E_e(m_1, r)) = 1]| \geq \frac{1}{n^c}\right] < \frac{1}{n^c},$$

where the external probability is over the choice of e and the coin-flips of $\mathcal{F}$ (which gets e as input), and each internal probability is taken the coin-flips of $\mathcal{T}$ and the choice of r.

For implementations of probabilistic encryption see [2, 14, 39, 52, 66]. In particular, such schemes can be constructed from *any* trapdoor permutation.

When describing the security of a cryptosystem, one must define what the attack is and what it means to break the system. The traditional notion of breaking (since [43]) has been a violation of semantic security or, equivalently, a violation of indistinguishability. This work introduces the notion of nonmalleable security, and a break will be a violation of nonmalleability. We return to this in section 2.2. We consider three types of attacks against a cryptosystem:

- Chosen plaintext. This is the weakest form of attack that makes any sense against a public-key cryptosystem. The attacker can (trivially) see a ciphertext of any plaintext message (because she can use the public encryption key to encrypt).
- Chosen ciphertext in the sense of [61], sometimes called lunch-break or lunchtime attacks in the literature; we prefer the term chosen ciphertext attack in the *preprocessing* mode, abbreviated CCA-pre. Here, the adversary may access a decryption oracle any polynomial (in the security parameter) number of times. Then the oracle is removed and a "challenge" ciphertext is given to the attacker.
- Chosen ciphertext in the sense of Rackoff and Simon [63]; we prefer the term chosen ciphertext attack in the *postprocessing* mode, abbreviated CCA-post. This is defined formally in section 3. The key point is that the attacker sees the challenge ciphertext *before* the oracle is removed, and can ask the oracle to decrypt any (possibly invalid) ciphertext *except the challenge*.

Our version of semantic security under chosen plaintext attack is the following: Let R be a relation computable in probabilistic polynomial time. We define two probabilities. Let $\mathcal{A}$ be an adversary that gets a key e and produces a distribution $\mathcal{M}$ on messages of length $\ell(n)$ by producing a description (including a specific time bound) of a polynomial-time machine that generates $\mathcal{M}$. $\mathcal{A}$ is then given a *challenge* consisting of a ciphertext $c \in_R E_e(m)$, where $m \in_R \mathcal{M}$ and $E_e(m)$ denotes the set $\{E_e(m, r)$ s.t. $|r| = p(n)\}$. In addition, $\mathcal{A}$ receives a "hint" (or history) about m in the

form of hist(m), where hist is a polynomially computable function. $\mathcal{A}$ then produces a string β. *We assume that the prefix of β is the description of $\mathcal{M}$.*

$\mathcal{A}$ is considered to have succeeded with respect to R if $R(m, \beta)$. Since β contains a description of $\mathcal{M}$, R is *aware* of $\mathcal{M}$ and may decide to accept or reject based on its description. This rules out achieving "success" by choosing a trivial distribution. Let $\pi(\mathcal{A}, R)$ be the probability that $\mathcal{A}$ succeeds with respect to R. The probability is over the choice of e, the coin-flips of $\mathcal{A}$, and the choice of m, so in particular it is also over the choice of $\mathcal{M}$.

For the second probability, we have an adversary simulator $\mathcal{A}'$ who will not have access to the encryption. On input e, $\mathcal{A}'$ chooses a distribution $\mathcal{M}'$. Choose an $m \in_R \mathcal{M}'$ and give hist(m) to $\mathcal{A}'$. $\mathcal{A}'$ produces β. As above, $\mathcal{A}'$ is considered to have succeeded with respect to R if $R(m, \beta)$. Let $\pi'(\mathcal{A}', R)$ be the probability that $\mathcal{A}'$ succeeds.

REMARK 2.2. 1. *In their seminal paper on probabilistic encryption, Goldwasser and Micali separate the power of the adversary into two parts: a message finder that, intuitively, tries to find a pair of messages on which the cryptosystem is weak, and the line tapper, that tries to guess which of the two chosen messages is encrypted by a given ciphertext* [43]. *Accordingly, we have let $\mathcal{A}$ choose the message space $\mathcal{M}$ on which it will be tested. By letting $\mathcal{A}'$ choose $\mathcal{M}'$ (rather than "inheriting" $\mathcal{M}$ from $\mathcal{A}$), we are letting the simulator completely simulate the behavior of the adversary, so in this sense our definition is natural. A second reason for this choice is discussed in section* 3.4.3.

2. *As noted above, the fact that the description of $\mathcal{M}$ or $\mathcal{M}'$ is given explicitly to R prevents $\mathcal{A}'$ from choosing a trivial distribution, e.g., a singleton, since R can "rule out" such $\mathcal{M}$'s.*

DEFINITION 2.3. *A scheme $\mathcal{S}$ for public-key cryptosystems is* semantically secure with respect to relations *under chosen plaintext attack if for every probabilistic polynomial time adversary $\mathcal{A}$ as above there exists a probabilistic polynomial time adversary simulator $\mathcal{A}'$ such that for every relation $R(m, \beta)$ and function hist(m), both computable in probabilistic polynomial time, $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial.* In this definition, the chosen plaintext attack is implicit in the definition of $\mathcal{A}$. This is a convention that will be followed throughout the paper.

Note the differences between our definition of semantic security with respect to relations and the original definition of semantic security [43]: in the original definition the challenge was to compute $f(x)$ given $E(x)$, where the function f is not necessarily even recursive. In contrast, here R is a relation and it is probabilistic polynomial time computable. Nevertheless, the two definitions are equivalent, as we prove in Theorem 2.4.[4]

We prove the following theorem for the case of chosen plaintext attacks; the proof carries over to both CCA-preprocessing and CCA-postprocessing.

THEOREM 2.4. *A public-key cryptosystem is semantically secure with respect to relations under chosen plaintext attack if and only if it has the indistinguishability property.*

[4]The literature shows for three versions of semantic security and three corresponding versions of indistinguishability that each version of semantic security is equivalent to the corresponding version of indistinguishability [54, 36]. We are using a fourth version of indistinguishability—a uniform version of the nonuniform one-pass version in [54]. Equivalence of this definition to a corresponding version of semantic security has not been proved in the literature, but we conjecture it holds.

Proof. We first show that if the cryptosystem has the indistinguishability property, then it is semantically secure with respect to relations. Consider the following three experiments. Choose an encryption key e using GP. Given the public key e, $\mathcal{A}$ produces a distribution $\mathcal{M}$. Sample $\alpha_1, \alpha_2 \in_R \mathcal{M}$.

In the first experiment, $\mathcal{A}$ is given hist(α_1) and $E_e(\alpha_1)$ and produces β_1. By definition, for any relation R,

$$\Pr[R(\alpha_1, \beta_1) \text{ holds}] = \pi(\mathcal{A}, R).$$

In the second experiment, $\mathcal{A}$ is given hist(α_1) and $E_e(\alpha_2)$ and produces β_2. Let

$$\chi = \Pr[R(\alpha_1, \beta_2) \text{ holds}].$$

Note that if R is probabilistic polynomial-time computable and $\Pr[R(\alpha_1, \beta_1) \text{ holds}]$ differs polynomially from $\Pr[R(\alpha_1, \beta_2) \text{ holds}]$, where the probabilities are taken over the coin-flips by R, and the random bits used in generating the encryptions (but not over the choice of e, $\mathcal{M}$, and α_1 and α_2), then we can create a distinguisher for encryptions of α_1 and α_2 under encryption key e, so in particular, given e, we have found a pair of messages whose encryptions are easy to distinguish. Thus, with overwhelming probability over choice of e, $\mathcal{M}$, and α_1, α_2, the individual probabilities (with fixed e) are close. It follows that the probabilities $\pi(\mathcal{A}, R)$ and χ (which are aggregated over choice of e) are also close.

For the third experiment, consider an $\mathcal{A}'$ that, on input e, simulates $\mathcal{A}$ on e to get a distribution $\mathcal{M}$. It gives $\mathcal{M}$ as the distribution on which it should be tested. $\mathcal{A}'$ is then given hist(α) for an $\alpha \in_R \mathcal{M}$. $\mathcal{A}'$ generates $\alpha' \in_R \mathcal{M}$ and gives to the simulated $\mathcal{A}$ the hint hist(α) and the encryption $E_e(\alpha')$. The simulated $\mathcal{A}$ responds with some β, which is then output by $\mathcal{A}'$. Note that $\pi'(\mathcal{A}', R) = \chi$. Thus, if the cryptosystem has the indistinguishability property, then $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial, so the cryptosystem is semantically secure with respect to relations.

We now argue that if a cryptosystem does not have the indistinguishability property, then it is not semantically secure with respect to relations. If a system does not have the indistinguishability property, then there exists a polynomial time machine M that given the public key can find two messages (m_0, m_1) for which it can distinguish encryptions of m_0 from encryptions of m_1. The specification of $\mathcal{A}$ is as follows: given a key e, $\mathcal{A}$ runs M to obtain (m_0, m_1). Let $\mathcal{M} = \{m_0, m_1\}$, where m_0 and m_1 each has probability $1/2$, be the message distribution on which $\mathcal{A}$ is to be tested. The function hist is the trivial hist$(x) = 1$ for all x. Given an encryption $\gamma \in_R E_e(m)$, where $m \in_R \mathcal{M}$, $\mathcal{A}$ uses M to guess the value of m and outputs β, the resulting guess plus the description of $\mathcal{M}$. The relation R that witnesses the fact that the cryptosystem is not semantically secure with respect to relations is equality plus a test of consistency with $\mathcal{M}$. Recall that the description of $\mathcal{M}$ is provided explicitly, and hence R can also check that $\mathcal{M}$ is of the right form.

Since M is by assumption a distinguisher, having access to the ciphertext γ gives $\mathcal{A}$ a polynomial advantage at succeeding with respect to R over any $\mathcal{A}'$ that does not have access to the ciphertext (which has probability $1/2$). □

Thus, a scheme is semantically secure with respect to relations if and only if it has the indistinguishability property. It follows from the results in [36, 43, 54] that the notions of (traditional) semantic security, indistinguishability, and semantically secure with respect to relations are all equivalent.

String commitment. The literature discusses two types of bit or string commitment: *computational* and *information* theoretic. These terms describe the type of

secrecy of the committed values offered by the scheme. In computational bit commitment, there is only one possible way of opening the commitment. Such a scheme is designed to be secure against a probabilistic polynomial-time receiver and an arbitrarily powerful sender. In information theoretic commitment, it is possible to open the commitment in two ways, but the assumed computational boundedness of the sender prevents him from finding the second way. Such a scheme is designed to be secure against an arbitrarily powerful receiver and a probabilistic polynomial-time prover. We restrict our attention to computational string commitment.

A *string commitment* protocol between sender A and receiver B consists of two stages:

- The *commit* stage. A has a string α to which she wishes to commit to B. She and B exchange messages. At the end of this stage, B has some information that represents α, but B should gain no information on the value of α from the messages exchanged during this stage.
- The *reveal* stage. At the end of this stage, B knows α. There should be only one string that A can reveal.

The two requirements of a string commitment protocol are *binding* and *secrecy*. Binding means that, following the commit stage, A can reveal at most one string. In our scenario we require the binding to be unconditional but probabilistic: with high probability over B's coin-flips, following the commit stage there is at most one string that B accepts (as the value committed) in the reveal stage.

The type of secrecy we require is semantic security. We specify what this means using the notions of security with respect to relations (however, as above, it is equivalent to the "traditional" way of defining semantic security). Let $\mathcal{A}$ be an adversary that produces a distribution $\mathcal{M}$ on strings of length $\ell(n)$ computable in probabilistic polynomial time. A string $\alpha \in_R \mathcal{M}$ is chosen and $\mathcal{A}$ receives $\text{hist}(\alpha)$, where hist is a probabilistic polynomial-time computable function. The commitment protocol is executed, where $\psi(A)$ follows the protocol and $\psi(B)$ is controlled by $\mathcal{A}$. The adversary $\mathcal{A}$ then produces a string β. We assume that the prefix of β is the description of $\mathcal{M}$.

$\mathcal{A}$ is considered to have succeeded with respect to R if $R(\alpha, \beta)$. Let $\pi(\mathcal{A}, R)$ be the probability that $\mathcal{A}$ succeeds with respect to R. The probability is over the coin-flips of $\mathcal{A}$, and the choice of α.

For the second probability, we have an adversary simulator $\mathcal{A}'$ who will not have access to the $(\psi(A), \psi(B))$ execution of the string commitment protocol. $\mathcal{A}'$ chooses a distribution $\mathcal{M}'$. An $\alpha \in_R \mathcal{M}'$ and $\text{hist}(\alpha)$ is given to $\mathcal{A}'$. $\mathcal{A}'$ produces β. As above, $\mathcal{A}'$ is considered to have succeeded with respect to R if $R(\alpha, \beta)$.

DEFINITION 2.5. *A commitment scheme is* semantically secure with respect to relations *if for every probabilistic polynomial-time adversary $\mathcal{A}$ as above there exists a probabilistic polynomial-time adversary simulator $\mathcal{A}'$ s.t. for every probabilistic polynomial-time computable relation $R(\alpha, \beta)$ and function hist(m) computable in probabilistic polynomial time, $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial.*

Zero-knowledge interaction. We next present a generalization of a (uniform) zero-knowledge interactive proof of language membership.

Let $(A, B)[a, b]$ be an interactive protocol, where (a, b) belongs to a set Π of legal input pairs to A and B. (In the special case of zero-knowledge proofs of language membership, the valid pairs (a, b) have the property that the prefixes of a and b are the common input $x \in L$.) Roughly speaking, we say that (A, B) is *zero-knowledge* with respect to B if for every polynomial-time bounded B', there exists a simulator that can produce conversations between (A, B') which are indistinguishable from the

actual (A, B') conversation. More accurately, and pursuing the terminology of this section, let $\mathcal{A}$ be an adversary that controls $\psi(B)$. $\mathcal{A}$ chooses a joint distribution $\mathcal{D}$, consistent with Π, on $[a, b]$, and then a pair $[a, b]$ is drawn according to $\mathcal{D}$; $\psi(A)$ gets a, $\psi(B)$ gets b, and the interaction proceeds by $\psi(A)$ following the protocol (while $\psi(B)$'s actions are controlled by $\mathcal{A}$). The result is a transcript T of the conversation between $\psi(A)$ and $\psi(B)$. $\mathcal{A}$ also produces a string σ which contains as a prefix the description of $\mathcal{D}$ (and may contain such information as the state of $\psi(B)$ at the end of the protocol).

Let R be a ternary relation. $\mathcal{A}$ is considered to have succeeded with respect to R if $R([a, b], T, \sigma)$. Let $\pi(\mathcal{A}, R)$ be the probability that $\mathcal{A}$ succeeds with respect to R. The probability is over the coin-flips of $\mathcal{A}$, the coin-flips of $\psi(A)$, and the choice of $[a, b]$.

On the other hand, we have $\mathcal{A}'$ that selects $\mathcal{D}'$ consistent with Π. A pair $[a, b]$ is then drawn according to $\mathcal{D}'$, and $\mathcal{A}'$ receives b. $\mathcal{A}'$ produces a transcript T' and a string σ'. $\mathcal{A}'$ is considered to have succeeded with respect to R if $R([a, b], T', \sigma')$. Let $\pi(\mathcal{A}', R)$ be the probability that $\mathcal{A}$ succeeds with respect to R. The probability is over the coin-flips of $\mathcal{A}'$ and the choice of $[a, b]$.

DEFINITION 2.6. *A protocol (A, B) is zero-knowledge with respect to B if for all probabilistic polynomial-time adversaries $\mathcal{A}$ as above there exists a probabilistic polynomial-time adversary simulator $\mathcal{A}'$ such that for every relation R computable in probabilistic polynomial time $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial.*

(If $(a, b) \notin \Pi$, then zero-knowledge is not ensured, but other requirements may hold, depending on the protocol.)

Two interesting examples of zero-knowledge interaction are proof of language membership [44, 40] and proofs of knowledge [31]. Both of these can be based on the existence of string commitment protocols.

Noninteractive zero-knowledge proof systems. An important tool in the construction of our public-key cryptosystem are noninteractive zero-knowledge proof systems (NIZKs). The following explanation is taken almost verbatim from [61]: A (single-theorem) noninteractive proof system for a language L allows one party $\mathcal{P}$ to prove membership in L to another party $\mathcal{V}$ for any $x \in L$. $\mathcal{P}$ and $\mathcal{V}$ initially share a string U of length polynomial in the security parameter n. To prove membership of a string x in $L_n = L \cap \{0, 1\}^n$, $\mathcal{P}$ sends a message p as a proof of membership. $\mathcal{V}$ decides whether to accept or to reject the proof. Noninteractive zero-knowledge proof systems were introduced in [12, 13]. A noninteractive zero-knowledge scheme for proving membership in any language in NP which may be based on *any* trapdoor permutation is described in [32]. Recently, Kilian and Petrank [49, 50] found more efficient implementations of such schemes. Their scheme is for the circuit satisfiability problem. Let k be a security parameter. Assuming a trapdoor permutation on k bits, the length of a proof of a satisfiable circuit of size L (and the size of the shared random string) is $O(Lk^2)$.

The shared string U is generated according to some distribution $\mathcal{U}(n)$ that can be generated by a probabilistic polynomial-time machine. (In all the examples we know of it is the uniform distribution on strings of length polynomial in n and k, where the polynomial depends on the particular protocol, although this is not required for our scheme.)

Let L be in NP. For any $x \in L$, let $WL(x) = \{z |\ z \textit{ is a witness for } x\}$ be the set of strings that witness the membership of x in L. For the proof system to be of any use, $\mathcal{P}$ must be able to operate in polynomial time if it is given a witness $z \in WL(x)$.

We call this the *tractability* assumption for $\mathcal{P}$. In general, z is not available to $\mathcal{V}$.

Let $\mathcal{P}(x, z, U)$ be the distribution of the proofs generated by P on input x, witness z, and shared string U. Suppose that $\mathcal{P}$ sends $\mathcal{V}$ a proof p when the shared random string is U. Then the pair (U, p) is called the conversation. Any $x \in L$ and $z \in WL(x)$ induces a probability distribution $\mathcal{CONV}(x, z)$ on conversations (U, p), where $U \in \mathcal{U}$ is a shared string and $p \in \mathcal{P}(x, z, U)$ is a proof.

For the system to be zero-knowledge, there must exist a simulator Sim which, on input x, generates a conversation (U, p). Let $Sim(x)$ be the distribution on the conversations that Sim generates on input x, let $Sim_U(x) = Sim_U$ be the distribution on the U part of the conversation, and let $Sim_P(x)$ be the distribution on the proof component. In the definitions of [13, 32] the simulator has two steps: it first outputs Sim_U without knowing x, and then, given x, it outputs $Sim_P(x)$. (This requirement, that the simulator not know the theorem when producing U, is not essential for our purposes; however, for convenience, our proof in section 3.3 does assume that the simulator is of this nature.)

Let

$$ACCEPT(U, x) = \{p | \mathcal{V} \text{ accepts on input } U, x, p\},$$

and let

$$REJECT(U, x) = \{p | \mathcal{V} \text{ rejects on input } U, x, p\}.$$

The following is the definition of noninteractive proof systems of [12], modified to incorporate the tractability of $\mathcal{P}$. The uniformity conditions of the system are adopted from Goldreich [35].

DEFINITION 2.7. *A triple $(\mathcal{P}, \mathcal{V}, \mathcal{U})$, where $\mathcal{P}$ is a probabilistic polynomial-time machine, $\mathcal{V}$ is a polynomial-time machine, and $\mathcal{U}$ is a polynomial-time sampleable probability distribution, is a* noninteractive zero-knowledge proof system *for the language $L \in NP$ if the following are present:*

1. Completeness *(if $x \in L$, then $\mathcal{P}$ generates a proof that $\mathcal{V}$ accepts): For all $x \in L_n$, for all $z \in WL(x)$, with overwhelming probability for $U \in_R \mathcal{U}(n)$ and $p \in_R \mathcal{P}(x, z, U)$, $p \in ACCEPT(U, x)$. The probability is over the choice of the shared string U and the internal coin-flips of $\mathcal{P}$.*
2. Soundness *(if $y \notin L$, then no prover can generate a proof that $\mathcal{V}$ accepts): For all $y \notin L_n$ with overwhelming probability over $U \in_R \mathcal{U}(n)$ for all $p \in \{0,1\}^*$, $p \in REJECT(U, y)$. The probability is over the choices of the shared string U.*
3. Zero-knowledge: *There is a probabilistic polynomial-time machine Sim which is a simulator for the system: For all probabilistic polynomial-time machines $\mathcal{C}$, if $\mathcal{C}$ generates $x \in L$ and $z \in WL(x)$, then*

$$|\text{Prob}[\mathcal{C}(w) = 1 | w \in_R Sim(x)] - \text{Prob}[\mathcal{C}(w) = 1 | w \in_R \mathcal{CONV}(x, z)]| < \frac{1}{p(n)}$$

for all polynomials p and sufficiently large n.

In the construction of the nonmalleable cryptosystem in section 3, noninteractive zero-knowledge proof systems are used to prove that encryptions generated under *independent* keys correspond to the *same* plaintext. This is similar to their application in [61].

2.2. Definitions specific to nonmalleability. In any interactive protocol (A, B) for primitive $\mathcal{P}$, party A has an *intended value*. In the case of encryption it is the value encrypted in $\psi(B)$'s public key; in string commitment it is the string to which $\psi(A)$ commits; in a zero-knowledge proof it is the theorem being proved interactively. The intended value is a generalization of the notion of an input. Indeed, when $\psi(A)$ is nonfaulty we may refer to the intended value as an *input* to A. However, we do not know how to define the input to a faulty processor that can, for example, refuse to commit to it. In this case we may need to substitute in a default value. The term *intended value* covers cases like this.

We sometimes refer to $\psi(A)$ as the *sender* and to $\psi(B)$ as the *receiver*. We use the verb *to send* to mean, as appropriate, to send an encrypted message, to commit to, and to prove knowledge of. Intuitively, in each of these cases information is being transmitted, or sent, from the sender to the receiver.

Interactive protocols (A, B), including the simple sending of an encrypted message, are executed in a context, and the participants have access to the history preceding the protocol execution. When $\psi(A)$ has intended value α, we assume both parties have access to $\text{hist}(\alpha)$, intuitively, information about the history that leads to $\psi(A)$ running the protocol with intended value α.

In some cases we also assume an underlying probability distribution $\mathcal{D}$ on intended values, to which both parties have access (that is, from which they can sample in polynomial time).

An *adversarially coordinated* system of interactive protocols

$$\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$$

consists of two interactive protocols (A, B) and (C, D), an adversary $\mathcal{A}$ controlling the agents $\psi(B)$ and $\psi(C)$, the communication between these agents, and the times at which all agents take steps.

Generally, we are interested in the situation in which $A = C$ and $B = D$, for example, when both interactive protocols are the same bit commitment protocol. Thus, for the remainder of the paper, unless otherwise specified, $(A, B) = (C, D)$, but $\psi(A), \psi(B), \psi(C)$, and $\psi(D)$ are all distinct.

Consider the adversarially coordinated system $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$. In an execution of this system, $\psi(A)$ sends an intended value $\alpha \in_R \mathcal{D}$ in its conversation with $\psi(B)$, and $\psi(C)$ sends an intended value β in its conversation with $\psi(D)$. If $\psi(C)$ fails to do so—e.g., fails to respond to a query, is caught cheating, or produces invalid ciphertexts—we take β to be all zeros.

We treat "copying" slightly differently in the context of encryption, which is noninteractive, and in the commitment and zero-knowledge settings, which are interactive. In particular, our results are stronger for encryption, since our construction rules out anything but exact copying of the ciphertext. Thus, seeing the ciphertext does not help the adversary to construct a different encryption of the same message. In the interactive setting we only ensure that if $\alpha \neq \beta$, then the two values are unrelated. We use identities (chosen by the users and not enforced provided by any authentication mechanism) to force α and β to be different. In particular, if the adversary wishes to be a transparent intermediary, then we do not bother to rule out the case in which the adversary commits to or proves exactly the same string as A does, even if it gives a different commitment (to the same value) or a different proof (of the same theorem).

We now formally define the nonmalleability guarantee in the interactive setting. A *relation approximator* R is a probabilistic polynomial-time Turing machine taking

two inputs[5] and producing as output either 0 or 1. The purpose of the relation approximator is to measure the correlation between α and β. That is, R measures how well the adversary manages to make β depend on α. In the interactive settings, we restrict our attention to the special class of relation approximators which on input pairs of the form (x, x) always output 0. The intuition here is that we cannot rule out copying, but, intuitively, this is not the case in which the adversary "succeeds."

When we discuss composition (or parallel execution) we will extend the definition so that the first input is actually a vector V of length k. The intuition here is that C may have access to several interactions with, and values sent by, nonfaulty players. In that case, the approximator must output zero on inputs (V, y) in which y is either a component of V, corresponding to the case in which $\psi(C)$ sends the same value as one of the nonfaulty players (in the case of encryption this is ruled out by the definition of the adversary).

Given a probability distribution on the pair of inputs, there is an a priori probability, taken over the choice of intended values and the coin-flips of R, that R will output 1. In order to measure the correlation between α and β, we must compare R's behavior on input pairs (α, β) generated as described above to its behavior on pairs (α, γ), where γ is sent without access to the sending of α (although as always we assume that $\psi(C)$ has access to $\mathcal{D}$ and hist(α)).

An *adversary simulator* for a commitment (zero-knowledge proof of knowledge) scheme $\mathcal{S}$ with input distribution $\mathcal{D}$ and polynomial-time computable function hist, is a probabilistic polynomial-time algorithm that, given hist, hist(α), and $\mathcal{D}$, produces an intended value γ.

Consider an adversarially coordinated system of interactive protocols $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$, where (A, B) and (C, D) are both instances of $\mathcal{S}$, and Π is the set of legal input pairs to the two parties executing $\mathcal{S}$. $\mathcal{A}$ may choose any probabilistic polynomial-time sampleable distribution $\mathcal{D}$ on the joint distribution to all four players,

$$\psi(A), \psi(B), \psi(C), \psi(D),$$

respectively, where the inputs to $\psi(A)$ and $\psi(B)$ are consistent with Π. Let $(\alpha, x, y, \delta) \in_R \mathcal{D}$. For any relation approximator R, let $\pi(\mathcal{A}, R)$ denote the probability, taken over all choices of $\psi(A)$, $\psi(D)$, $\mathcal{A}$, and R, that $\mathcal{A}$, given x, y, hist(α), and participation in the (A, B) execution in which $\psi(A)$ sends α, causes $\psi(C)$ to send β in the (C, D) execution, such that $R(\alpha, \beta)$ outputs 1, *under some specified form of attack.* (Since $\psi(C)$ is under control of the adversary, there is no reason that β should equal y.)

Similarly, for an adversary simulator $\mathcal{A}'$ choosing a joint distribution $\mathcal{D}'$ for all four players where the inputs to $\psi(A)$ and $\psi(B)$ are consistent with Π, for $(\alpha, x, y, \delta) \in_R \mathcal{D}'$, let $\mathcal{A}'$ have access to x, y, and hist(α), and let $\mathcal{A}'$ send γ. Let $\pi'(\mathcal{A}', R)$ denote the probability, taken over the choices of $\mathcal{A}'$, and the choices of R, that $R(\alpha, \gamma) = 1$.

DEFINITION 2.8. *A scheme $\mathcal{S}$ for a primitive $\mathcal{P}$ is nonmalleable with respect to itself under a given type of attack G, if for all adversarially coordinated systems $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$, where $(A, B) = (C, D) = \mathcal{S}$, where $\mathcal{A}$ mounts an attack of type G, there exists an adversary simulator $\mathcal{A}'$ such that for all relation approximators R, $|\, \pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R) \,|$ is subpolynomial.*[6]

[5] Sometimes we will need R to take three inputs, the third being in plaintext.

[6] In the previous version of this paper the order of quantifiers was $\forall R \forall \mathcal{A} \exists \mathcal{A}'$, yielding a possibly weaker definition. However, all the constructions in our work satisfy the stronger order of quantifiers given here. Now all our definitions share a common order of quantifiers.

This definition is applicable to *all* three primitives. As stated above, the precise attack against the system is crucial to the definition of $\mathcal{A}$ and hence of $\pi(\mathcal{A}, R)$. In particular, when we discuss encryption in section 3, we will specify the nature of the adversary precisely. The definition makes sense for all types of attack, with the appropriate choices of $\pi(\mathcal{A}, R)$. Finally, we must specify the "unit" which we are trying to protect; i.e., is it a single encryption or several?

REMARK 2.9. *There are three possible interpretations of Definition* 2.8*, according to the running time of* $\mathcal{A}'$*:*

1. $\mathcal{A}'$ *runs in a fixed polynomial time; this is* strict *nonmalleability (we usually drop the appellation "strict").*
2. $\mathcal{A}'$ *runs in* expected *polynomial time; in accordance with Goldreich's taxonomy for zero-knowledge, we call this* liberal [35] *nonmalleability.*
3. *For every* ε *there exists* $\mathcal{A}'$ *running in time polynomial in* n *and* ε^{-1} *such that* $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)| < \varepsilon$*; this is the* ε-malleability*, this time in analogy to* ε*-knowledge* ([43]; *see also* [29])*.*

Our public-key cryptosystem is strictly nonmalleable. M. Fischlin and R. Fischlin have pointed out that we do not prove strict nonmalleability in our commitment scheme; however, we prove both liberal nonmalleability and ε*-malleability.*

2.3. System model. We assume a completely asynchronous model of computing. For simplicity, we assume FIFO communication links between processors (if the links are not FIFO, then this can be simulated using sequence numbers). We *do not* assume authenticated channels.

We *do not* assume the usual model of a fixed number of mutually aware processors. Rather, we assume a more general model in which a given party does not know which other parties are currently using the system. For example, consider a number of interconnected computers. A user ("agent") can log into any machine and communicate with a user on an adjacent machine, without knowing whether a given third machine is actually in use at all, or if the second and third machines are currently in communication with each other. In addition, the user does not know the set of potential other users, nor need it know anything about the network topology.

Thus, we do not assume a given user knows the identities of the other users of the system. On the other hand, our protocols may make heavy use of user identities. One difficulty is that in general, one user may be able to impersonate another. There are several ways of avoiding this. For example, Rackoff and Simon [63] propose a model in which each sender possesses a secret associated with a *publicly known* identifying key issued by a trusted center.

In the scenario of interconnected computers described above, an identity could be composed of the computer serial number and a timestamp, possibly with the addition of the claimed name of the user. In the absence of some way of verifying claimed identities, *exact copying* of the pair, claimed identity and text, cannot be avoided, but we rule out essentially all other types of dependence between intended values.

We can therefore assume that the intended value α sent by $\psi(A)$ contains as its first component a user identity, which may or may not be verifiable. Fix a scheme $\mathcal{S}$ and an adversarially coordinated system of interactive protocols $\langle (A, B), (C, D), \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$, where (A, B) and (C, D) are both instances of $\mathcal{S}$, and let α and β be sent by $\psi(A)$ and $\psi(C)$, respectively. Then, whether or not the identities can be checked, if $\mathcal{S}$ is nonmalleable and $\alpha \neq \beta$, then β's dependence on α is limited to dependence on $\text{hist}(\alpha)$. In addition, if the identities can be checked, then $\alpha \neq \beta$.

In order to avoid assumptions about the lengths of intended values sent, we assume

the space of legal values is prefix-free.

3. Nonmalleable public-key cryptosystems. A *public-key cryptosystem* allows one participant, the owner, to publish a public key, keeping secret a corresponding private key. Any user that knows the public key can use it to send messages to the owner; no one but the owner should be able to read them. In this section, we show how to construct nonmalleable public-key cryptosystems. The definitions apply, mutatis mutandi, to private-key cryptosystems. As was done by [45] in 1984 in the context of digital signatures, when defining the security of a cryptosystem one must specify (a) the type of attack considered and (b) what it means to break the cryptosystem.

The cryptosystem we construct is secure against chosen ciphertext attacks. In fact it is secure against a more severe attack suggested by Rackoff and Simon [63] and which we call CCA-post: The attacker knows the ciphertext she wishes to crack while she is allowed to experiment with the decryption mechanism. She is allowed to feed it with any ciphertext she wishes, except for the exact one she is interested in. Thus the attacker is like a student who steals a test and can ask the professor any question, except the ones on the test. This is the first public-key cryptosystem to be provably secure against such attacks. Indeed, (plain) RSA [64] and the implementation of probabilistic encryption based on quadratic residuousity [43] are insecure against CCA-post.

Malleability, as defined in section 2.2, specifies what it means to "break" the cryptosystem. Informally, given a relation R and a ciphertext of a message α, the attacker $\mathcal{A}$ is considered successful if it creates a *ciphertext* of β such that $R(\alpha, \beta) = 1$. The cryptosystem is nonmalleable under a given attack G if for every $\mathcal{A}$ mounting an attack of type G, there is an $\mathcal{A}'$ that, without access to the ciphertext of α, succeeds with similar probability as $\mathcal{A}$ in creating a ciphertext of γ such that $R(\alpha, \gamma) = 1$. Given the notion of semantic security with respect to relations and Theorem 2.4, nonmalleability is clearly an extension of semantic security. See section 3.4.2 for the relationship between nonmalleability and the type of attack.

We now define precisely the power of the CCA-post adversary $\mathcal{A}$. Let R be a polynomial-time computable relation. Let n be the security parameter. $\mathcal{A}$ receives the public key $e \in_R GP(n)$ and can adaptively choose a sequence of ciphertexts $c_1, c_2, \ldots$. On each of them, $\mathcal{A}$ receives the corresponding plaintext. It then produces a distribution $\mathcal{M}$ on messages of length $\ell(n)$, for some polynomial ℓ, by giving the polynomial-time machine that can generate this distribution. $\mathcal{A}$ then receives as a challenge a ciphertext $c \in_R E_e(m)$, where $m \in_R \mathcal{M}$, together with some "side-information" about m in the form of hist(m), where hist is some polynomially computable function. $\mathcal{A}$ then engages in a second sequence of adaptively choosing ciphertexts $c'_1, c'_2, \ldots$. The only restriction is that $c \neq c'_1, c'_2, \ldots$. At the end of the process, $\mathcal{A}$ produces a polynomially bounded length vector of ciphertexts $(f_1, f_2, \ldots)$ not containing the challenge ciphertext c, with each $f_i \in E_e(\beta_i)$, and a *cleartext* string σ which we assume contains a description of $\mathcal{M}$.[7] Let $\beta = (\beta_1, \beta_2, \ldots)$. $\mathcal{A}$ is considered to have succeeded with respect to R if $R(m, \beta, \sigma)$. (We separate β from σ because the goal of the adversary is to produce encryptions of the elements in β.) Let $\pi(\mathcal{A}, R)$ be the probability that $\mathcal{A}$ succeeds where the probability is over the coin-flips of the key generator, $\mathcal{A}, \mathcal{M}$ and

[7] In the public key context, σ serves no purpose other than providing the description of $\mathcal{M}$ as an input to R, since in this situation from any plaintexts $p \in \mathcal{M}$ that are part of σ it is always possible to compute an encryption of σ, so we could always add an additional $f_i \in E_e(p)$ to our vector of ciphertexts. However, we introduce the possibility of including plaintexts p in σ so that the definition can apply to symmetric, or private-key, encryption.

the encryption of m.

Let $\mathcal{A}'$ be an adversary simulator that does not have access to the encryptions or to the decryptions but can pick the distribution $\mathcal{M}'$. On input e, $\mathcal{A}'$ produces $\mathcal{M}'$ and then $m \in_R \mathcal{M}'$ is chosen. $\mathcal{A}'$ receives hist(m) and without the benefit of the chosen ciphertext attack should produce a vector of ciphertexts $(f_1, f_2, \ldots)$, where each $f_i \in E_e(\beta_i)$, and a string σ containing $\mathcal{M}'$. Let $\beta = (\beta_1, \beta_2, \ldots)$. As above, $\mathcal{A}'$ is considered to have succeeded with respect to R if $R(m, \beta, \sigma)$. Let $\pi'(\mathcal{A}', R)$ be the probability that $\mathcal{A}'$ succeeds where the probability is over the coin-flips of the key generator, $\mathcal{A}'$ and $\mathcal{M}'$.

Note that $\mathcal{A}'$ has a lot less power than $\mathcal{A}$: not only does it not have access to the ciphertext encrypting α, but it cannot perform *any* type of chosen ciphertext attack, even in choosing the distribution $\mathcal{M}'$. Note also that as in the definition of semantically secure with respect to relations, the fact that $\mathcal{M}$ is given to R prevents $\mathcal{A}'$ from choosing trivial distributions.

DEFINITION 3.1. *A scheme $\mathcal{S}$ for public-key cryptosystems is nonmalleable with respect to chosen ciphertext attacks in the postprocessing mode (CCA-post attacks), if for all probabilistic polynomial-time adversaries $\mathcal{A}$ as above there exists a probabilistic polynomial-time adversary simulator $\mathcal{A}'$ s.t. for all relations $R(\alpha, \beta, \sigma)$ computable in probabilistic polynomial time, $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial.*

Note that the definition does not require R to be restricted (to a relation approximator) as described in section 2.2.

An illustration of the power of nonmalleability under CCA-post attacks is presented in section 3.5, where we discuss an extremely simple protocol for *public-key authentication*, a relaxation of digital signatures that permits an authenticator A to authenticate messages m, but in which the authentication needn't (and perhaps shouldn't!) be verifiable by a third party. The protocol requires a nonmalleable public-key cryptosystem and is simply incorrect if the cryptosystem is malleable.

Simple ideas that do not work. A number of simple candidates for nonmalleable cryptosystems come to mind. Let E be a cryptosystem semantically secure against a chosen ciphertext attack. Assume for concreteness that A wishes to send the message m and B wishes to send "1 + the value sent by A." That is, B, without knowing m, wishes to send $m + 1$.

One "solution" would be to append to $E(m)$ a noninteractive zero-knowledge proof of knowledge of the encrypted value m. The problem with this approach is that the proof of knowledge may itself be malleable: conceivably, given $E(m)$ and a proof of knowledge of m, it may be possible to generate $E(m+1)$ and a proof of knowledge of $m + 1$.

Another frequently suggested approach is to sign each message. Thus, to send a message m, party A sends $(E(m), S_A(E(m)))$, where S_A is a private signing algorithm for which a public verification key is known. There are two problems with this: first, it assumes that *senders* as well as *receivers* have public keys; second, it misses the point: if E is malleable, then B, seeing $(E(m), S_A(E(m)))$, simply ignores the second component, generates $E(m+1)$, say, based on $E(m)$, and sends $(E(m+1), S_B(E(m+1)))$.

Yet another suggestion is to put the signature *inside* the ciphertext: A sends $E(m \circ S_A(m))$. This still suffers from the assumption that A has a public verification key corresponding to S_A, and it again misses the point: B is not trying to produce $E(m+1, S_A(m+1))$, but only $E(m+1 \circ S_B(m+1))$. The unforgeability properties of S_A say absolutely nothing about B's ability to produce an encryption of $S_B(m + 1)$.

One more suggestion is to append an ID to each message and send, for example, $E(A \circ m)$. Again, we do not know how to show that, based only on the semantic security of E against chosen ciphertext attack, seeing $E(A \circ m)$ does not help B to produce $E(B \circ m)$ or $E(B \circ m + 1)$.

Overview of the scheme. The public key consists of 3 parts: (1) a collection of n pairs of keys $\{\langle e_i^0,\ e_i^1\rangle | 1 \le i \le n\}$; (2) a random string U for providing zero-knowledge proofs of consistency in a noninteractive proof system; (3) a universal one-way hash function. U is uniformly distributed because it is to the advantage of its creator (the verifier in the noninteractive zero-knowledge proof) that it should be so.

The process of encryption consists of four parts.

1. An "identity" is chosen for the message by creating a public signature verification key; the corresponding signing key is kept private. The signing key is only used to sign a single message, so a one-time signature scheme may be used here.
2. The message is encrypted under several encryption keys chosen from $\{\langle e_i^0, e_i^1\rangle | 1 \le i \le n\}$ as a function of the public signature verification key chosen in the first step. The selection is made by hashing the public signature verification key using the universal one-way hash function that is part of the public key for the cryptosystem.
3. A (noninteractive zero-knowledge) proof of consistency is provided, showing that the value encrypted under all the selected keys is the same one.
4. The encryptions and the proof are signed using the private signing key chosen in the first step.

When a message is decrypted, the signature verification key comprising the identity is used to verify that the signature is valid; the proof of consistency of encryptions is also checked. Only then is the (now well-defined) plaintext extracted.

The hash function is used only for efficiency; without it we would have to increase n, the number of encryption key pairs $\langle e_i^0,\ e_i^1\rangle$ in the public key for the cryptosystem. Thus, intuitively, the hash function plays a role analogous to the usual role of a hash function in an implementation of a signature scheme; however, we use it to hash the public verification key of the (freshly chosen) signature scheme, rather than the text of a message to be signed. As we will see, the critical point is that every identity chosen yields a distinct set of keys under which consistent encryptions must be created.

The idea of encrypting under several keys and proving consistency appeared in [61]. However, in [61] every plaintext bit is encrypted under *every* public key (there are only two), while here each identity for a message yields a distinct set of keys. Thus, the main changes here to the scheme in [61] are

1. the addition of an "identity" for each message to select a distinct set of keys;
2. using a (hash of the) freshly chosen public signature verification key as the identity;
3. signing the encryptions under the selected keys and the proof of consistency with the (secret) signing key that corresponds to the identity.

To develop some intuition for how the identities are used, consider a hypothetical situation in which all the keys in the pairs $\langle e_i^0,\ e_i^1\rangle$, $i = 1, \ldots, n$, are completely malleable, and suppose further that given an NIZK that one set of encryptions is consistent, it is easy to generate a proof of the true theorem that a set of related encryptions is also consistent.

If (as is the case in [61], where only two keys are used) we were *not* to use

signatures and we were *not* to select a new set of keys for each message (so that an encryption $E(m)$ would be a consistent set of encryptions under *all* the e_i^0 and e_i^1, $i = 1, \ldots, n$, and a proof of consistency), then given an encryption $E(m)$, creating an encryption, say of $2m$, would be easy: use the assumed malleability of all the $e_i^b(m)$ to create encryptions $e_i^b(2m)$, and use the assumed malleability of the NIZK to prove (the true theorem) that the resulting set of encryptions is consistent. We combat this hypothetical attack (which we cannot rule out!) using the identities, as we now describe.

Consider an attacker that has an encryption $\alpha \in_R E(m)$ under our scheme and that wishes to create from it an encryption $\beta \in E(2m)$. Suppose, as above, that the encryption functions e_i^b are completely malleable and that the NIZKs are malleable in the sense previously described. In our case, the attacker must create an identity for the message. Remember that an identity is the public verification key for a signature scheme. The attacker can choose to use the same identity (signature verification key) as in α or a different one. If the identity is preserved, this means that the attacker is using the public signature verification key appearing in α, for which he does not know the corresponding signature key. In this case, while the attacker can exploit the malleability of the e_i^b and the NIZK, in the last step of the encryption process he must forge a signature on the *new* encryptions and *new* proof of consistency—which he cannot do because he does not know the private signing key. On the other hand, if the attacker selects a new identity for β, different than the one used in α, then, since the identity selects the keys e_i^b under which the message is encrypted, for some i the attacker will have in α only $e_i^b(m)$ (and he will *not* have $e_i^{1-b}(m)$), but he will need to create $e_i^{1-b}(2m)$, so there will be no way to exploit the malleability of the encryption schemes e_i^0 and e_i^1. To summarize, nonmalleability comes from the fact that the choice of the subsets and the signature each authenticate the other.

As in [61], anyone can decide whether a ciphertext is legitimate, i.e., decrypts to some meaningful message, by verifying the NIZK proof of consistency and checking, using the signature verification key that comprises its identity, that the message is correctly signed. Thus, no information is ever gained during an attack when the decrypting mechanism rejects an invalid ciphertext.

Intuitively, given $E(\alpha)$, an attacker with access to a decryption mechanism can generate a legal ciphertext $E(\beta)$ and learn β, but nonmalleability implies that an adversary simulator can generate $E(\gamma)$ without access to $E(\alpha)$, where γ is distributed essentially as β is distributed. Thus β is unrelated to α (nonmalleability), and learning β yields no information about α (semantic security).

3.1. The tools. We require a probabilistic public-key cryptosystem that is semantically secure (see section 2.1). Recall that GP denotes the key generator, e and d denote the public and private keys, respectively, and E and D denote, respectively, the encryption and decryption algorithms.

For public keys $e_1, e_2, \ldots, e_n$ a *consistent encryption* is a string w that is equal to

$$E_{e_1}(b, r_1), E_{e_2}(b, r_2), \ldots, E_{e_n}(b, r_n)$$

for some $b \in \{0, 1\}$ and $r_1, r_2, \ldots, r_n \in \{0, 1\}^{p(n)}$, for some polynomial p. The language of consistent encryptions $L = \{e_1, e_2, \ldots, e_n, w | w$ is a consistent encryption$\}$ is in NP. For a given word $w = E_{e_1}(b, r_1)$, $E_{e_2}(b, r_2)$, $\ldots, E_{e_n}(b, r_n)$, the sequence r_1, $r_2, \ldots, r_n$ is a witness for its membership in L. In order to prove consistency we need a *noninteractive zero-knowledge proof system* for L, as defined in section 2.1. Recall that the system consists of a prover, a verifier, and a common random string

U known to both the prover and the verifier and that such a scheme can be based on any trapdoor permutation. Note that the length of U depends only on the security parameter and not on the number of messages to be encrypted over the lifetime of this public key.

The cryptosystem uses a *universal family of one-way hash functions* as defined in [60]. This is a family of functions H such that for any x and a randomly chosen $h \in_R H$ the problem of finding $y \neq x$ s.t. $h(y) = h(x)$ is intractable. The family we need should compress from any polynomial in n bits to n bits. In [65] such families are constructed from any one-way function.

Finally we need a one-time *signature scheme*, which consists of GS, the scheme generator that outputs F, the public key of the signature scheme, and P the private key. Using the private key P any message can be signed in such a way that anyone knowing F can verify the signature and no one who does not know the private key P can generate a valid signature on any message except the one signed. For exact definition and history see [5, 45, 60].

3.2. The nonmalleable public-key encryption scheme. We are now ready to present the scheme $\mathcal{S}$.

Key generation.

1. Run $GP(1^n)$, the probabilistic encryption key generator, $2n$ times. Denote the output by
$$(e_1^0, d_1^0), (e_1^1, d_1^1), (e_2^0, d_2^0), (e_2^1, d_2^1), \ldots, (e_n^0, d_n^0), (e_n^1, d_n^1).$$
2. Generate random reference string U.
3. Generate $h \in_R H$.

The public encryption key is
$$\langle h, e_1^0, e_1^1, e_2^0, e_2^1, \ldots, e_n^0, e_n^1, U \rangle,$$
and the corresponding private decryption key is $\langle d_1^0, d_1^1, d_2^0, d_2^1, \ldots, d_n^0, d_n^1 \rangle$.

Encryption. To encrypt a message $m = b_1 b_2 \ldots b_k$:

1. Run $GS(1^n)$, the signature key generator. Let F be the public signature key and P be the private signature key.
2. Compute $h(F)$. Denote the output by the n-bit string $v_1 v_2 \ldots v_n$.
3. For each $1 \leq i \leq k$:
   (a) For $1 \leq j \leq n$,
      i. generate random $r_{ij} \in_R \{0,1\}^{p(n)}$;
      ii. generate $c_{ij} = E_{e_j^{v_j}}(b_i, r_{ij})$, an encryption of b_i using $e_j^{v_j}$.
   (b) Run $\mathcal{P}$ on $c_i = e_1^{v_1}, e_2^{v_2}, \ldots, e_n^{v_n}, c_{i1}, c_{i2}, \ldots, c_{in}$, with witness $r_{i1}, r_{i2}, \ldots, r_{in}$ and string U to get a proof p_i that $c_i \in L$.
4. Create a signature s of the sequence (c_1, p_1), $(c_2, p_2), \ldots, (c_k, p_k)$ using the private signature key P.

The encrypted message is
$$\langle F, s, (c_1, p_1), (c_2, p_2), \ldots, (c_k, p_k) \rangle.$$

Decryption. To decrypt a ciphertext $\langle F, s, (c_1, p_1), (c_2, p_2), \ldots, (c_k, p_k)\rangle$:

1. Verify that s is a signature of $(c_1, p_1),(c_2, p_2),\ldots,(c_k, p_k)$ with public signature key F.
2. For all $1 \le i \le k$, verify that c_i is consistent by running the verifier $\mathcal{V}$ on c_i, p_i, U.
3. Compute $h(F)$. Denote the output by $v_1 v_2 \ldots v_n$.
4. If $\mathcal{V}$ accepts in all k cases, then for all $1 \le i \le k$, retrieve b_i by decrypting using any one of $\langle d_1^{v_1}, d_2^{v_2}, \ldots, d_n^{v_n}\rangle$. Otherwise the output is null.

Note that, by the proof of consistency, the decryptions according to the different keys in step 4 are identical with overwhelming probability.

From this description it is clear that the generator and the encryption and decryption mechanisms can be operated in polynomial time. Also if the decryption mechanism is given a legitimate ciphertext and the right key, it produces the message encrypted.

3.3. Nonmalleable security under CCA-post. We now prove the nonmalleability of the public-key encryption scheme $\mathcal{S}$ under a CCA-post. We define a related scheme $\mathcal{S}'$ whose (malleable) semantic security with respect to relations under chosen *plaintext* attack is straightforward. We then argue that the semantic security of $\mathcal{S}'$ under chosen plaintext attack implies the nonmalleability of $\mathcal{S}$ under CCA-post.

The cryptosystem $\mathcal{S}'$.

1. Run $GP(1^n)$, the probabilistic encryption key generator, n times. Denote the output by

$$(e_1, d_1), (e_2, d_2), \ldots, (e_n, d_n).$$

   The public key is the n-tuple $\langle e_1, \ldots, e_n\rangle$; the private key is the n-tuple $\langle d_1, \ldots, d_n\rangle$.
2. Encrypt a message $m = b_1 b_2 \ldots b_k$ as follows.
3. For $1 \le j \le n$:
   - For $1 \le i \le k$,
     (a) generate random $r_{ij} \in_R \{0,1\}^{p(n)}$, and
     (b) generate $c_{ij} = E_{e_j}(b_i, r_{ij})$, an encryption of b_i under public key e_j using random string r_{ij}.
   - Let $c_j = c_{1j}, c_{2j}, \ldots, c_{kj}$ (c_j is the jth encryption of m).
4. The encryption is the n-tuple $\langle c_1, c_2, \ldots, c_n\rangle$.
5. To decrypt an encryption $\langle \alpha_1, \ldots, \alpha_n\rangle$, compute $m_j = D_{d_j}(\alpha_j)$ for $1 \le j \le n$. If $m_1 = m_2 = \cdots = m_n$, then output m_1; otherwise output "invalid encryption."

LEMMA 3.2. *The public-key encryption scheme $\mathcal{S}'$ is semantically secure with respect to relations under chosen plaintext attack.* □

We will prove nonmalleability of $\mathcal{S}$ by reduction to the semantic security of $\mathcal{S}'$. To this end, we define an adversary $\mathcal{B}$ that, on being given an encryption under $\mathcal{S}'$, generates an encryption under $\mathcal{S}$. As above, we abuse notation slightly: given a public key E in $\mathcal{S}$ (respectively, E' in $\mathcal{S}'$), we let $E(m)$ (respectively, $E'(m)$) denote the set of encryptions of m obtained using the encryption algorithm for $\mathcal{S}$ (respectively, for $\mathcal{S}'$) with public key E (respectively, E').

Notation. In what follows, adversaries $\mathcal{A}$ and $\mathcal{A}'$ are adversaries against the scheme $\mathcal{S}$. Adversaries $\mathcal{B}$ and $\mathcal{B}'$ are adversaries against the system $\mathcal{S}'$.

Procedure for $\mathcal{B}$: Given a public key $E' = \langle e_1, \ldots, e_n \rangle$ in $\mathcal{S}'$:

Preprocessing phase.

1. Generate n new (e, d) pairs.
2. Run the simulator for the noninteractive zero-knowledge proof of consistency to generate a random string U (the simulator should be able to produce a proof of consistency of n encryptions that will be given to it later on).
3. Choose a random hash function $h \in_R H$.
4. Run $GS(1^n)$ to obtain a signature scheme (F, P), where F is the public verification key.
5. Compute $h(F)$. Arrange the original n keys and the n new keys so that the keys "chosen" by $h(F)$ are the original n. Let E denote the resulting public key (instance of $\mathcal{S}$).

Simulation phase.

1. Run $\mathcal{A}$ on input E. $\mathcal{A}$ adaptively produces a polynomial length sequence of encryptions $x_1, x_2, \ldots$. For each x_i produced by $\mathcal{A}$, $\mathcal{B}$ verifies the signatures and the proofs of consistency. If these verifications succeed, $\mathcal{B}$ decrypts x_i by using one of the new decryption keys generated in preprocessing step 1 and returns the plaintext to $\mathcal{A}$.
2. $\mathcal{A}$ produces a description of $\mathcal{M}$, the distribution of messages it would like to attack. $\mathcal{B}$ outputs $\mathcal{M}$. We will show that the semantic security of $\mathcal{S}'$ with respect to relations under chosen plaintext attack implies the nonmalleability of $\mathcal{S}$ under CCA-post.
3. $\mathcal{B}$ is given $c' \in_R E'(m)$ and hist(m) for $m \in_R \mathcal{M}$. It produces a ciphertext $c \in E(m)$ using the simulator of preprocessing step 2 to obtain a (simulated) proof of consistency and the private key P generated at preprocessing step 5 to obtain the signature.
4. Give $\mathcal{A}$ the ciphertext c and hist(m). As in simulation step 1, $\mathcal{A}$ adaptively produces a sequence of encryptions $x'_1, x'_2, \ldots$ and $\mathcal{B}$ verifies their validity, decrypts, and returns the plaintexts to $\mathcal{A}$.

Extraction phase.

$\mathcal{A}$ produces the vector of encryptions $(E(\beta_1), E(\beta_2), \ldots)$ and a string σ. $\mathcal{B}$ produces $\beta = (\beta_1, \beta_2, \ldots)$ by decrypting each $E(\beta_i)$ as in the simulation phase and outputs β and σ. This concludes the description of $\mathcal{B}$.

LEMMA 3.3. *Let $\mathcal{A}$ be an adversary attacking the original scheme $\mathcal{S}$. On input E' and $c' \in_R E'(m)$, let E be generated by $\mathcal{B}$ as above, and let c be the encryption of m under E created by $\mathcal{B}$ in simulation step* 3. *Let $\zeta \neq c$ be any ciphertext under E, generated by $\mathcal{A}$. If the signatures in ζ are valid (can be verified with the public signature verification key in ζ), then $\mathcal{B}$ can decrypt ζ.*

Proof. Let F' be the public signature verification key in ζ. If $F' \neq F$, then by the security of the universal one-way hash functions, $h(F') \neq h(F)$ (otherwise using $\mathcal{A}$ one could break H). Thus, at least one of the encryption keys generated in preprocessing step 1 of the procedure for $\mathcal{B}$ will be used in ζ. Since $\mathcal{B}$ generated this encryption key and its corresponding decryption key, $\mathcal{B}$ can decrypt.

We now argue that $F' \neq F$ (that is, that we must be in the previous case). Since $\mathcal{A}$ has not seen F or anything depending on F during its chosen ciphertext attack in step 1 of the simulation phase, the probability that $\mathcal{A}$ uses $F' = F$ in a ciphertext during this step is negligible. Suppose for the sake of contradiction that after it has seen F in the target ciphertext c, $\mathcal{A}$ uses $F' = F$ in ζ. Then by the security of the signature scheme, only the original ciphertexts and proofs of consistency $(c_1, p_1), \ldots, (c_n, p_n)$

from preprocessing step 2 and simulation step 3 can be signed; otherwise $\mathcal{A}$ could be used to break the signature scheme. This forces $\zeta = c$, contradicting the fact that $\zeta \neq c$. □

Note that in step 3 of the simulation phase, the vector c' is a legitimate encryption under E' and therefore is a vector of consistent encryptions, so the simulated noninteractive proof of consistency is a proof of a true theorem. Note also that this is the only place in which a proof is simulated by $\mathcal{B}$. Thus, even though the shared random string is used to generate many proofs of consistency during the lifetime of the public key, the zero-knowledge property we will need for the proof is only for a single theorem, since the only simulated proof will be on the target ciphertext.

The next lemma says that, as in the Naor–Yung scheme, $\mathcal{A}$ cannot distinguish the "instance" of $\mathcal{S}$ concocted by $\mathcal{B}$ from a real instance of $\mathcal{S}$, so $\mathcal{A}$ is just as likely to break the concocted instance.

For any probabilistic polynomial-time relation R, let $\pi(\mathcal{B}, R)$ denote the probability that $\mathcal{B}$, using $\mathcal{A}$ as described in the simulation phase, generates a vector of plaintexts $(\beta_1, \beta_2, \ldots)$ and a string σ such that $R(m, \beta, \sigma)$ holds, where $\beta = (\beta_1, \beta_2, \ldots)$. By choice of $\mathcal{B}$, $\pi(\mathcal{B}, R)$ is exactly the probability that $\mathcal{A}$ breaks $\mathcal{S}$ with respect to R in the simulation phase: $\mathcal{A}$ (interacting with $\mathcal{B}$) generates σ and a vector of encryptions $(E(\beta_1), E(\beta_2), \ldots)$; by Lemma 3.3, $\mathcal{B}$ can decrypt these values and so outputs β_i, $i = 1, 2, \ldots$, together with σ.

LEMMA 3.4. *For any probabilistic polynomial-time relation R, let $\pi(\mathcal{B}, R)$ denote the probability that $\mathcal{B}$, using $\mathcal{A}$ as described in the simulation phase, generates a vector of plaintexts $(\beta_1, \beta_2, \ldots)$ and a string σ such that $R(m, \beta, \sigma)$ holds, where $\beta = (\beta_1, \beta_2, \ldots)$. Let $\pi(\mathcal{A}, R)$ denote the probability that $\mathcal{A}$ breaks a random instance of $\mathcal{S}$ with respect to R. Then $\pi(\mathcal{B}, R)$ and $\pi(\mathcal{A}, R)$ are subpolynomially close.*

Proof. As noted above, $\pi(\mathcal{B}, R)$ is exactly the probability with which $\mathcal{A}$ breaks $\mathcal{S}$ with respect to R in the simulation phase. The only difference between the instance of $\mathcal{S}$ generated by $\mathcal{B}$ and an instance of $\mathcal{S}$ generated at random is in the reference string U and the proof of consistency for the target ciphertext: in the former case these are produced by the simulator (steps 2 and 3 of the simulation phase) and in the latter case they are authentic. The lemma therefore follows immediately from the definition of noninteractive zero-knowledge (Definition 2.7): any difference between the two probabilities can be translated into an ability to distinguish a simulated proof from a true proof. □

THEOREM 3.5. *The public-key encryption scheme $\mathcal{S}$ is nonmalleable against CCA-post.*

Proof. Let $\mathcal{A}$ be any polynomially bounded adversary, and assume for the sake of contradiction that $\mathcal{A}$ and the probabilistic polynomial-time computable relation R witness the malleability of $\mathcal{S}$ under a CCA-post. We will use the semantic security of $\mathcal{S}'$ with respect to relations to derive a contradiction by exhibiting an adversary simulator $\mathcal{A}'$ that, without access to the target ciphertext and without mounting any kind of chosen ciphertext attack against $\mathcal{S}$, does (negligibly close to) as well as $\mathcal{A}$ at breaking $\mathcal{S}$ (in the malleability sense).

Let E' be an encryption key in $\mathcal{S}'$. Let $\mathcal{B}$ be as described above. $\mathcal{B}$ generates an encryption key E in $\mathcal{S}$, invokes $\mathcal{A}$ on E to obtain a message distribution $\mathcal{M}$, and outputs $\mathcal{M}$. $\mathcal{B}$ is then given a ciphertext $c' = E'(m)$ for $m \in_R \mathcal{M}$, generates a ciphertext $c = E(m)$, and presents E and c to $\mathcal{A}$. If $\mathcal{A}$ produces valid encryptions $E(\beta_i)$ s.t. $E(\beta_i) \neq E(m)$, then by Lemma 3.3, $\mathcal{B}$ can extract the β_i. Let $\beta = (\beta_1, \beta_2, \ldots)$. Let R be any probabilistic polynomial-time computable relation. By Lemma 3.4, the

probability that $R(m, \beta, \sigma)$ holds is subpolynomially close to $\pi(\mathcal{A}, R)$.

Recall the definition of semantically secure with respect to relations: There exists a procedure $\mathcal{B}'$ that "does as well" as $\mathcal{B}$ at producing messages related to m, in the following sense. On input $(E', 1^n)$, $\mathcal{B}'$ outputs a message distribution $\mathcal{M}'$; $m' \in_R \mathcal{M}'$ is selected, but $\mathcal{B}'$ is not given access to the ciphertext for m', just the hint $\text{hist}(m')$. $\mathcal{B}'$ generates β', σ'. The definition of semantic security with respect to relations guarantees that for every $\mathcal{B}$ there exists $\mathcal{B}'$ such that $|\Pr[R(m, \beta, \sigma)] - \Pr[R(m', \beta', \sigma')]| \leq \nu(n)$, where the probabilities are over the choice of public key, the coin-flips of $\mathcal{B}$ and $\mathcal{B}'$, respectively, the choices of m and m', and the coin-flips in creating the encryption of m. Note that we are *rerandomizing* the public key: it is chosen afresh for each probability.

We use $\mathcal{B}'$ to define the adversary simulator $\mathcal{A}'$ whose existence is mandated by the definition of nonmalleability under chosen ciphertext postprocessing. On input $(E, 1^n)$, $\mathcal{A}'$ ignores E and selects a public key for an instance E' of $\mathcal{S}'$ (with security parameter n). It then runs $\mathcal{B}'$ on $(E', 1^n)$ to select a message space $\mathcal{M}'$. $\mathcal{A}'$ outputs $\mathcal{M}'$. A message $m' \in_R \mathcal{M}'$ is chosen, and $\mathcal{A}'$ is given $\text{hist}(m')$, which it forwards to $\mathcal{B}'$. $\mathcal{B}'$ outputs (β', σ'), where $\beta' = (\beta'_1, \beta'_2, \ldots)$ is a vector of *plaintexts*. $\mathcal{A}'$ outputs the vector of *encryptions* $E(\beta) = (E(\beta'_1), E(\beta'_2), \ldots)$, together with σ'.

Clearly $\pi'(\mathcal{A}', R) = \pi'(\mathcal{B}', R)$, where the first term is the probability that $\mathcal{A}'$ succeeds at producing $E(\beta', \sigma')$ such that $R(m', \beta', \sigma')$ and the second term is the probability that $\mathcal{B}'$ succeeds at the same task.

By choice of $\mathcal{B}'$ $|\pi'(\mathcal{B}', R) - \pi(\mathcal{B}, R)|$ is negligible. This, together with Lemma 3.4 and the fact that $\pi'(\mathcal{A}', R) = \pi'(\mathcal{B}', R)$, implies that $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)| \leq \nu(n)$. Therefore $(\mathcal{A}, R)$ cannot witness the malleability of $\mathcal{S}$. □

COROLLARY 3.6. *If there exists a public-key cryptosystem semantically secure against chosen plaintext attack and if noninteractive zero-knowledge satisfying the requirements of Definition* 2.7 *is possible, then there exists a nonmalleable public-key cryptosystem secure against CCA-post. In particular, if trapdoor permutations exist, then such cryptosystems exist.*

An interesting open problem is whether one can rely on the existence of a public-key cryptosystem semantically secure against chosen plaintext attacks alone to argue that nonmalleable public-key cryptosystems secure against CCA-posts exist. Two assumptions that are known to be sufficient for semantically secure public-key cryptosystems secure against plaintext attacks but where the existence of the stronger kind of cryptosystems is not clear are the hardness of the Diffie–Hellman (search) problem and the unique shortest vector problem (used in the Ajtai–Dwork cryptosystem [1]).

3.4. Remarks.

3.4.1. On vectors of encryptions. 1. We have defined nonmalleable public-key encryptions to cover the case in which $\mathcal{A}$ produces a vector of encryptions $(E(\beta_1), \ldots, E(\beta_n))$, having been given access to only a *single* $E(\alpha)$. It is natural to ask, what happens if $\mathcal{A}$ is given access to encryptions of *multiple* α's, $(E(\alpha_1), \ldots, E(\alpha_n))$. Security under this type of composition is, intuitively, a sine qua non of encryption. A simple "hybrid" argument shows that *any* nonmalleable public key cryptosystem is secure in this sense: seeing the encryptions of multiple α's does not help the adversary to generate an encryption of even one related β.

2. The computational difficulty of generating a single $E(\beta)$ for a related β does not imply the computational difficulty of generating a vector $(E(\beta_1), \ldots, E(\beta_n))$ such that $R(\alpha, \beta_1, \ldots, \beta_n)$ holds. We next describe a counterexample in the case of a CCA-pre. Let E' be a nonmalleable cryptosystem under CCA-pre. Let $E(m)$ be constructed

as $(E'_0(m_0), E'_1(m_1))$, where $m = m_0 \oplus m_1$. Given a ciphertext of this form, the adversary can construct two ciphertexts: $(E'_0(m_0), E'_1(0))$ and $(E'_0(0), E'_1(m_1))$. The parity of the two decrypted values is $(m_0 \oplus 0) \oplus (0 \oplus m_1) = m_0 \oplus m_1 = m$. On the other hand, it can be shown from the nonmalleability of the E'_i that seeing $E(m)$ is of no assistance in generating *a single* encryption $E(m')$ s.t. $R(m, m')$.

3.4.2. Security taxonomy and comparisons. We have discussed two notions of breaking a cryptosystem, semantic security and nonmalleability, and three types of attacks:

- Chosen plaintext,
- chosen ciphertext attack in the preprocessing mode, and
- chosen ciphertext attack in the postprocessing mode.

This yields six types of security, and the question is whether they are all distinct and which implications exist. Two immediate implications are (i) nonmalleable security implies semantic security under the same type of attack, and (ii) security against CCA-post implies security against CCA-pre which in turn implies security against chosen plaintext attacks, using the same notion of breaking the cryptosystem. We now explore other possibilities—the discussion is summarized in Figure 3.1.

The first observation is that if a cryptosystem is *semantically secure* against CCA-post, then it is also *nonmalleable* against CCA-post, since the power of the adversary allows it to decrypt whatever ciphertext it generated. On the other hand, it is not difficult to start with a cryptosystem that is secure against CCA-pre and make it only secure against a chosen plaintext attack (under any notion of breaking), as we now explain. For the case of semantic security, simply add to the decryption mechanism the instruction that on input all 0's output the private key.

The case of nonmalleable security is more subtle. Choose a fixed random ciphertext c_0, and instruct the decryption mechanism to output the decryption key when presented with input c_0. In addition, instruct the decryption mechanism to output c_0 on input all 0's.

There is a simple method for "removing" nonmalleability without hurting semantic security: starting with a cryptosystem that is nonmalleable against CCA-pre, one can construct a cryptosystem that is only semantically secure against CCA-pre—add to each ciphertext a cleartext bit whose value is Xor-ed with the first bit of the plaintext. Thus, given a ciphertext of a message m it is easy to create a ciphertext of a message where the last bit is flipped, so the scheme is malleable. However, the semantic security remains, as long as the adversary does not have access to the challenge ciphertext while it can access the decryption box.

We do not know whether a scheme that is nonmalleable against CCA-pre is also nonmalleable against CCA-post. We conjecture that whenever deciding whether or not a string represents a legitimate ciphertext (that could have been generated by any user) is easy (to someone *not* holding the private key), nonmalleability implies semantic security against CCA-post. From the above discussion (summarized in Figure 3.1), we conclude that of the six possibilities for security of a cryptosystem (combinations of the type of attack and notion of breaking), we have that either four or five are distinct.[8]

Note that the type of combination to be used depends on the application. For instance, for the bidding example given in the introduction, if the public key is *not*

[8]For a very recent discussion of the relationship between these notions, see Bellare et al. [3], where they show that there are indeed five distinct possibilities.

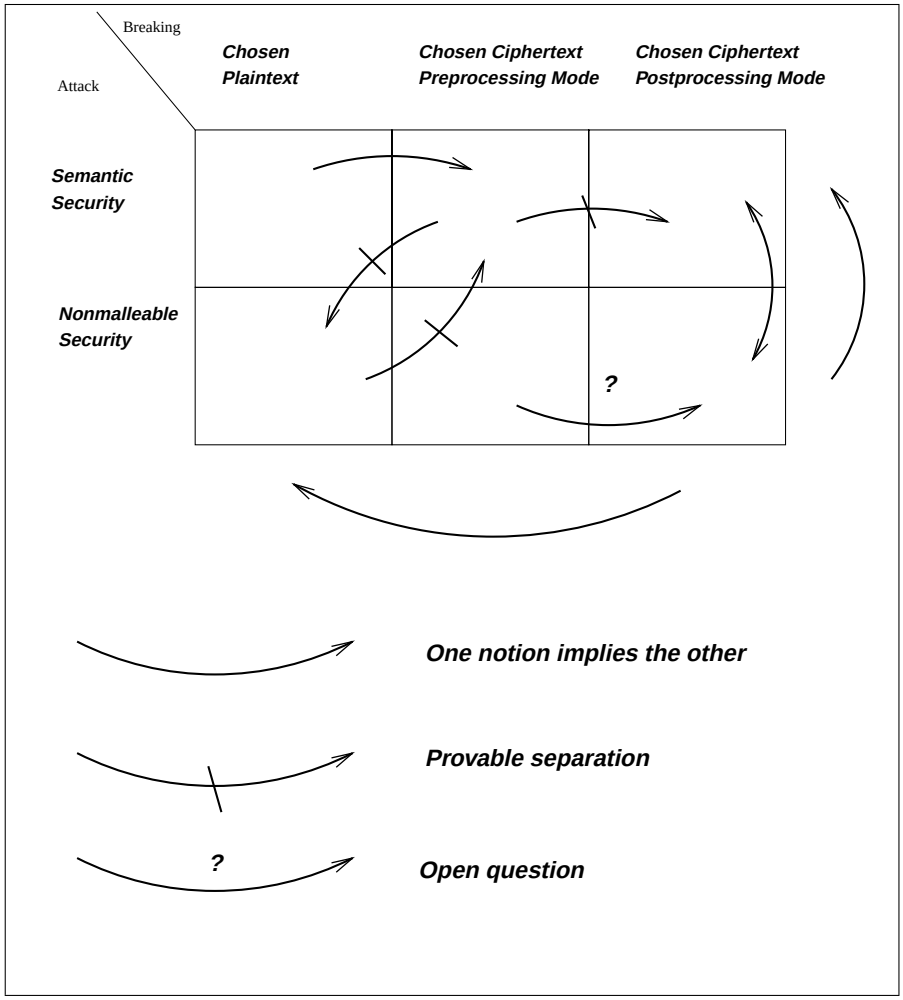


FIG. 3.1. *Relationship between security notions.*

going to be used for bidding on more than a single contract, and assuming the bids are not secret after the bids are opened, then the type of security needed is *nonmalleability against chosen plaintext attacks.* If the same public key is to be used for bidding on several contracts successively but the secrecy of nonwinning bids need not be preserved, then nonmalleability under chosen ciphertext in the preprocessing mode is required. On the other hand, if the same public key is to be used for bidding on several contracts, and the secrecy of nonwinning bids must be preserved, one should use a nonmalleable cryptosystem secure against CCA-post.

Finally, one may wonder what is the "correct" description of the notion of breaking a cryptosystem secure against CCA-post: semantic security or nonmalleable security, given their equivalence under this attack. We think it is more helpful to think in terms of nonmalleability, since the way to think about trying to break a candidate system is to think of trying to maul the target ciphertext(s). This was done (without the vocabulary of nonmalleability) in the recent work by Bleichenbacher [11] (see section 6).

3.4.3. On allowing $\mathcal{A}'$ to choose $\mathcal{M}'$. Having $\mathcal{A}'$ choose $\mathcal{M}'$, rather than inheriting $\mathcal{M}' = \mathcal{M}$ from $\mathcal{A}$, makes the adversary simulator *weaker*: the real adversary $\mathcal{A}$ is allowed to mount a chosen ciphertext attack before choosing its target distribution $\mathcal{M}$, while the adversary simulator $\mathcal{A}'$ must choose $\mathcal{M}'$ without the benefit of such an attack. Since the adversary simulator is weaker, $\forall\mathcal{A}\exists\mathcal{A}' \ldots$ becomes a stronger requirement on the cryptosystem. Our cryptosystem satisfies this strong requirement.

3.5. Public-key authentication. In this section we informally describe a method for obtaining a public-key authentication scheme based on any nonmalleable public-key cryptosystem. Our goal is to demonstrate a "real" protocol that allows cheating in case the public-key cryptosystem used is malleable.

In a public-key authentication scheme, an authenticator A chooses a public key E. The scheme permits A to *authenticate* a message m of her choice to a second party B. Similar to a digital signature scheme, an authentication scheme can convince B that A is willing to authenticate m. However, unlike the case with digital signatures, an authentication scheme need not permit B to convince a third party that A has authenticated m.

Our notion of security is analogous to that of *existential unforgeability under an adaptive chosen plaintext attack* for signature schemes [45], where we must make sure to take care of "man-in-the-middle" attacks. Let $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C)\rangle$ be an adversarially coordinated system in which $(A,B) = (C,D)$ is a public-key authentication protocol. We assume that A is willing to authenticate any number of messages $m_1, m_2, \ldots$, which may be chosen adaptively by $\mathcal{A}$. We say that $\mathcal{A}$ successfully attacks the scheme if $\psi(C)$ (under control of $\mathcal{A}$ and pretending to have A's identity) succeeds in authenticating to D a message $m \neq m_i$, $i = 1, 2, \ldots$.

Protocol $\mathcal{P} = (A, B)$ for A to authenticate message m to B. A's public key is E, chosen according to $\mathcal{S}$, a nonmalleable public-key cryptosystem secure against CCA-post (e.g., the one from section 3.2).

1. A sends to B: "A wishes to authenticate m." (This step is unnecessary if m has previously been determined.)
2. B chooses $r \in_R \{0,1\}^n$ and computes and sends the "query" $\gamma \in_R E(m \circ r)$ to A.
3. A decrypts γ and retrieves r and m. If the decryption is of the right format (i.e., the first component of the decrypted pair corresponds to the message that is to be authenticated), then A sends r to B.

LEMMA 3.7. *Given an adversary $\mathcal{B}$ that can break the authentication protocol $\mathcal{P}$ with probability ρ, one can construct an adversary $\mathcal{A}$ for breaking the (presumed nonmalleable) encryption scheme E with probability at least $\rho/p(n) - 2^{-n}$ for some polynomial p.*

Proof. The procedure for $\mathcal{A}$ to attack the cryptosystem is as follows. Assume A's public key is E and that the adversary $\mathcal{A}$ has access to a decryption box for E. Therefore $\mathcal{A}$ can simulate the system $\langle (A,B),(C,D),\ \mathcal{B} : \psi(B) \leftrightarrow \psi(C)\rangle$, where $(A,B) = (C,D) = \mathcal{P}$. Note that since this is a simulation, $\mathcal{A}$ can control the messages sent by $\psi(D)$ in the simulation. Run the system $\langle (A,B),(C,D),\ \mathcal{B} : \psi(B) \leftrightarrow \psi(C)\rangle$ until $\psi(C)$, under control of $\mathcal{B}$, is about to authenticate to D a message $m \neq m_i$, $i = 1, 2 \ldots$ not authenticated by A. (In case it is not clear whether D accepts or not, then we just guess when this occurs, whence the polynomial degradation of ρ.) The distribution $\mathcal{M}$ on messages that $\mathcal{A}$ will attempt to maul is $\mathcal{M}_m = \{(m,r) | r \in_R \{0,1\}^n\}$. Given γ as the challenge ciphertext, $\mathcal{A}$ lets $\psi(D)$ send the query γ in the simulation. Let r' be $\psi(C)$'s reply. $\mathcal{A}$ outputs $\theta \in_R E(m \circ r')$.

The distribution that $\mathcal{B}$ sees in the simulation of the adversarially coordinated system $\langle (A,B),(C,D),\ \mathcal{B} : \psi(B) \leftrightarrow \psi(C)\rangle$ is exactly as usual. Therefore by assumption the probability of success in authenticating m is ρ, and with probability ρ the value r' is the correct one. The relation that is violated is equality: θ and γ encrypt the same string, whereas given the distribution $\mathcal{M}_m$ the probability of producing the correct r without access to $E(m \circ r)$ is 2^{-n}. □

This solution will be of practical use as soon as the current constructions of nonmalleable cryptosystems are improved to be more practical [28]. The very recent construction of Cramer and Shoup [23] (see section 6 of this paper) makes this scheme very attractive.

REMARK 3.8. *If the cryptosystem $\mathcal{S}$ is malleable, and in particular if given an encryption of a message $\lambda \circ r$ it is easy (possibly after mounting a CCA-post or other type of attack) to generate an encryption of a message $\lambda' \circ r$, where $\lambda' \neq \lambda$ (many cryptosystems have this property), then there is a simple attack on the protocol proposed: as before $\psi(C)$ is pretending to be $\psi(A)$. To forge an authentication of a message m, when D sends challenge $\gamma = m \circ r$, $\psi(B)$ asks A to authenticate a message m' by sending the challenge $\gamma' = m' \circ r$. When A replies with r, $\psi(C)$ sends r to D, who will accept.*

REMARK 3.9. *As mentioned above, this protocol provides a weaker form of authentication than digital signatures (no third party verification). However, this can be viewed as a feature: there may be situations in which a user does not wish to leave a trace of the messages the user authenticated ("plausible deniability"). We do not know whether the protocol presented is indeed zero-knowledge in this sense, i.e., that the receiver could have simulated the conversation alone (although it is almost surely not black-box zero-knowledge* [38]*). By adding a (malleable) proof of knowledge to the string r this can be ensured in the sequential case. We do not know if the resulting zero-knowledge authentication protocol remains zero-knowledge if many executions with the same authenticator execute* concurrently*. The straightforward simulation fails. (See* [51] *for impossibility results for 4-round black-box concurrent zero-knowledge protocols.) Very recently, an approach for achieving deniable authentication in the concurrent setting based on* timing constraints *was suggested by Dwork, Naor, and Sahai* [29]*, who also present several efficient protocols in the standard model (no timing) for the sequential case.*

3.6. Nonmalleable encryption in other settings. In this section we briefly mention nonmalleable encryption in two additional settings: private-key cryptography and interactive public-key cryptography. In both cases we begin with a known semantically secure system and add authentication to achieve nonmalleability.

Private-key encryption. As mentioned in the beginning of section 3, the definition of nonmalleable security is applicable for private (or shared) key cryptography as well. For example, in their celebrated paper on a logic of authentication [16], Burrows, Abadi, and Needham give the following analysis of a scenario (the Needham–Schroeder authentication protocol) in which A and B share a key K_{AB}. Party B chooses a nonce N_b and sends an encryption of N_b under K_{AB} to A. A then responds with an encryption of $N_b - 1$ under K_{AB} in order for B

> "... to be assured that A is present currently ... Almost any function of N_b would do as long as B can distinguish his message from A's—thus, subtraction is used to indicate that the message is from A, rather than from B."

The unproved and unstated assumption here is that K_{AB} provides nonmalleable encryption; malleability completely destroys their reasoning and their proof of security, *even if the adversary's access to the system is very limited* (i.e., an attack weaker than CCA-pre).

Achieving nonmalleability in the private-key setting is much simpler and more efficient than in the public-key setting. Let K_{AB} be a private key shared by A and B. We first describe a system that is semantically secure against CCA-pre: Treat K_{AB} as (K_1, K_2) which will be used as seeds to a pseudorandom function f (see [37] for definition of pseudorandom functions, [56, 57] for recent constructions, and [58] for a recent discussion on using pseudorandom functions for encryption and authentication). In order to encrypt messages which are n bits long we need a pseudorandom function $f_K : \{0,1\}^\ell \mapsto \{0,1\}^n$, i.e., it maps inputs of length ℓ to outputs of length n, where ℓ should be large enough so as to prevent "birthdays," i.e., collision of randomly chosen elements. For A to send B a message m, A chooses a random string $r \in \{0,1\}^\ell$ and sends the pair $(r, m \oplus f_{K_1}(r))$. Semantic security of the system against CCA-pre follows from the fact that the pseudorandom function is secure against *adaptive* attacks. However, this scheme is *malleable* and not secure against CCA-post: given a ciphertext (r, c) one can create a ciphertext (r, c'), where c' is obtained from c by flipping the last bit. This implies that the corresponding plaintext also has its last bit flipped. In order to thwart such an attack we employ another pseudorandom function $g_k : \{0,1\}^{n+\ell} \mapsto \{0,1\}^\ell$ and add a third component to the message:

$$g_{K_2}(r \circ (m \oplus f_{K_1}(r))).$$

When decrypting a message (r, c, a), one should first verify that the third component, a, is indeed proper, i.e., $a = g_{K_2}(r \circ c)$. This acts as an *authentication tag* for the original encryption and prevents an adversary from creating any other legitimate ciphertext, except the ones he was given explicitly. (Recall that by definition of a pseudorandom function, seeing any number of pairs $(r, f_{K_2}(r))$ does not yield any information about $(r', f_{K_{AB}}(r'))$ for any new r' and in particular they are unpredictable.)

Since it is known that the existence of one-way functions implies the existence of pseudorandom functions [37, 48] we have the following theorem.

THEOREM 3.10. *If one-way functions exist, then there are nonmalleable private-key encryption schemes secure against CCA-post.*

Since it is known that in order to have private-key cryptography we must have one-way functions [47], we conclude the following corollary.

COROLLARY 3.11. *If any kind of private-key encryption is possible, then nonmalleable private-key encryption secure against CCA-post is possible.*

Note that the property of "self-validation" enjoyed by the above construction is stronger than needed for nonmalleability, i.e., there are nonmalleable cryptosystems that do not have this property: one can start with a nonmalleable private-key cryptosystem and add to it the possibility of encryption using a pseudorandom permutation; this possibility is never (or rarely) used by the legitimate encrypter but may be used by the adversary. The resulting cryptosystem is still nonmalleable but not self-validating, since the adversary can create ciphertexts of random messages.

For a recent application of the above construction to the security of remotely keyed encryption, see Blaze, Feigenbaum, and Naor [10].

Interactive encryption. The second setting resembles the one studied by Goldwasser, Micali, and Tong [46], in which they constructed an *interactive* public-key

cryptosystem secure against chosen ciphertext attack (see also [34, 67]). An "interactive public-key cryptosystem" requires a public file storing information for each message recipient, but this information alone is not sufficient for encrypting messages. The additional information needed is chosen interactively by the sender and receiver. To the best of our knowledge, their paper was the first to try to cope with an oracle for distinguishing valid from invalid ciphertexts in any setting (interactive or not). An interactive system is clearly less desirable than what has now come to be called "public-key cryptography," in which the public key *is* sufficient for sending an encrypted message, without other rounds of interaction.

The definitions of nonmalleable security can be easily adapted to this case, but when discussing the attack there is more freedom for the adversary, due to the interactive nature of the communication. In general, we assume that the adversary has complete control over the communication lines and can intercept and insert any message it wishes. A precise definition is outside the scope of this paper.

Our nonmalleable interactive public-key cryptosystem requires a digital signature scheme that is *existentially unforgeable against a chosen message attack* (see the introduction for an informal definition of existential unforgeability). Let (S_i, P_i) denote the private/public signature keys of player i (the model assumes that there is a public directory containing P_i for each player i that is to *receive* messages, but the sender is *not* required to have a key in the public directory). The system will also use a public-key cryptosystem semantically secure against chosen plaintext attacks.

The idea for the system is straightforward: for each interaction the receiver chooses a fresh public-key/private-key pair that is used only for one message. However, this is not sufficient, since an active adversary may intercept the keys and substitute its own keys. We prevent this behavior by using signatures. A sender j wishing to send a message m to receiver i performs the following:

1. Sender j chooses a *fresh* private/public pair of signature keys (s_j, p_j) and sends the public part, p_j, to i (lower case is used to distinguish p_j from what is in the directory);
2. Receiver i chooses a *fresh* private/public pair of *encryption and decryption* keys (E_{ij}, D_{ij}), where E_{ij} is *semantically secure against chosen plaintext attack*, and sends E_{ij} together with $S_i(E_{ij} \circ p_j)$ (i.e., a signature on the fresh public key E_{ij} concatenated with the public signature key j chose) to j; j verifies the signature and that p_j is indeed the public key it sent in step 1.
3. Sender j encrypts m using E_{ij} and sends $E_{ij}(m)$ together with $s_j(E_{ij}(m))$ to i. Receiver i verifies that the message encrypted with E_{ij} is indeed signed with the corresponding p_j.

Note that the sender may use a one-time signature scheme for (s_j, p_j), and if the receiver uses a signature scheme such as in [27, 22], then the approach is relatively efficient.

4. A nonmalleable scheme for string commitment. We present below a scheme $\mathcal{S}$ for string commitment that is nonmalleable with respect to itself (Definition 2.8). We first present $\mathcal{S}$ and show some properties of $\mathcal{S}$ important in proving its security. We then describe a knowledge extractor algorithm that works not on $\mathcal{S}$ but on $\mathcal{S}'$ which is a (malleable) string commitment protocol with a very special relation to $\mathcal{S}$: knowledge extraction for $\mathcal{S}'$ implies nonmalleability of $\mathcal{S}$. Thus, in this section, the new $\mathcal{S}'$ plays a role analogous to the role of $\mathcal{S}'$ in section 3.

Our nonmalleable scheme for string commitment requires as a building block a (possibly malleable) string commitment scheme. Such a scheme, based on pseudo-

random generators, is presented in [55] (although any computational scheme will do). The protocol described there is interactive and requires two phases: first the receiver sends a string and then the sender actually commits. However, the first step of the protocol can be shared by all subsequent commitments. Thus, following the first commitment, we consider string commitment to be a one-phase procedure. In what follows, when we refer to the string commitment in [55], we consider only the second stage of that protocol.

We also require *zero-knowledge proofs* satisfying the security requirements in [35]. These can be constructed from any bit commitment protocol [40].

Before we continue it would be instructive to consider the protocol of Chor and Rabin [21]. They considered the "usual" scenario, where all n parties know of one another and the communication is synchronous and proceeds in rounds. Their goal was for each party to prove to all other parties possession of knowledge of a decryption key. Every participant engages in a sequence of proofs of possession of knowledge. In some rounds the participant acts as a prover, proving the possession of knowledge of the decryption key, and in others it acts as a verifier. The sequence is arranged so that every pair of participants A, C is separated at least once, in the sense that there exists a round in which C is proving while A is not. This ensures that C's proof is independent of A's proof.

Running this protocol in our scenario is impossible; for example, (1) we make no assumptions about synchrony of the different parties, and (2) in our scenario the parties involved do not know of one another. However, we achieve a similar effect to the technique of Chor and Rabin by designing a carefully ordered sequence of actions a player must make, as a function of an identifier composed of its external identity, if one exists, and some other information described below.

4.1. The nonmalleable string commitment scheme $\mathcal{S}$. Protocol $\mathcal{S}$ consists of two general stages. The first is a string commitment as in [55]. The second stage, basic commit with knowledge, consists of the application of many instances of a new protocol, called **BCK**, to the string committed to in the first stage.

Following the commit stage of two string commitment protocols, deciding whether they encode the same string is in NP. Therefore there exists a zero-knowledge proof for equality of two committed values. This will be used repeatedly during each execution of **BCK**, which we now describe. In the following, n is a security parameter.

Protocol BCK(α) (assumes the committer has already committed to α). Concurrently run n instances of the following three steps. All instances of each step are performed at once.

- **BCK**1 (commit): Committer selects random $x_0, x_1 \in \{0,1\}^k$, where $k = |\,\alpha\,|$, and commits to both of them using the protocol in [55].
- **BCK**2 (challenge): Receiver sends committer a random bit $r \in \{0,1\}$.
- **BCK**3 (response): Committer reveals x_r and $x_{1-r} \oplus \alpha$, and engages in a proof of consistency of $x_{1-r} \oplus \alpha$ with the initial commitment to α and the commitment to x_{1-r} in **BCK**1. The proof of consistency with the initial commitment is done for all n instances together as a single statement.

The interactive proof in **BCK**3 is a proof of consistency; it need not be proof of knowledge in the sense of [31].

REMARK 4.1. *From $\alpha \oplus x_{1-r}$, x_{1-r}, and the proof of consistency, one can obtain α. This is why we call the protocol Basic Commit with Knowledge (of α).*

Note also that the interactive proof is of consistency; it is not a proof of knowledge in the sense of [31].

(A, B) **interaction**	(C, D) **interaction**
BCK1(α)	
	BCK1(β)
	BCK2(β)
BCK2(α)	
BCK3(α)	
	BCK3(β)

FIG. 4.1. **BCK**(α) *is useful to* **BCK**(β).

In the rest of the section we consider each **BCK***i* as single operation, thus it can be viewed as an operation on an n-dimensional vector or array. Note that **BCK**1 and **BCK**2 are indeed "instantaneous," in that each requires a single send, while **BCK**3, due to its interactive nature, requires more time to carry out. We frequently refer to an instance of **BCK** as a *triple.*

In the **BCK** stage of $\mathcal{S}$ we apply **BCK** repeatedly for the same string, α. However, **BCK** may itself be malleable. To see this, conceptually label the three steps of **BCK** as commitment, challenge, and response, respectively. Consider an $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$ in which $(A, B) = (C, D) =$ **BCK**. Then $\psi(C)$ can make its commitment depend on the commitment of $\psi(A)$, $\psi(B)$ can make its challenge to $\psi(A)$ depend on the challenge that $\psi(D)$ poses to $\psi(C)$, and $\psi(C)$ can respond to the challenge with the "help" of $\psi(A)$'s response to $\psi(B)$ (see Figure 4.1 for the timing of events). In this case the triple between $\psi(A)$ and $\psi(B)$ is, intuitively, useful to $\psi(C)$. The **BCK** stage of $\mathcal{S}$ interleaves executions of **BCK** so as to ensure that in every execution there is some triple for which no other triple is useful. This is analogous to Chor and Rabin ensuring that for every pair of participants A, C there exists a round in which C is proving knowledge while A is not. We say such a triple is *exposed* (defined precisely below). This is the key idea in the construction.

The next two *sixplet* protocols perform a pair of distinct instances of **BCK**(α) in two different interleaved orders. To distinguish between the two instances of **BCK** we will refer to the operation taking place at each stage and the associated variables. Thus α_i and α_{i+1} are two distinct applications of **BCK**. These sixplet protocols will be used to ensure the existence of an exposed triple in the **BCK**. The intention of the spacing of the presentation is to clarify the difference between the protocols. It has no meaning with respect to the execution of the protocols.

0**-sixplet**	1**-sixplet**
BCK1(α_i)	**BCK**1(α_i)
BCK2(α_i)	
BCK3(α_i)	**BCK**1(α_{i+1})
	BCK2(α_{i+1})
BCK1(α_{i+1})	**BCK**3(α_{i+1})
BCK2(α_{i+1})	
BCK3(α_{i+1})	**BCK**2(α_i)
	BCK3(α_i)

The difference between the two protocols is the order in which we interleave the stages of the two distinct instances of the **BCK** protocol.

Using these sixplets, we can present the scheme $\mathcal{S}$. The identifier I used in the scheme is the concatenation of the original identity with the commitment for α at stage 1 (by the "commitment" we mean a transcript of the conversation). I_j denotes the jth bit of I. To force an exposed triple we will use the fact that every two identifiers differ in at least one bit. This is exactly the same fact that was exploited by Chor and Rabin in the synchronous "everyone-knows-everyone" model to enforce the condition that for every pair of provers $A \neq C$, there is a round in which C is proving but A is not [21]. The same fact is used in both cases for the same purpose, but we do it without any assumption of synchrony and without any assumption that each processor knows of all other processors in the system.

$\mathcal{S}$: Nonmalleable commitment to string α.

- Commit to α (e.g., using the protocol in [55]).
- For $j = 1$ to $|\, I \,|$
  Execute an I_j-sixplet
  Execute a $(1 - I_j)$-sixplet

  End

For simplicity we will assume that all identifiers I are n bits long. Each I_j-sixplet and each $(1 - I_j)$-sixplet involves two executions of **BCK**, and each of these in turn requires n concurrent executions of **BCK**1, followed by n concurrent executions of **BCK**2 and then of **BCK**3. Thus, a nonmalleable string commitment requires invoking each **BCK**i a total of $4n^2$ times.

4.2. Properties of $\mathcal{S}$. We now show some properties of $\mathcal{S}$ that allow us to prove its nonmalleability. Suppose that $(A, B) = (C, D) = \mathcal{S}$, and suppose further that adversary $\mathcal{A}$ controls $\psi(B)$ and $\psi(C)$. Let x be the identifier used by $\psi(A)$ and y that used by $\psi(C)$. If the original identities of $\psi(A)$ and $\psi(C)$ are different or if the strings to which they commit are different, then $x \neq y$. (Thus the only case not covered is copying.) Note also that, given the proofs of consistency, both sender and receiver know at the end of the commitment protocol whether or not the sender has succeeded in committing to a well-defined value. Thus, the event of *successful commitment to some value* by $\psi(C)$ is independent of the value committed to by $\psi(A)$.

Each run of the two interactions determines specific times at which the two pairs of machines exchange messages. The adversary can influence these times, but the time at which an interaction takes place is well defined. Let σ_x and σ_y be the respective schedules. For $1 \leq i \leq 2n$, let

- τ_1^i be the time at which **BCK**1 begins in the ith instance of **BCK** in σ_x;
- τ_2^i be the time at which **BCK**2 ends in the ith instance of **BCK** in σ_x.

In contradistinction, let

- δ_1^i be the time at which **BCK**1 ends in the ith instance of **BCK** in σ_y;
- δ_2^i be the time at which **BCK**2 begins in the ith instance of **BCK**2 in σ_y.

Finally, let

- τ_3^i and δ_3^i denote the times at which **BCK**3 ends in the ith instances of **BCK** in σ_x and σ_y, respectively.

These values are well defined because each **BCK**i involves sequential operations of a single processor. We do not assume that these values are known to the parties involved—there is no "common clock."

We can now formalize the intuition, described above, of what it means for a triple in σ_x to be useful to a triple in σ_y. Formally, the ith triple in σ_x is *useful* to the jth

triple in σ_y if three conditions hold: (1) $\tau_1^i < \delta_1^i$; (2) $\delta_2^j < \tau_2^i$; and (3) $\delta_3^j > \tau_2^i$ (see Figure 4.1).

Let $\Gamma^{(i)} = \{j \mid \delta_1^j > \tau_1^i \ \wedge \ \delta_2^j < \tau_2^i \ \wedge \ \delta_3^j > \tau_2^i\}$. $\Gamma^{(i)}$ is the set of indices of triples between $\psi(C)$ and $\psi(D)$ for which the ith triple between $\psi(A)$ and $\psi(B)$ can be useful. We say that a triple j is *exposed* if $j \notin \Gamma^{(i)}$ for all i. Our goal is to show that there is at least one exposed triple in any schedule. Intuitively, exposed triples are important because the committer is forced to act on its own, without help from any other concurrent interaction. Technically, exposed triples are important because they allow the knowledge extractor to explore the adversary's response to two different queries, without the cooperation of $\psi(A)$.

CLAIM 4.1. *For all* $i \ |\Gamma^{(i)}| \leq 1$.

Proof. By inspection of the possible interleavings, there exists at most one j for which $\delta_2^j < \tau_2^i$ and $\delta_3^j > \tau_2^i$. □

CLAIM 4.2. *If* $j_1 \in \Gamma^{(i_1)}$, $j_2 \in \Gamma^{(i_2)}$, *and* $j_1 < j_2$, *then* $sixplet(i_1) \leq sixplet(i_2)$, *where* $sixplet(i)$ *denotes the index of the sixplet containing the* i*th triple.*

Proof. Assume to the contrary that $sixplet(i_2) < sixplet(i_1)$. This implies that $\tau_2^{i_2} < \tau_1^{i_1}$. By definition, $j_1 \in \Gamma^{(i_1)}$ implies $\tau_1^{i_1} < \delta_1^{j_1}$. Similarly, $j_2 \in \Gamma^{(i_2)}$ implies $\delta_1^{j_2} < \tau_2^{j_2}$. Thus, $\delta_1^{j_2} < \delta_1^{j_1}$. This contradicts the assumption that $j_1 < j_2$. □

CLAIM 4.3. *Let triples* $2k-1, 2k$ *form a* 0*-sixplet in* σ_x, *and let triples* $2\ell-1, 2\ell$ *form a* 1*-sixplet in* σ_y. *Then there exists a* $j \in \{2\ell-1, 2\ell\}$ *such that neither* $2k-1$ *nor* $2k$ *is useful to triple* j *in* σ_y.

Proof. Assume to the contrary that the claim does not hold. Thus, both triples have a useful triple in $\{2k-1, 2k\}$. By Claim 4.1 $|\Gamma^{(2k-1)}| \leq 1$, and $|\Gamma^{(2k)}| \leq 1$. Therefore, each of the two triples should be useful. A simple look at the time scale implies that for either matching between the pairs, it should be the case that $\tau_1^{2k} < \delta_1^{2\ell}$. Thus, $\tau_2^{2k-1} < \delta_2^{2\ell}$ and $\tau_2^{2k-1} < \delta_2^{2\ell-1}$. This implies that $2k-1$ is not useful to either of the triples, which is a contradiction. □

Notice that the reverse claim does not hold.

LEMMA 4.2. *For any* $x \neq y$ *and for any two sequences* σ_x *and* σ_y, *there exists an exposed triple in* σ_y.

Proof. From Claims 4.1 and 4.2, if none of triples 1 through j are exposed and $j \in \Gamma^{(i)}$, then $sixplet(i) \geq sixplet(j)$. Since $x \neq y$, there exists a bit, say the jth one, at which their IDs differ. Since the scheme uses both an I_j-sixplet and $1-I_j$-sixplet, there exists some k such that the kth sixplet in σ_x is a 0-sixplet while the kth sixplet in σ_y is a 1-sixplet. The lemma now follows from Claim 4.3. □

4.3. The knowledge extractor. Consider an adversarially coordinated system $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$, where (A,B) and (C,D) are both instances of $\mathcal{S}$. Intuitively, if $\psi(C)$ succeeds in committing to a string β, then our goal is to extract β. To achieve this we devise a somewhat different protocol, called $\mathcal{S}'$, on which the extractor operates, and from which it extracts β. This new protocol is a string commitment protocol that is not necessarily nonmalleable. In the next section we prove that extraction of β from the $\mathcal{S}'$-adaptor-$\mathcal{S}$ system implies the nonmalleability of $\mathcal{S}$.

The string commitment scheme $\mathcal{S}'$ consists of a committer P and a receiver Q and takes a parameter m. (As we will see, $m = |I| \frac{16}{(\varepsilon^2 \log \varepsilon)}$, where ε is how close we would like the extraction probability to be to the probability of successful completion of the protocol by $\mathcal{A}$.)

Protocol $\mathcal{S}'$: P Commits to a string α.

- Commit to α (e.g., using the protocol in [55]).
- Repeat m times:
  1. Q chooses a bit b and requests a b-sixplet; according to additional inputs, Q requests that the b-sixplet be augmented by an *additional* proof of consistency in step **BCK**3 of either triple in the b-sixplet;
  2. P and Q run a (possibly augmented) b-sixplet.

From the semantic security of the regular string commitment protocol and from the zero-knowledge properties, a simulation argument yields the following lemma.

LEMMA 4.3. *For any strategy of choosing the sixplets and for any receiver Q', the string commitment protocol $\mathcal{S}'$ is semantically secure.* □

We provide an adaptor that allows us to emulate to $\mathcal{A}$ (and its controlled machines) a player A that executes $\mathcal{S}$, whereas in reality $\psi(B)$ (under control of $\mathcal{A}$) communicates with the sender P of $\mathcal{S}'$. $\mathcal{S}'$ has been designed so that it can actually tolerate communicating with many copies of $\mathcal{A}$, with messages from the different copies being "multiplexed" by the adaptor.

In more detail, suppose that player $\psi(P)$ is running the sender part of $\mathcal{S}'$ and that player $\psi(B)$ is supposed to run the receiver part of $\mathcal{S}$. ($\psi(B)$ might deviate from the protocol as written, but the communication steps are as in $\mathcal{S}$.) It is not hard to construct an adaptor that operates between P and B: whenever (A, B) calls for a b-sixplet the adaptor "pretends it is Q" and asks for a b-sixplet; then $\psi(B)$ and $\psi(P)$ run the b-sixplet. It should be clear that the distribution of conversations that $\psi(B)$ sees when it participates in $\mathcal{S}$ and the distribution of conversations it sees when it participates through the adaptor in $\mathcal{S}'$ are identical.

We are now ready to present the extractor. Suppose that in the adversarially coordinated system the probability that $\psi(C)$ completes its part successfully is ρ. Following the commit stage (during which C may or may not have committed in any meaningful way), we cannot in general hope to extract β with probability greater than ρ. However we can get arbitrarily close: we will show that for any ε we can successfully extract β with probability at least $\rho - \varepsilon$.[9]

Fix $\varepsilon > 0$. The knowledge extractor begins to run $\mathcal{S}' = (P, Q)$ and $\mathcal{S} = (C, D)$ with the adaptor arranging that $\mathcal{A}$ cannot distinguish this from the adversarially coordinated system $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$ (see Figure 4.2) in which $(A, B) = (C, D) = \mathcal{S}$.

Once $\psi(C)$ completes the first (commitment) stage of $\mathcal{S}$, the extractor freezes the random tape of $\mathcal{A}$.

$\mathcal{A}$ now defines a tree according to all possible messages sent by A and D. The tree contains A-nodes and D-nodes, according to which of the two is the next one to send a message. The root of the tree corresponds to the point at which the tape was frozen. Thus, the branching at each node is all possible messages that either A or D can send at that point. In order to exploit Remark 4.1, we will be interested in D-nodes corresponding to a **BCK**2 step. The branches correspond to the different possible challenge vectors that D can send in this step. In what follows, these are the only types of D-node that we will consider. To enable us to follow more than a single path (that is, to *fork*) in the tree, we keep at each such D-node a snapshot of

[9] The extraction procedure runs in a fixed polynomial time (in n and ε^{-1}) and succeeds only with probability p. This leads to an ε-malleable, which we suspect is sufficient "for all practical purposes." A modification of the procedure runs in *expected* polynomial time and succeeds with probability ρ (the best possible), yielding liberal nonmalleability. See Remark 4.4.

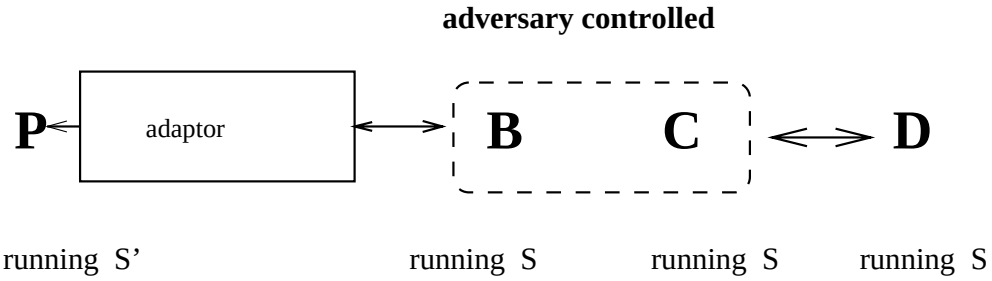


FIG. 4.2. *The $\mathcal{S}'$-adaptor-$\mathcal{S}$ system used in constructing the extractor.*

the current state, i.e., a copy of $\mathcal{A}$'s tapes and the states of A and D.

A node v is *good* if all the communication between C and D up to v is legal (according to the nonmalleable protocol $\mathcal{S}$) and C successfully opened and proved whenever it was asked to do so. Our goal is to identify two nodes having the following properties: (1) at each of the two, C has just completed a **BCK**3 step; (2) the paths to the two nodes depart in a branching at a D-node. As noted in Remark 4.1, given two such nodes we can extract β.

To identify such a pair of nodes, choose $\ell = \frac{16}{(\varepsilon^2 \log \varepsilon)}$, and run the following extraction procedure ℓ times, each time starting again at the root of tree. (Recall that the root of the tree corresponds to the point at which the tape was frozen; we do not restart (C, D) each time the extraction procedure is repeated.)

By Lemma 4.2 every path to a good leaf contains an exposed triple. Run the $\mathcal{S}'$-adaptor-$\mathcal{S}$ system until an exposed triple j in σ_y is reached (or we reach a bad node). We partition the exposed triples into two types according to the interleavings (the interleavings are shown pictorially after the types are formally defined):

- j is of the first type if for all i $\tau_1^i > \delta_1^j$ (nothing happened yet in σ_x between $\psi(A)$ and $\psi(B)$) or for all i s.t. $\tau_1^i < \delta_1^j$ we have $\tau_2^i < \delta_2^j$ (the challenge in σ_x ends before the challenge in σ_y begins).
- j is of the second type if it is not of the first type and for all i s.t. $\tau_1^i < \delta_1^j$ and $\tau_2^i \geq \delta_2^j$ we have $\tau_2^i > \delta_3^j$ (the challenge in σ_x ends after the reply in σ_y ends, so $\psi(C)$ can't use the answers from (A, B) to help it answer challenges from $\psi(D)$).

In the first type of exposure, for each i there are two possible interleavings:

$$\begin{array}{cccccc} \tau_1^i & & & & \tau_1^i & \\ & \delta_1^j & \text{or} & & \tau_2^i & \\ \tau_2^i & & & & & \delta_1^j \\ & \delta_2^j & & & & \delta_2^j \end{array} .$$

Thus, in the first type of exposure, there exists a time t, $\delta_1^j \leq t \leq \delta_2^j$ such that for all i, $\tau_1^i \leq t \Rightarrow \tau_2^i \leq t$. The time t is the maximum of δ_1^j and the maximum over all i such that $\tau_1^i \leq \delta_1^j$, of τ_2^i. In this case, intuitively, for every i such that the values committed to by $\psi(C)$ in **BCK**(j) may depend on the values committed to by $\psi(P)$ in **BCK**(i), the queries made by $\psi(Q)$ to $\psi(P)$ about these values are independent of the queries made by $\psi(D)$ to $\psi(C)$. It follows that $\psi(C)$ can't get any help from $\psi(P)$ in answering $\psi(D)$'s queries in **BCK**(j).

At the point t defined above, P has no triples of which step **BCK**1 has completed but **BCK**2 has not yet ended, and thus A doesn't play a part in $\mathcal{S}$ right now. At this point we fork: the extractor creates a new copy of $\mathcal{A}$ and D, and runs both this copy and the original, with each copy of D making independent random challenges in **BCK**2 of triple j. Note that with overwhelming probability any two such challenge vectors differ in some position. Since at the point t defined above the challenges sent to A in **BCK**2 of triple i are already fixed, the two copies of **BCK**3 of triple i will differ only in the proofs of consistency. The adaptor multiplexes to P the two proofs of consistency. This completes the treatment of the first type of exposed triple.

In the second type of exposed triple, the exposure does not become evident until δ_3^j. At any point in time there are at most two triples between A and B that are *open*, in that step **BCK**1 has been executed but **BCK**2 has not. Say that at δ_1^j the open triple is the ith triple; if there are two open triples then they are the ith and $(i+1)$st ones. We know that $\tau_1^i < \tau_1^{i+1} < \delta_1^j$ and $\tau_2^i > \tau_2^{i+1} > \delta_1^j$ and $\tau_2^i > \delta_3^j$. We distinguish between two cases: (a) $\tau_2^{i+1} < \delta_2^j$ and (b) $\tau_2^{i+1} > \delta_3^j$ (since j is exposed it cannot be the case that $\delta_2^j < \tau_2^{i+1} < \delta_3^j$). We show the interleavings and mark the forking points with asterisks:

Case (a)			**Case (b)**	
τ_1^i			τ_1^i	
τ_1^{i+1}			τ_1^{i+1}	
	δ_1^j			δ_1^j
τ_2^{i+1}				$*$
$*$		or		δ_2^j .
τ_3^{i+1}				δ_3^j
	δ_2^j		τ_2^{i+1}	
	δ_2^j		τ_3^{i+1}	
τ_2^i			τ_2^i	

In Case (a) we fork right after τ_2^{i+1}, running a copy of $\mathcal{A}$ until the conclusion of triple j in the copy.

Although this means there will be two copies of **BCK**3$(i+1)$, they will differ only in their interactive proofs of consistency: the challenges are fixed by time τ_2^{i+1}. (Note that we can assume that $\tau_3^{i+1} < \delta_2^j$ because the replies to the challenges and the statements to be proved by P in **BCK**3$(i+1)$ are completely determined by **BCK**1 and **BCK**2 and are therefore are completely determined by time τ_2^{i+1}. Moreover, the challenges sent in **BCK**2(j) by D are independent of **BCK**2$(i+1)$ because **BCK**2$(i+1)$ ends before **BCK**2(j) starts and D is nonfaulty.) D makes independent challenges in the two runs. We will not run the original beyond δ_3^j. The communication with A is limited in the original execution to the replies to the challenges sent in **BCK**2$(i+1)$ and the zero-knowledge proof of consistency in **BCK**3$(i+1)$. However, since the challenges in the two copies are the same, and since in $\mathcal{S}'$ the committer P is willing to repeat this proof, when running the copy we simply ask for a repeated proof of consistency and continue as before. We stop when the copy finishes **BCK**3 of the jth triple. Note that in the copy the jth triple need not be exposed (this depends on τ_2^i).

Case (b) is simpler: we fork right after δ_1^j. In the original $\psi(B)$ does not communicate with P until δ_3^j, so we simply continue with the copy until it finishes **BCK**3 of the jth triple. Here again we have that j need not be exposed in the copy.

In exploiting either type of exposure, if in both branches (the original and the copy) the proof of consistency in **BCK**3(j) succeeds, then in triple j the extractor obtains the answers to two different and independent queries; hence β can be extracted. The significance of the zero-knowledge proof of consistency is that it allows the extractor to know whether any trial is successful. Therefore if any trial is successful the extractor succeeds. This completes the description of the extractor.

REMARK 4.4. *To remove the dependency on ε, at the expense of running in expected polynomial time, i.e., to obtain liberal nonmalleability (see Remark 2.9), pursue the following strategy. Choose a random path to a leaf in the tree defined above. If the leaf is not good, then abort. Otherwise "extract at all costs." That is, repeat until done:*

1. *Choose a random path to a leaf. If this leaf is good, then add to a set of previously chosen paths.*
2. *For all previously chosen paths (necessarily ending with a good leaf), attempt once (again) to extract as described above.*

We now show that its probability of success is high. At each node v of the tree we can define the probability of success, $\rho(v)$, i.e., the probability that the communication between A and D leads to a good leaf. Let ρ_0 be the probability of success at the root. Notice that by definition, the expected value of ρ_0 is ρ.

LEMMA 4.5. *In each run of the above experiment the value of β is successfully extracted with probability $\rho_0^2/4 - 1/2^{2n}$.*

Proof. Consider a random root-leaf path w in the tree (the randomness is over the coin-flips of A and D). At each node v let $\rho(v)$ denote the probability, taken over choices of A and of D, of successfully completing the execution from v. Let $\rho(w)$ be the minimum along the execution path w. Note that $\rho(w)$ is a random variable.

CLAIM 4.4. *With probability at least $\rho_0/2$ we have $\rho(w) > \rho_0/2$.*

Proof. The probability of failure is $1 - \rho_0$. Let $\mathcal{V}$ be the set of nodes v s.t. $\rho(v) < \rho_0/2$ and for no parent u of v is $\rho(u) < \rho_0/2$ (i.e., $\mathcal{V}$ consists of the "first" nodes s.t. $\rho(v) < \rho_0/2$, and hence no member of $\mathcal{V}$ is an ancestor of another). We know that $\Pr[\rho(w) \le \rho_0/2] \le \sum_{v\in\mathcal{V}} \Pr[v \text{ is reached}]$. On the other hand, the probability of failure, $1 - \rho_0$, is

$$\sum_{v\in\mathcal{V}} \Pr[v \text{ is reached}](1 - \rho(v)) \ge (1 - \rho_0/2) \sum_{v \text{ s.t. } \rho(v)\le\rho_0/2} \Pr[v \text{ is reached}].$$

Therefore $\Pr[\rho(w) \le \rho_0/2] \le \frac{1-\rho_0}{1-\rho_0/2} = 1 - \frac{\rho_0/2}{1-\rho_0/2} < 1 - \rho_0/2$. □

Thus, with probability $\rho_0/2$ the main path we take succeeds. The experiment branches at a point with probability of success $\rho_0/2$. The probability of success of each branch is independent. Therefore, the new branch succeeds with probability $\rho_0/2$. Excluding a small probability $1/2^{2n}$ that both branches choose identical strings, the experiment succeeds with probability $\rho_0^2/4 - 1/2^{2n}$. □

To obtain an analogous result for the liberal nonmalleability extraction procedure outlined in Remark 4.4, consider a random path that yields a good leaf. By Claim 4.4, $\Pr[\rho(w) > \rho/2] \ge 1/2$, that is, with probability 1/2 a good leaf is also a good investment for extraction. Thus the "extract at all costs" procedure runs in expected time $O(1/\rho_0)$ and the probability this extraction is invoked is ρ_0, yielding expected polynomial time (taking the usual precautions against running forever).

Continuing with the proof of ε-malleability, with probability $\rho - \varepsilon/2$, the probability of success at the root, ρ_0, is at least $\varepsilon/2$. The extractor makes ℓ independent

experiments. Because of the proof of consistency, extraction fails only if all experiments fail. This occurs with probability at most $(1 - \frac{\rho_0^2}{4})^\ell$. The choice of ℓ implies that the probability that the extractor succeeds, given that $\rho_0 > \varepsilon/2$, is at least

$$1 - \left(1 - \frac{\rho_0^2}{4}\right)^\ell \geq 1 - \left(1 - \frac{\varepsilon^2}{4}\right)^\ell \geq 1 - \varepsilon/2.$$

Therefore, with probability at least $\rho - \varepsilon$ the string β is extracted in at least one of the ℓ experiments. Thus we can conclude the following lemma.

LEMMA 4.6. *For any adversarially coordinated system $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C)\rangle$ in which $(A,B) = (C,D) = \mathcal{S}$, there is a knowledge extraction procedure that succeeds in extracting from the $\mathcal{S}'$-adaptor-$\mathcal{S}$ system the value committed to by $\psi(C)$ with probability arbitrarily close to ρ.* □

We can therefore conclude that, in essence, the values β obtained by the extractor are "correctly" distributed. We would like to say that when β is obtained by the extractor, then for every relation approximator R, the probability that $R(\alpha,\beta)$ outputs 1 is subpolynomially close to $\pi(\mathcal{A},R)$ (the probability that it holds under a "normal" execution). However, in the true execution $\psi(C)$ might make moves that force R to reject (for instance when the real player makes an illegal move or refuses to open with certain probability). This doesn't necessarily imply that the extraction would fail. However, such cases only help us and we can conclude the following corollary.

COROLLARY 4.7. *Let $\alpha \in_R \mathcal{D}$, and let β be obtained by the extractor. Then for every relation approximator R, either* (1) *the probability that $R(\alpha,\beta)$ outputs* 1, *where the probability space is over the choice of α and the internal coin-flips of the machines involved, is larger than $\pi(\mathcal{A},R)$ or* (2) *these two probabilities arbitrarily close.*

4.4. Extraction implies nonmalleability. In this section we reduce the nonmalleability of $\mathcal{S}$ to the semantic security of $\mathcal{S}'$. Let R be a relation approximator and let $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C)\rangle$ be an adversarially controlled system, where (A,B) and (C,D) are both instances of $\mathcal{S}$.

Recall that $R(x,x) = 0$ for all relation approximators. We view the goal of $\mathcal{A}$ (respectively, $\mathcal{A}'$) as trying to maximize $\pi(\mathcal{A},R)$ (respectively, $\pi'(\mathcal{A}',R)$). Consider the following procedure for an adversary simulator $\mathcal{A}'$ with access to the probability distribution $\mathcal{D}$ chosen by $\mathcal{A}$, on inputs to $\psi(A)$.

Procedure for $\mathcal{A}'$ on input hist(α).

1. Set $\mathcal{D}' = \mathcal{D}$.
2. Generate $\delta \in_R \mathcal{D}' = \mathcal{D}$.
3. Emulate the system $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C)\rangle$ where $\psi(A)$ is running $\mathcal{S}'$ with private input δ and $\mathcal{A}$ has access to hist(α), and if $\psi(C)$ succeeds in committing to a value γ, extract γ.
4. Output γ (that is, give γ as input to $\psi(C)$).

The structure of the proof is as follows. Let $\alpha \in_R \mathcal{D}$. We define three random variables:

1. Let β be the value, if any, committed to by C in an execution of $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C)\rangle$ in which A has input α and $\mathcal{A}$ has input hist(α). By definition, for any probabilistic polynomial-time relation approximator R, $\Pr[R(\alpha,\beta)] = \pi(\mathcal{A},R)$.
2. Let β' be obtained by extraction from $\mathcal{A}'$ in a run of the $\mathcal{S}'$-adaptor-$\mathcal{S}$ system in which P has input α and $\mathcal{A}'$ has input hist(α). Let $\tilde{\pi}(\mathcal{A},R) = \Pr[R(\alpha,\beta')]$ Intuitively, this is the probability that $\psi(C)$ commits to something related to

α in an $\mathcal{S}'$-adaptor-$\mathcal{S}$ system in which all parties have the "right" inputs, and this value is successfully extracted.

3. Let β'' be obtained by extraction from $\mathcal{A}'$ in a run of the $\mathcal{S}'$-adaptor-S system in which $\psi(P)$ has input $\delta \in_R \mathcal{D}$ but $\mathcal{A}'$ has input $\text{hist}(\alpha)$. Then $\pi'(\mathcal{A}', R) = \Pr[R(\alpha, \beta'')]$ since this is exactly the setting of the variables when $\mathcal{A}'$ receives as input $\text{hist}(\alpha)$ (see the definition of $\mathcal{A}'$ above).

We will first show that if $|\Pr[R(\alpha, \beta')] - \Pr[R(\alpha, \beta'')]|$ is polynomial, then there is a distinguisher for $\mathcal{S}'$. By the semantic security of $\mathcal{S}'$, this means that $\pi'(\mathcal{A}', R) = \Pr[R(\alpha, \beta'')]$ is very close to $\tilde{\pi}(\mathcal{A}, R) = \Pr[R(\alpha, \beta')]$. In other words, on seeing the history $\text{hist}(\alpha)$, $\mathcal{A}'$, interacting with P having input α, is essentially no more successful at committing to a value related by R to α than $\mathcal{A}'$ can be when it again has history $\text{hist}(\alpha)$ but is actually interacting with P having input δ (unrelated to α). This means that, for $\mathcal{A}'$, having the interaction with P doesn't help in committing to a value related to P's input.

Let us say that $\mathcal{A}'$ *succeeds* in an execution of the $\mathcal{S}'$-adaptor-$\mathcal{S}$ system, if $\psi(C)$ commits to a value related by R to P's input (the value to which P commits). Similarly, we say that $\mathcal{A}$ succeeds in an execution of $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$ if $\psi(C)$ commits to a value related by R to A's input. Recall that, by Corollary 4.7, either $\mathcal{A}$ is essentially equally likely to succeed as $\mathcal{A}'$, or $\mathcal{A}$ is less likely to succeed than $\mathcal{A}'$ is. So $\pi(\mathcal{A}, R)$, the probability that $\mathcal{A}$ succeeds, is essentially less than or equal to $\tilde{\pi}(\mathcal{A}, R)$, which we show in the first step of the proof to be close to $\pi'(\mathcal{A}', R)$. From this we conclude the nonmalleability of $\mathcal{S}$.

LEMMA 4.8. *If $|\tilde{\pi}(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is polynomial, then there is a distinguisher for $\mathcal{S}'$ that violates the indistinguishability of committed values.*

Proof. Assume $|\tilde{\pi}(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is polynomial. The distinguisher is as follows.

Distinguisher for $\mathcal{S}'$.

1. Create a random challenge $(\alpha_1 \in_R \mathcal{D}, \alpha_2 \in_R \mathcal{D})$.
2. Choose $i \in_R \{1, 2\}$. Emulate the system $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$, where $\psi(A)$ is running $\mathcal{S}'$ with private input α_i and $\mathcal{A}$ has access to $\text{hist}(\alpha_1)$, and extract ζ, the value committed to by $\psi(C)$ in the emulation.
3. Output $R(\alpha_1, \zeta)$.

If, in the emulation, $i = 1$, then the input to $\psi(P)$ is α_1 and so the distinguisher outputs 1 with probability $\tilde{\pi}(\mathcal{A}, R)$. Similarly, if in the emulation $i = 2$, then the input to $\psi(P)$ is α_2, and so the distinguisher outputs 1 with probability $\pi'(\mathcal{A}', R)$. Since by assumption these two quantities differ polynomially, we have a polynomial distinguisher for commitments in $\mathcal{S}'$. □

COROLLARY 4.9. *$|\tilde{\pi}(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial.* □

THEOREM 4.10. *The string commitment scheme $\mathcal{S}$ is* (1) *ε-malleable and* (2) *liberal nonmalleable.*

Proof. (1) By Corollary 4.7, $\pi(\mathcal{A}, R) < \tilde{\pi}(\mathcal{A}, R)$ or the two can be made arbitrarily close. Thus $\mathcal{A}$ is at most ε more likely to successfully commit to a value related by R to the value committed to by $\psi(A)$ than $\mathcal{A}'$ is able to commit to a value related by R to the value committed to by $\psi(P)$. However, by Lemma 4.8, $\pi'(\mathcal{A}', R)$ is subpolynomially close to $\tilde{\pi}(\mathcal{A}, R)$; that is, interacting with P does not help $\mathcal{A}'$ to commit to a value related to the value committed to by $\psi(P)$.

For (2), note that the expected polynomial time extraction procedure described in Remark 4.4 succeeds with probability ρ, so the ε difference disappears. □

REMARK 4.11. *The number of rounds in the above protocol is proportional to the length of I. However, the number of rounds may be reduced to $\log |I|$ using the*

following: Let $n = |I|$. To commit to string α, choose random $\alpha_1, \alpha_2, \ldots, \alpha_n$ satisfying $\bigoplus_{i=1}^{n} \alpha_1 = \alpha$. For each α_i (in parallel) commit to α_i with identity (i, I_i) (i concatenated with the ith bit of the original identity). Let $\mathcal{F}$ (for fewer) denote this string commitment protocol.

To see why $\mathcal{F}$ is secure, consider an adversary with identity $I' \neq I$ who commits to α'. For $I' \neq I$ there must be at least one i such that $I'_i \neq I_i$ (we assume some prefix free encoding). This i implies the nonmalleability of the resulting scheme: Make α_j for $j \neq i$ public. Since all the identities of the form (j, I'_j) are different than (i, I_i) we can extract all the α'_j's and hence α'.

Using this approach, the result of Chor and Rabin [21] *can be improved to require* $\log \log n$ *rounds of communication (down from* $\log n$ *rounds). Recall that their model differs from ours in that they assume all n parties are aware of each other and that the system is completely synchronous.*

REMARK 4.12. 1. *As we have seen, the proofs of consistency aid in the extraction procedure. Interestingly, they also ensure that if there are many concurrent invocations of (A, B), call them $(A_1, B_1), \ldots, (A_k, B_k)$, such that the adversary controls all the $\psi(B_i)$ and $\psi(C)$, then if C commits to a value β to D then β is essentially unrelated to all the α_i committed to by the A_i in their interactions with the B_i. As in section* 3.4.1, *this is shown by a hybrid argument.*

2. *There is a lack of symmetry between our definitions of nonmalleable encryption and nonmalleable string commitment: the first requires that it should be computationally difficult, given $E(\alpha)$, to generate a* vector *of encryptions $(E(\beta_1), \ldots, E(\beta_n))$ s.t. $R(\alpha, \beta_1, \ldots, \beta_n)$ holds, while the second requires only that access to a commitment to a string α should not help in committing to a* single *related string β. It is possible to modify the definition to yield this stronger property. Roughly speaking, we add a fictitious step after the adversary attempts to commit to its values, in which the adversary specifies which successfully committed values will be the inputs to the relation approximator R. The extraction procedure is then modified by first running $\mathcal{S}'$ with a simulation of $\mathcal{A}$ to see which commitments succeed. Then we argue that with high probability the extraction procedure succeeds on all of these. This follows from the high probability of success during any single extraction* (*Lemma* 4.6). *We chose not to use the extended definition because it would complicate the proofs even beyond their current high level of complexity.*

3. *The weaker definition does not imply the stronger one: the protocol $\mathcal{F}$ of Remark* 4.11 *is a counterexample. Let $(A, B) = \mathcal{F}$ and let $\psi(A)$, running $\mathcal{F}$, commit to α by splitting it into $\alpha_1, \ldots, \alpha_n$. Let $(C_1, D_1) = \cdots = (C_n, D_n) = \mathcal{F}$. If the $n+1$ parties $\psi(C_1), \ldots, \psi(C_n)$ have identities such that for each i the ith bit of the identity of $\psi(C_i)$ equals the ith bit of the identity of $\psi(A)$, then the parties $\psi(B), \psi(C_1), \ldots \psi(C_n)$ can collude as follows. Each $\psi(C_i)$ commits to the string $\beta_i = \alpha_i$ by splitting it into $\beta_i = \beta_{i1} \oplus \cdots \oplus \beta_{in}$, where $\beta_{ii} = \alpha_i$ and $\beta_{ij} = 0^{|\alpha_i|}$. In this way the colluding parties can arrange to commit to $\beta_1, \ldots, \beta_n$ such that the exclusive-or of the β's equals α.*

This counterexample also illustrates why the technique for reducing rounds described in Remark 4.11 *cannot be iterated to obtain a constant round protocol.*

5. Zero-knowledge proofs and general nonmalleable zero-knowledge interactions. For the results in this section we assume the players have unique identities. Let $(A, B)[a, b]$ be a zero-knowledge interactive protocol with valid set Π of input pairs. Recall from section 2.1 that (A, B) is *zero-knowledge* with respect to B if for every $\psi(B)$ under control of a polynomial-time bounded adversary $\mathcal{A}$, there exists a simulator *Sim* such that the following two ensembles of conversations are indistin-

guishable. In the first ensemble, $\mathcal{A}$ chooses a distribution $\mathcal{D}$ consistent with Π, a pair (α, β) is drawn according to $\mathcal{D}$, $\psi(A)$ gets α, $\psi(B)$ gets β, and the interaction proceeds and produces a conversation. In the second ensemble, and adversary simulator $\mathcal{A}'$ with the same computational power as $\mathcal{A}$ chooses a distribution $\mathcal{D}'$ consistent with Π, $(\alpha, \beta) \in_R \mathcal{D}'$ is selected, $\mathcal{A}'$ is given β, and produces a simulated conversation.

We construct a compiler $\mathcal{C}$, which, given *any* zero-knowledge interaction (A, B), produces a zero-knowledge protocol which is nonmalleable in the sense described next.

Let (A', B') be any zero-knowledge protocol and let $(A, B) = \mathcal{C}(A', B')$. Let (C', D') be any (not necessarily zero-knowledge) protocol, and let $(C, D) = \mathcal{C}(C', D')$. Consider the adversarially coordinated system $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$. Note that, if (A, B) were to be run in isolation, then given the inputs (α, β) and the random tapes of $\psi(A)$ and $\psi(B)$, the conversation between these agents is completely determined. A similar statement applies to (C, D). For every polynomial-time relation approximator R and for every adversarially coordinated system of the compiled versions with adversary $\mathcal{A}$ there exists an adversary simulator $\mathcal{A}'$ satisfying the following requirement.

Let $\mathcal{D}$ now denote a distribution for inputs to all four players chosen by $\mathcal{A}$ consistent with the valid inputs for (A, B). Let $(\alpha, \beta, \gamma, \delta) \in_R \mathcal{D}$, and run the compiled versions of the two protocols. Let $\pi(\mathcal{A}, R)$ denote the probability that $R(\alpha, \beta, \gamma, \delta, \mathcal{D}, \mathcal{K}(C, D)) = 1$, where $\mathcal{K}(C, D)$ denotes the conversation between $\psi(C)$ and $\psi(D)$. The probability is over the coin-flips of $\mathcal{A}, \psi(A)$ and $\psi(D)$ and the choice of $(\alpha, \beta, \gamma, \delta)$ in $\mathcal{D}$. As above, R rejects if a conversation is syntactically incorrect.

Let $\mathcal{D}'$ (consistent with the legal input pairs for (A, B)) be chosen by $\mathcal{A}'$, and let $(\alpha, \beta, \gamma, \delta) \in_R \mathcal{D}'$. $\mathcal{A}'$ gets inputs β, γ. Run an execution of (C, D) in which $\mathcal{A}'$ controls $\psi(C)$, and let $\mathcal{K}'(C, D)$ denote the resulting conversation. Let $\pi(\mathcal{A}', R)$ denote the probability that $R(\alpha, \beta, \gamma, \delta, \mathcal{D}', \mathcal{K}'(C, D)) = 1$. The probability is over the coin-flips of $\mathcal{A}$ and $\psi(D)$ and the choice of $(\alpha, \beta, \gamma, \delta)$ in $\mathcal{D}'$.

The *nonmalleable zero-knowledge* security requirement is that for every polynomial-time bounded $\mathcal{A}$, there exists a polynomial-time bounded $\mathcal{A}'$ such that for every polynomial-time computable relation approximator R $|\pi(\mathcal{A}, R) - \pi'(\mathcal{A}', R)|$ is subpolynomial.

THEOREM 5.1. *There exists a compiler $\mathcal{C}$ that takes as inputs a 2-party protocol and outputs a compiled protocol. Let (A', B') be any zero-knowledge protocol and let $(A, B) = \mathcal{C}(A', B')$. Let (C', D') be any (not necessarily zero-knowledge) protocol, and let $(C, D) = \mathcal{C}(C', D')$. Then the adversarially coordinated system $\langle (A, B), (C, D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$ is nonmalleable zero-knowledge secure.*

Proof. Our compiler is conceptually extremely simple: A and B commit to their inputs and random tapes and then execute the protocol (A', B'), at each step proving that the messages sent are consistent with the committed values. We have to make sure that these zero-knowledge proofs of consistency do not interfere with the original protocol. The goal of the preprocessing phases is to make all the players' actions in the rest of the protocol predetermined. We now describe the action of the compiler on (A', B') in more detail.

Preprocessing phase I. Initially A and B choose a random string R_A as follows. A nonmalleably commits to a string σ_A using a sequence of nonmalleable bit commitments. B then sends a random string σ_B. The string R_A, not yet known to B, is the bitwise exclusive-or of σ_A and σ_B. A and B then choose a random string R_B in the same manner, but with the roles reversed, so that B knows R_B while A does not yet know it.

Preprocessing phase II. Each player performs a sequence of pairs of nonmalleable bit commitments. Each pair contains a commitment to zero and a commitment to one, in random order.

Preprocessing phase III. Each player commits to its input and to the seed of a cryptographically strong pseudorandom bit generator, using the nonmalleable scheme for string commitment described in section 4. The pseudorandom sequence is used instead of a truly random sequence whenever the original protocol calls for a random bit. Note in particular that A and B *both* begin with a nonmalleable commitment to their inputs and random tapes—this is critical.

Executing the original protocol. The parties execute the original protocol (with the pseudorandom bits), with each player proving at each step that the message it sends at that step is the one it should have sent in the unique conversation determined by its committed input and random tape, and the messages of the original protocol received so far. The commitments performed as part of the proofs of consistency are selected from the list of pairs of commitments generated in preprocessing step II. Since proving the consistency of the new message with the conversation so far can be done effectively (given the random tape and the input), this has a (malleable) zero-knowledge proof [40] in which the verifier only sends random bits. These random bits are taken from R_A and R_B. In particular, R_A is used as the random bits when B proves something to A: A, acting as verifier and knowing R_A, reveals the bits of R_A to B as they are needed by opening the necessary commitments from preprocessing phase I. The analogous steps are made when A proves consistency to B.

Before sketching the proof, we give some intuition for why we included preprocessing phases I and II. (While it is possible that these extra preprocessing steps are not needed, we do not see a complete proof without them.) First, note that the compiler uses a specific nonmalleable string commitment scheme (the one from section 4), rather than *any* such protocol. We used this protocol because of its extraction properties (which we use for proving nonmalleability). However, as we saw in section 4 in order to do the extraction in an adversarially coordinated system $\langle (A,B),(C,D),\ \mathcal{A} : \psi(B) \leftrightarrow \psi(C) \rangle$ in which $(A,B) = (C,D) = \mathcal{S}$, we needed to define $\mathcal{S}'$, a relaxed version of $\mathcal{S}$, and construct an $\mathcal{S}'$-adaptor-$\mathcal{S}$ system. We do not know how to construct "relaxed versions" of *arbitrary* protocols (A', B'). Since the compiled protocol (A,B) has a very special form, the construction of its relaxation is straightforward.

We now sketch the proof that the compiled protocol satisfies the requirements of the theorem. A's proofs of consistency are zero-knowledge since they use the random bits in R_B and in the simulation of this part of the interaction R_B can be extracted. A's proofs are sound since its bit commitments performed in preprocessing phase II are independent of R_B (since all the commitments are nonmalleable, and in particular, involve proofs of knowledge).

Since A and B commit in preprocessing phase III to their random tapes and values, the parts of the compiled communication that correspond to messages in (A', B') are completely determined before the execution corresponding to the (A', B') interaction is carried out.

Note that the three stage protocol described above remains zero-knowledge. This is true, since under the appropriate definition [41], the sequential composition of zero-knowledge protocols is itself zero-knowledge. So in particular, the (A,B) interaction is zero-knowledge.

Nonmalleable zero-knowledge security is proved as follows. We first note that the commitment of its input and random tape that A makes to B in preprocessing phase III remains nonmalleable despite the proofs of consistency during the execution of the original protocol. We then construct an extractor for the committed value in (C, D) in a fashion similar to the one constructed in section 4. To do this, we construct a "relaxed" zero-knowledge protocol analogous to $\mathcal{S}'$, based on (A, B). We apply Lemma 4.6 to show that the probability of extraction is similar to the probability that $\mathcal{A}$ succeeds (in the compiled (C, D) protocol). The key point is that an exposed triple remains exposed despite the presence of the proofs of consistency because the queries in the proofs of consistency have been predetermined in preprocessing phase I.

As in Lemma 4.8, extraction violates the zero-knowledge nature of (the relaxed) (A, B). □

6. Concluding remarks and future work. There are several interesting problems that remain to be addressed:

1. The issue of preserving the nonmalleability of compiled programs (as in section 5) under concurrent composition is challenging, as, unlike the cases of encryption and string commitment, in general zero-knowledge proofs are not known to remain zero-knowledge under concurrent composition (see, e.g., [35, 38]). On the other hand, there are various techniques for changing zero-knowledge protocols so that they become parallelizable, such as witness indistinguishability [30] and perfect commitments (see Chapter 6.9 in [35]). These techniques do not necessarily yield protocols that can be executed concurrently while preserving zero-knowledge.
2. All our nonmalleability results are for protocols that are in some sense zero-knowledge. Extend the definition of nonmalleability to interactions that are not necessarily zero-knowledge, such as general multiparty computation, and construct nonmalleable protocols for these problems.
3. Simplify the constructions in this paper. Bellare and Rogaway present simplified constructions using a random oracle [6, 7]. A challenging open problem is to (define and) construct a *publicly computable pseudorandom function*. Such a construction is essential if [6, 7] are to be made complexity-based. For a recent discussion on constructing such functions see [17, 18, 19]; note that none of the proposals there is sufficient to yield nonmalleability.

   Very recently Cramer and Shoup [23] suggested an efficient construction of a nonmalleable cryptosystem secure against CCA-post. The scheme is based on the decisional Diffie–Hellman assumption (see [57] for a discussion of the assumption) and requires only a *few* modular exponentiations for encryption and decryption.

   Recently, Di Crescenzo, Ishai, and Ostrovsky [26] showed that in a model in which there is a common random string shared by all parties, it is possible to obtain a noninteractive weaker variant of nonmalleable commitments. Recall that, informally, in our definition of nonmalleable commitment, the adversary succeeds if it commits to a "related" value. Our definition therefore does not require the adversary to actually *open* its commitment in order to succeed. In the [26] scheme, an adversary that commits to a related value but never opens the commitment is *not* considered to have succeeded.
4. Another recent development related to malleability in encryption is the work of Bleichenbacher [11] who showed how the ability to maul ciphertexts in the PKCS # 1 standard allows for a CCA-post. The interesting fact about this

attack is that the only type of feedback the attacker requires is whether a given string represents a valid ciphertext. This demonstrates yet again the significance of using a *provable* nonmalleable cryptosystem.

5. A recent result that utilizes nonmalleability in an interesting way is [4] which explores the issue of reducing an adversary's success probability via parallel repetition. The authors give an example of a protocol where the fact that the upper bound on the adversary's probability of success is 1/2 is due to the *nonmalleability* of a cryptosystem used, while the repeated protocol fails to reduce the error due to the *malleability* of the protocol itself.
6. *The selective decryption problem:* a type of chosen ciphertext attack *not* addressed in this paper is when the adversary is given the random bits used to generate the ciphertext (in addition to the plaintext). The following problem, phrased here in terms of a CD-ROM, is a concrete instance in which this kind of attack is relevant (the version presented here is due to [59] and is a variant of a problem posed by O. Goldreich): A CD-ROM is generated containing the encryptions of 100 images (generally, n images). A user, having a copy of the CD-ROM, chooses any subset, say of size 50, of the images, and purchases the decryption information for the selected images. Suggest an encryption scheme for this problem such that, *assuming the decryption information is significantly shorter* than the combined plaintexts of the purchased images, the remaining encryptions remain secure once the decryption information for the purchased images is known. Suppose we start with a semantically secure cryptosystem and encrypt each image with its own key. Then, if the decryption information is the collection of keys for the selected images, it is easy to show that an adversary can't, for any given undecrypted image P_i produce an I related to P_i. The challenge is to show that no adversary can find an I related to, say, all the remaining P_i's. For example, show that the adversary can't find the bitwise logical-OR of the remaining pictures.
   This type of problem is simply ignored in papers on generating session keys (see, e.g., [8, 9]). If session keys are to be used for encryption, then the selective decryption problem must be addressed.
7. Design a *completely malleable* cryptosystem in which, given $E(x)$ and $E(y)$ it is possible to compute $E(x + y)$, $E(xy)$, and $E(\bar{x})$, where $\bar{x}$ denotes the bitwise complement of x. Such a cryptosystem has application to secure two-party computation. For example, to compute $f(x, y)$ player A generates a completely malleable E/D pair and sends $(E(x), E)$ to player B. Player B, knowing y and a circuit for f, can return $E(f(x, y))$.
   Alternatively, prove the *nonmalleability conjecture*: if a cryptosystem is completely malleable, then it is insecure. A related statement holds for discrete logarithms modulo p, and in general for the *black-box field problem*. See the elegant papers of Maurer [53] and Boneh and Lipton [15].

Acknowledgments. Discussions with Moti Yung on achieving independence started this research. Advice and criticism from Russell Impagliazzo, Charlie Rackoff, and Dan Simon were critical in the formation of the paper. We thank Ran Canetti, Uri Feige, Marc Fischlin, Roger Fischlin, Oded Goldreich, Omer Reingold, and Adi Shamir for valuable discussions.

REFERENCES

[1] M. Ajtai and C. Dwork, *A public-key cryptosystem with worst-case/average-case equivalence*, in Proceedings 29th Annual ACM Symposium on the Theory of Computing, El Paso, TX, 1997, pp. 284–293.

[2] W. Alexi, B. Chor, O. Goldreich, and C. Schnorr, *RSA and Rabin functions: Certain parts are as hard as the whole*, SIAM J. Comput., 17 (1988), pp. 194–209.

[3] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, *Relations among notions of security for public-key encryption schemes*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 26–45.

[4] M. Bellare, R. Impagliazzo, and M. Naor, *Does parallel repetition lower the error in computationally sound protocols?*, in Proceedings 38th Annual IEEE Symposium on Foundations of Computer Science, Miami Beach, FL, 1997, pp. 374–383.

[5] M. Bellare and S. Micali, *How to sign given any trapdoor function*, J. ACM, 39 (1992), pp. 214–233.

[6] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, in Proceedings 1st ACM Conference on Computer and Communications Security, Fairfax, VA, 1993, pp. 62–73.

[7] M. Bellare and P. Rogaway, *Optimal asymmetric encryption—How to encrypt with RSA*, in Advances in Cryptology—Eurocrypt '94, Lecture Notes in Comput. Sci. 950, Springer-Verlag, New York, 1994, pp. 92–111.

[8] M. Bellare and P. Rogaway, *Entity authentication and key distribution*, in Advances in Cryptology—Crypto '93, Lecture Notes in Comput. Sci. 773, Springer-Verlag, New York, 1994, pp. 232–249.

[9] M. Bellare and P. Rogaway, *Provably secure session key distribution: The three party case*, in Proceedings 27th Annual ACM Symposium on the Theory of Computing, Las Vegas, NV, 1995, pp. 57–66.

[10] M. Blaze, J. Feigenbaum, and M. Naor, *A formal treatment of remotely keyed encryption*, in Advances in Cryptology—Eurocrypt '98, Lecture Notes in Comput. Sci. 1403, Springer-Verlag, New York, 1998, pp. 251–265.

[11] D. Bleichenbacher, *Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 1–12.

[12] M. Blum, P. Feldman, and S. Micali, *Noninteractive zero-knowledge proof systems*, in Proceedings 20th ACM Symposium on the Theory of Computing, Chicago, IL, 1988, pp. 103–112.

[13] M. Blum, A. De Santis, S. Micali, and G. Persiano, *Noninteractive zero-knowledge*, SIAM J. Comput., 20 (1991), pp. 1084–1118.

[14] M. Blum and S. Goldwasser, *An efficient probabilistic public-key encryption that hides all partial information*, in Advances in Cryptology—Crypto '84, Lecture Notes in Comput. Sci. 196, Springer-Verlag, New York, 1985, pp. 289–299.

[15] D. Boneh and R. Lipton, *Algorithms for black-box fields and their application to cryptography*, in Advances in Cryptology—Crypto '96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, New York, 1996, pp. 283–297.

[16] M. Burrows, M. Abadi, and R. Needham, *A logic of authentication*, ACM Trans. Comput. Systems, 8 (1990), pp. 18–36.

[17] R. Canetti, *Towards realizing random oracles: Hash functions that hide all partial information*, in Advances in Cryptology—Crypto '97, Lecture Notes in Comput. Sci. 1294, Springer-Verlag, New York, 1997, pp. 455–469.

[18] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology*, in Proceedings 30th Annual ACM Symposium on the Theory of Computing, Dallas, TX, 1998, pp. 209–218.

[19] R. Canetti, D. Micciancio, and O. Reingold, *Perfectly one-way probabilistic hashing*, in Proceedings 30th Annual ACM Symposium on the Theory of Computing, Dallas, TX, 1998, pp. 131–140.

[20] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch, *Verifiable secret sharing in the presence of faults*, in Proceedings 26th IEEE Symposium on Foundations of Computer Science, Portland, OR, 1985, pp. 383–395.

[21] B. Chor and M. Rabin, *Achieving independence in logarithmic number of rounds*, in Proceedings 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, 1987, pp. 260–268.

[22] R. Cramer and I. Damgard, *New generation of secure and practical RSA-based signatures*, in

Advances in Cryptology—Crypto '96, Lecture Notes in Comput. Sci. 1109, Springer-Verlag, New York, 1996, pp. 137–185.

[23] R. Cramer and V. Shoup, *A practical public key cryptosystem provable secure against adaptive chosen ciphertext attack*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 13–25.

[24] Y. Desmet, C. Goutier, and S. Bengio, *Special uses and abuses of the Fiat–Shamir passport protocol*, in Advances in Cryptology—Crypto '87, Lecture Notes in Comput. Sci. 293, Springer-Verlag, New York, 1988, pp. 21–39.

[25] A. De Santis and G. Persiano, *Noninteractive zero-knowledge proof of knowledge*, in Proceedings 33rd IEEE Symposium on the Foundation of Computer Science, Pittsburgh, PA, 1992, pp. 427–436.

[26] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky, *Noninteractive and nonmalleable commitment*, in Proceedings 30th Annual ACM Symposium on the Theory of Computing, Dallas, TX, 1998, pp. 141–150.

[27] C. Dwork and M. Naor, *An efficient existentially unforgeable signature scheme and its applications*, J. Cryptology, 11 (1998), pp. 187–208.

[28] C. Dwork and M. Naor, *Method for Message Authentication from Nonmalleable Crypto Systems*, US Patent 05539826, issued August 29, 1996.

[29] C. Dwork, M. Naor, and A. Sahai, *Concurrent zero-knowledge*, in Proceedings 30th Annual ACM Symposium on the Theory of Computing, Dallas, TX, 1998, pp. 409–418.

[30] U. Feige and A. Shamir, *Witness hiding and witness indistinguishability*, in Proceedings 22nd Annual ACM Symposium on the Theory of Computing, Baltimore, MD, 1990, pp. 416–426.

[31] U. Feige, A. Fiat, and A. Shamir, *Zero knowledge proofs of identity*, J. Cryptology, 1 (1988), pp. 77–94.

[32] U. Feige, D. Lapidot, and A. Shamir, *Multiple noninteractive zero-knowledge proofs based on a single random string*, in Proceedings 31st IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 308–317.

[33] M. J. Fischer and N. A. Lynch, *A lower bound for the time to assure interactive consistency*, Inform. Process Lett., 14 (1982), pp. 183–186.

[34] Z. Galil, S. Haber, and M. Yung, *Interactive public-key cryptosystems*, *symmetric public-key encryption*, in Advances in Cryptology—Crypto '85, Lecture Notes in Comput. Sci. 218, Springer-Verlag, New York, 1986, pp. 128–137.

[35] O. Goldreich, *Foundations of Cryptography*, 1995, also available online from http://www.eccc.uni-trier.de/eccc/info/ECCC-Books/eccc-books.html (Electronic Colloquium on Computational Complexity).

[36] O. Goldreich, *A Uniform Complexity Encryption of Zero-Knowledge*, Technion CS-TR 570, June 1989.

[37] O. Goldreich, S. Goldwasser, and S. Micali, *How to construct random functions*, J. ACM, 33 (1986), pp. 792–807.

[38] O. Goldreich and H. Krawczyk, *On the composition of zero-knowledge proof systems*, SIAM J. Comput., 25 (1996), pp. 169–192.

[39] O. Goldreich and L. Levin, *A hard predicate for all one-way functions*, in Proceedings 21st Annual ACM Symposium on the Theory of Computing, Seattle, WA, 1989, pp. 25–32.

[40] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity, and a methodology of cryptographic protocol design*, J. ACM, 38 (1991), pp. 691–729.

[41] O. Goldreich and Y. Oren, *Definitions and properties of zero-knowledge proof systems*, J. Cryptology, 6 (1993), pp. 1–32.

[42] O. Goldreich and E. Petrank, *Quantifying knowledge complexity*, in Proceedings 32nd IEEE Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1991, pp. 59–68.

[43] S. Goldwasser and S. Micali, *Probabilistic encryption*, J. Comput. System Sci., 28 (1984), pp. 270–299.

[44] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof-systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[45] S. Goldwasser, S. Micali, and R. Rivest, *A digital signature scheme secure against adaptive chosen-message attacks*, SIAM J. Comput., 17 (1988), pp. 281–308.

[46] S. Goldwasser, S. Micali, and P. Tong, *Why and how to establish a private code on a public network*, in Proceedings 23rd IEEE Symposium on the Foundation of Computer Science, Chicago, IL, 1982, pp. 134–144.

[47] R. Impagliazzo and M. Luby, *One-way functions are essential to computational based cryptography*, in Proceedings 30th IEEE Symposium on the Foundation of Computer Science, Research Triangle Park, NC, 1989, pp. 230–235.

[48] R. Impagliazzo, L. Levin, and M. Luby, *Pseudo-random generation from one-way functions*,

in Proceedings 21st ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 12–24.

[49] J. Kilian, *On the complexity of bounded-interaction and non-interactive zero-knowledge proofs*, in Proceedings 35th IEEE Symposium on the Foundation of Computer Science, Santa Fe, NM, 1994, pp. 466–477.

[50] J. Kilian and E. Petrank, *An efficient non-interactive zero-knowledge proof system for NP with general assumptions*, J. Cryptology, 11 (1998), pp. 1–27.

[51] J. Kilian, E. Petrank, and C. Rackoff, *Lower bounds for zero knowledge on the Internet*, in Proceedings 39th IEEE Symposium on the Foundation of Computer Science, Palo Alto, CA, 1998, pp. 484–492.

[52] M. Luby, *Pseudo-Randomness and Applications*, Princeton University Press, Princeton, NJ, 1996.

[53] U. Maurer, *Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms*, in Advances in Cryptology—Crypto '94, Lecture Notes in Comput. Sci. 839, Springer-Verlag, New York, 1994, pp. 271–281.

[54] S. Micali, C. Rackoff, and R. Sloan, *The notion of security for probabilistic cryptosystems*, SIAM J. Comput., 17 (1988), pp. 412–426.

[55] M. Naor, *Bit commitment using pseudo-randomness*, J. Cryptology, 4 (1991), pp. 151–158.

[56] M. Naor and O. Reingold, *Synthesizers and their application to the parallel construction of pseudo-random functions*, in Proceedings 36th IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 170–181.

[57] M. Naor and O. Reingold, *Number-theoretic constructions of efficient pseudo-random functions*, in Proceedings 38th IEEE Symposium on the Foundation of Computer Science, Miami Beach, FL, 1997, pp. 458–467.

[58] M. Naor and O. Reingold, *From unpredictability to indistinguishability: A simple construction of pseudo-random functions from MACs*, in Advances in Cryptology—Crypto '98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 267–282.

[59] M. Naor and A. Wool, *Access control and signatures via quorum secret sharing*, in Proceedings 3rd ACM Conference on Computer and Communications Security, New Delhi, India, 1996, pp. 157–168.

[60] M. Naor and M. Yung, *Universal one-way hash functions and their cryptographic applications*, in Proceedings 21st Annual ACM Symposium on the Theory of Computing, Seattle, WA, 1989, pp. 33–43.

[61] M. Naor and M. Yung, *Public-key cryptosystems provably secure against chosen ciphertext attacks* in Proceedings 22nd Annual ACM Symposium on the Theory of Computing, Baltimore, MD, 1990, pp. 427–437.

[62] M. O. Rabin, *Randomized Byzantine generals*, in Proceedings 24th IEEE Symposium on the Foundation of Computer Science, Tucson, AZ, 1983, pp. 403–409.

[63] C. Rackoff and D. Simon, *Noninteractive zero-knowledge proof of knowledge and chosen ciphertext attack*, in Advances in Cryptology—Crypto '91, Lecture Notes in Comput. Sci. 576, Springer Verlag, New York, 1992, pp. 433–444.

[64] R. Rivest, A. Shamir, and L. Adleman, *A method for obtaining digital signature and public key cryptosystems*, Comm. ACM, 21 (1978), pp. 120–126.

[65] J. Rompel, *One-way functions are necessary and sufficient for signatures*, in Proceedings 22nd Annual ACM Symposium on the Theory of Computing, Baltimore, MD, 1990, pp. 387–394.

[66] A. C. Yao, *Theory and applications of trapdoor functions*, in Proceedings 23rd IEEE Symposium on the Foundation of Computer Science, Chicago, IL, 1982, pp. 80–91.

[67] M. Yung, *Cryptoprotocols: Subscription to a public key, the secret blocking and the multiplayer mental poker game*, in Advances in Cryptology—Crypto '84, Lecture Notes in Comput. Sci. 196, Springer-Verlag, New York, 1985, pp. 439–453.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 438–456

TIME AND SPACE LOWER BOUNDS FOR NONBLOCKING IMPLEMENTATIONS*

PRASAD JAYANTI†, KING TAN†, AND SAM TOUEG‡

Abstract. We show the following time and space complexity lower bounds. Let $\mathcal{I}$ be any randomized nonblocking n-process implementation of any object in set A from any combination of objects in set B, where $A = \{$increment, fetch&add, modulo k counter (for any $k \geq 2n$), LL/SC bit, k-valued compare&swap (for any $k \geq n$), single-writer snapshot$\}$, and $B = \{$resettable consensus$\}$ $\cup$ $\{$historyless objects such as registers and swap registers$\}$. The space complexity of $\mathcal{I}$ is at least $n-1$. Moreover, if $\mathcal{I}$ is deterministic, both its time and space complexity are at least $n-1$. These lower bounds hold even if objects used in the implementation are of unbounded size.

This improves on some of the $\Omega(\sqrt{n})$ space complexity lower bounds of Fich, Herlihy, and Shavit [*Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing*, Ithaca, NY, 1993, pp. 241–249; *J. Assoc. Comput. Mach.*, 45 (1998), pp. 843–862]. It also shows the near optimality of some known wait-free implementations in terms of space complexity.

Key words. asynchronous shared memory algorithms, nonblocking, wait-free, synchronization, randomized shared object implementations, space complexity, time complexity, lower bounds

AMS subject classifications. 68Q17, 68W15

PII. S0097539797317299

1. Introduction.

Nonblocking and wait-free implementations of shared objects have been the subject of much research. While there have been several results on when such implementations are feasible and when they are not, results establishing their intrinsic time and space requirements are relatively scarce, especially for randomized implementations. In this paper, we present a technique by which one can obtain a linear lower bound on the space complexity of several randomized nonblocking implementations. The technique also yields a linear lower bound on the time complexity of several deterministic nonblocking implementations.

Specifically, our results are as follows. Let $\mathcal{I}$ be any randomized nonblocking n-process implementation of any object in set A from any combination of objects in set B, where $A = \{$increment, fetch&add, modulo k counter (for any $k \geq 2n$), LL/SC bit, k-valued compare&swap (for any $k \geq n$), single-writer snapshot$\}$, and $B = \{$resettable consensus$\} \cup \{$historyless objects$\}$. (Roughly speaking, an object is historyless if each of its operations either does not affect the state of the object or overwrites the previously applied operations. Examples include registers, test and set objects, and swap registers.) The space complexity of $\mathcal{I}$ is at least $n-1$. Moreover, if $\mathcal{I}$ is deterministic, both its time and space complexity are at least $n-1$. These lower bounds hold even if objects used in the implementation are of unbounded size.

Some of the results in this paper improve known lower bounds, while others are completely new. In particular, Fich, Herlihy, and Shavit proved a $\Omega(\sqrt{n})$ space complexity lower bound for a randomized nonblocking n-process implementation of

*Received by the editors February 28, 1997; accepted for publication (in revised form) September 22, 1999; published electronically June 3, 2000. This work was partially supported by NSF grants CCR-9402894, CCR-9711403, and CCR-9410421, DARPA/NASA Ames grant NAG-2-593, and a Dartmouth College Startup grant.

http://www.siam.org/journals/sicomp/30-2/31729.html

†Sudikoff Laboratory for Computer Science, Dartmouth College, Hanover, NH 03755 (prasad@cs.dartmouth.edu, kytan@cs.dartmouth.edu).

‡Department of Computer Science, Cornell University, Upson Hall, Ithaca, NY 14853 (sam@cs.cornell.edu).

binary consensus from historyless objects [FHS93, FHS98]. Using this result, they showed that any randomized nonblocking n-process implementation of *compare&swap*, or *fetch&add*, or *bounded-counter* from historyless objects requires $\Omega(\sqrt{n})$ instances of such objects. Our results on compare&swap, fetch&add, and bounded-counter are stronger in two ways: (i) we show that at least $n-1$ objects are necessary, and (ii) we show that $n-1$ objects are needed even if the implementation is free to use *resettable consensus objects*, besides the historyless objects allowed by [FHS93, FHS98]. On the other hand, our lower bound technique applies only to implementations of "multiple-use objects" in which each process can access the implemented object many times. In contrast, the technique of Fich, Herlihy, and Shavit applies even to implementations of "single-use objects" (that is, all decision problems). Thus, our proof technique does not (and cannot) improve upon the main result of Fich, Herlihy, and Shavit, namely, their $\Omega(\sqrt{n})$ space complexity lower bound for a randomized nonblocking n-process implementation of binary consensus.

The results presented in this paper also imply that the following deterministic implementations in the literature are almost space-optimal.

1. Afek et al. give two wait-free implementations of a *single-writer snapshot object* consisting of n segments, each one written by a different process: one from *unbounded registers* and one from *bounded registers* [AAD$^+$93]. The one that uses unbounded registers is of space complexity n. We prove a lower bound of $n-1$.
2. Aspnes gives a wait-free implementation of an n-process *bounded-counter* from a single instance of a *single-writer snapshot object* [Asp90]. Combined with the above result of Afek et al., this implies that bounded-counter can be implemented from n unbounded registers. We prove that at least $n-1$ *registers* are necessary when the bounded-counter is a modulo k counter, where $k \geq 2n$.

In both cases above, the lower bound of $n-1$ is particularly appealing because it applies to even randomized nonblocking implementations while the upper bound of n holds for deterministic wait-free implementations.

In fact, the lower bounds proved in this paper (and the lower bounds in [FHS93, FHS98]) apply not just to nonblocking implementations, but also to any implementation satisfying a weaker progress condition called *solo-termination*, defined in [FHS98]. Roughly speaking, a deterministic implementation is solo-terminating if at every configuration C in a system execution the following holds for all processes p: if p runs alone from C, p's operation on the implemented object will eventually complete. For a randomized implementation to be considered solo-terminating we require that for all C and p, if p runs alone from C there is at least one sequence of outcomes for p's coin tosses that will enable p to complete its operation on the implemented object.

It is well known that a wait-free implementation is also nonblocking. It is clear that a nonblocking implementation is also solo-terminating. Thus, the lower bounds that we prove here for solo-terminating implementations apply also to nonblocking and wait-free implementations.

There is a large body of research on algorithms for synchronous parallel computers (such as the PRAM model, the mesh, perfect shuffle, and hypercube architectures) that has resulted in many algorithms whose time complexity is polylogarithmic in the number of processors. In contrast, for the asynchronous model of computing, wait-free algorithms of polylog complexity are rare ([Cha96, Aum97, CJT98] are some notable exceptions). In fact, our paper formally proves that for deterministic wait-free implementations of many common objects from certain base objects, there are

no algorithms of sublinear time complexity, let alone polylog complexity.

Our proof technique for the lower bounds is general in that we have successfully applied it to implementations of a variety of objects. The technique is also interesting because (i) it simultaneously yields a lower bound on time and space complexities of implementations, (ii) the lower bound on space complexity holds even for randomized implementations, and (iii) the lower bounds apply not just to nonblocking or wait-free implementations, but also to any implementation satisfying the weaker solo-termination progress condition.

The paper is organized as follows. We present an informal model in section 2. In section 3, we define the class of perturbable types and prove the lower bound for implementations of *any* perturbable type. In section 4, we show that many common types of objects, such as counter, compare&swap, LL/SC bit, single-writer snapshot, are perturbable. Section 5 elaborates on the consequences of the results in sections 3 and 4.

2. Informal model.

2.1. Definitions of type, historyless type, and resettable consensus. A *type* is a tuple (OP, RES, Q, δ), where OP is a set of operations, RES is a set of responses, Q is a set of states, and $\delta : Q \times OP \to Q \times RES$ is a function, known as the *sequential specification* of the type. Intuitively, if $\delta(\sigma, op) = (\sigma', res)$, it means the following: applying the operation op to an object of this type in state σ causes the object to move to state σ' and return the response res.

Let $op(\sigma)$ denote the first component of the tuple $\delta(\sigma, op)$. The following definitions are from Fich, Herlihy, and Shavit [FHS98]. An operation op is *trivial* if its application does not affect the state; that is, for all states σ, $op(\sigma) = \sigma$. Operation op' *overwrites* operation op if applying op and then op' results in the same state as simply applying op'; more precisely, for all states σ, $op'(op(\sigma)) = op'(\sigma)$. A type is *historyless* if all its nontrivial operations overwrite one another. The types `register`, `test and set`, and `swap register` are examples of historyless types.

PROPOSITION 2.1. *For a historyless type, the following statements are true:*

(1) *For all states σ, nontrivial operations op_k and finite sequences $op_{k-1} \cdots op_1$ of operations, $op_k(op_{k-1}(\cdots op_1(\sigma) \cdots)) = op_k(\sigma)$.*
(2) *For all states σ and finite sequences $op_k op_{k-1} \cdots op_1$ of trivial operations, $op_k(op_{k-1}(\cdots op_1(\sigma) \cdots)) = \sigma$.*

Proof. The proposition is proved by a simple induction on the length of the operation sequence. □

We now define the type `resettable consensus` as (OP, RES, Q, δ), where

- $OP = \{\text{read}, \text{reset}\} \cup \{\text{propose } v \mid v \in \mathcal{N}\}$, where $\mathcal{N}$ is the set of natural numbers;
- $RES = \mathcal{N} \cup \{ack\}$;
- $Q = \mathcal{N} \cup \{\perp\}$;
- δ, the transition function, is as follows:
  - For all $u \in Q$, $\delta(u, \text{read}) = (u, u)$;
  - For all $u \in Q$, $\delta(u, \text{reset}) = (\perp, ack)$;
  - $\delta(\perp, \text{propose } v) = (v, v)$ and, for all $u \in \mathcal{N}$, $\delta(u, \text{propose } v) = (u, u)$.

Resettable consensus was first defined by Herlihy [Her88, Her91], but included only the propose and reset operations. We added the read operation to make our lower bound result stronger. Our definition of resettable consensus is similar to the sticky bit defined by Plotkin [Plo89].

2.2. Implementation. A *randomized implementation* is specified by the following elements:

- The type and the initial state of the implemented object $\mathcal{O}$ (the initial state of $\mathcal{O}$ is a state of its type).
- A set of objects $O_1, \ldots, O_m$ from which $\mathcal{O}$ is implemented, their types, and their initial states.
- A set of processes $p_1, \ldots, p_n$ that may access $\mathcal{O}$.
- A set of *randomized access procedures* $\texttt{Apply}(p_i, op, \mathcal{O})$ for $p_i \in \{p_1, \ldots, p_n\}$ and $op \in OP$, where OP is the set of operations associated with the type of $\mathcal{O}$.

The access procedure $\texttt{Apply}(p_i, op, \mathcal{O})$ specifies how p_i should execute the operation op on $\mathcal{O}$ in terms of operations on $O_1, \ldots, O_m$. The value returned by the procedure is deemed to be the response from $\mathcal{O}$. We call $O_1, \ldots, O_m$ the *base objects* of the implementation. The *space complexity* of the implementation is m.

The definitions presented in the rest of this section are with respect to a system that consists of processes $p_1, \ldots, p_n$ and an implemented object $\mathcal{O}$ that $p_1, \ldots, p_n$ may access. We denote such a system by $(p_1, \ldots, p_n; \mathcal{O})$. Each p_i has a set of states and has a distinguished input variable *op-list*$_i$. This variable is initialized (by the user of the system) with any infinite sequence of operations $op_1, op_2, \ldots$, where each op_j is an operation supported by $\mathcal{O}$. The initial state of p_i is determined uniquely by the implementation as a function of the value assigned to *op-list*$_i$. Each p_i performs the following actions repeatedly forever: obtain the next operation op from *op-list*$_i$ and execute the access procedure $\texttt{Apply}(p_i, op, \mathcal{O})$ until it returns.[1]

A process p_i executes an access procedure $\texttt{Apply}(p_i, op, \mathcal{O})$ in *steps*. Each step consists of the following sequence of actions, all of which occur together atomically:

- p_i tosses a coin. Let *toss-outcome* $\in$ COINSPACE denote the outcome of this toss.
  We make the following assumptions: (i) coin tosses are independent, and (ii) COINSPACE is a nonempty countable set of all possible outcomes of a coin toss, and the probability of each outcome in the set is nonzero.
- *toss-outcome*, p_i's present state, and the operation op that p_i wants to apply to $\mathcal{O}$ uniquely determine an operation *oper* and a base object O_j that *oper* should be applied to. Accordingly, p_i applies *oper* to O_j.
- O_j changes state and returns a response. The new state of O_j and the response are uniquely determined by the sequential specification of O_j.
- The response from O_j, together with *toss-outcome* and p_i's present state, uniquely determine the new state of p_i. It is possible for the procedure $\texttt{Apply}(p_i, op, \mathcal{O})$ to terminate, returning some response. In this case, the new state of p_i reflects both the fact that the access procedure terminated and the response returned by the access procedure. Further, p_i's step enabled from this state corresponds to the first step of the access procedure $\texttt{Apply}(p_i, op', \mathcal{O})$, where op' is the earliest unconsumed operation in *op-list*$_i$.

A *configuration* of $(p_1, \ldots, p_n; \mathcal{O})$ is a tuple $(\sigma_1, \ldots, \sigma_n, rem_1, \ldots, rem_n, \tau_1, \ldots, \tau_m)$, where σ_i is a state of p_i, rem_i is a suffix of *op-list*$_i$ (and corresponds to the infinite sequence of operations that p_i is yet to initiate on $\mathcal{O}$), and τ_j is a state of base object O_j. Since the implementation specifies a unique initial state for each base object and

[1] Our model is more restrictive than a model in which the sequence of operations that each p_i applies on $\mathcal{O}$ is not fixed initially. Since our time and space lower bounds apply to the restrictive model, they clearly apply to the more general model as well.

a unique initial state for each process p_i as a function of the value assigned to *op-list*$_i$, it follows that the initial configuration is uniquely determined by an assignment of infinite sequences of operations to the input variables *op-list*$_i$ ($1 \leq i \leq n$). An *execution fragment from configuration* C_0 of $(p_1, \ldots, p_n; \mathcal{O})$ is a finite or infinite sequence $C_0, C_1, C_2, \ldots$ of configurations such that, for all $k \geq 0$, C_{k+1} is the configuration that results when some process performs a step in configuration C_k. An *execution* is an execution fragment from an initial configuration.

A *schedule* is a finite or an infinite sequence $[p_{i_1}, t_1], [p_{i_2}, t_2], \ldots$, where each p_{i_j} is from $\{p_1, \ldots, p_n\}$ and each t_j is from COINSPACE. If C is a configuration and $\alpha = [p_{i_1}, t_1], [p_{i_2}, t_2], \ldots$ is a schedule, EXEC(C, α) denotes the execution fragment $C_0, C_1, C_2, \ldots$, where $C = C_0$ and each C_k results from C_{k-1} when p_{i_k} takes a step in which the outcome of p_{i_k}'s toss is t_k. A configuration C is *reachable* if there is some initial configuration C_0 and a finite schedule α such that the configuration at the end of EXEC(C_0, α) is C.

An implementation is *correct* if it has two properties—linearizability (safety property) and solo-termination (liveness property), which are described next.

2.3. Linearizability. Let $\mathcal{O}$ be an implemented object shared by processes $p_1, \ldots, p_n$. Each completed operation on $\mathcal{O}$ in an execution of $(p_1, \ldots, p_n; \mathcal{O})$ can be viewed as a pair of events—the invocation of the operation followed by the response to the operation. Informally, an execution E of $(p_1, \ldots, p_n; \mathcal{O})$ is *linearizable* if every operation in E appears to take effect at some instant between its invocation and response (in other words, operations appear atomic). The *implementation of* $\mathcal{O}$ *is linearizable* if every execution of $(p_1, \ldots, p_n; \mathcal{O})$ is linearizable. For a precise definition of linearizability, we refer the reader to the work of Herlihy and Wing in which this concept was first formally introduced [HW90].

2.4. Solo-termination: A progress property. An implementation whose access procedures never terminate is trivially linearizable. Such an implementation, however, is hardly useful. Thus, in addition to linearizability, implementations should guarantee certain progress properties. *Wait-freedom* and *nonblockingness* are the progress conditions that received the most attention recently [HS93]. In this paper, we consider a weaker progress property called *solo-termination*, first defined in [FHS98]. Informally, an implementation has the solo-termination property if, for each reachable configuration C and each process p, the following holds: if p runs alone from configuration C, then there is at least one sequence of outcomes for p's coin tosses that will enable p to complete an operation on the implemented object. More precisely, a *randomized implementation of* $\mathcal{O}$ *is solo-terminating* if, for all reachable configurations C and all processes p_i, there is some finite schedule $\alpha = [p_i, t_1], [p_i, t_2], \ldots, [p_i, t_k]$ such that p_i completes an operation on $\mathcal{O}$ during EXEC(C, α). (Since the probability of each t_j is nonzero by definition, there is a nonzero probability of p_i completing an operation on $\mathcal{O}$ when p_i runs alone from C.)

The lower bounds proved in this paper apply to solo-terminating (and therefore to nonblocking and wait-free) implementations.

2.5. Notation. For a schedule α, $|\alpha|$ denotes its length. We say α *contains process* p if, for some t, $[p, t]$ is in the sequence α. PSET(α) denotes the set of all processes contained in α. If α and β are any schedules, $\alpha\beta$ denotes the schedule which is the concatenation of α and β.

If Σ is a set, Σ^* denotes the set of all *finite* sequences of elements from Σ (including the empty sequence, denoted by ϵ). Notice that $(\{p_1, \ldots, p_n\} \times \text{COINSPACE})^*$ is the set of all finite schedules.

In our proofs, when we consider a system $(p_1, \ldots, p_n; \mathcal{O})$ (where $\mathcal{O}$ is an object implemented for processes $p_1, \ldots, p_n$), we fix the initial configuration of the system at some value, say, C_0, right at the beginning of the proof by specifying the initial values of the input variables *op-list*$_i$ $(1 \leq i \leq n)$. Since the initial configuration is fixed, each schedule $\alpha \in (\{p_1, \ldots, p_n\} \times \text{COINSPACE})^*$ uniquely determines the execution $\text{EXEC}(C_0, \alpha)$. Therefore, for brevity, if α is a schedule, we will use the same symbol α also to denote the execution $\text{EXEC}(C_0, \alpha)$. From the context it will be clear whether α refers to the schedule or to the execution. If α and β are schedules and S is a set of processes or a set of base objects, then we write $\alpha \approx_S \beta$ if, for all $A \in S$, A is in the same state at the end of the executions α and β.

2.6. Definitions of "just completes" and deterministic time complexity. Let E be any execution fragment of $(p_1, \ldots, p_n; \mathcal{O})$, where $\mathcal{O}$ is an object implemented using a randomized implementation. We say process p_i *just completes* an operation on $\mathcal{O}$ in E if in its last step in E, p_i returns from an access procedure completing an operation on $\mathcal{O}$.

A *deterministic implementation* is a special case of a randomized implementation for which COINSPACE, the set of possible outcomes for a coin toss, is a singleton set. The *solo-termination time complexity of a deterministic implementation of* $\mathcal{O}$ is the maximum, over all reachable configurations C of $(p_1, \ldots, p_n; \mathcal{O})$ and all processes p_i, of $|\alpha|$ such that (i) α is a schedule that contains only p_i, and (ii) in $\text{EXEC}(C, \alpha)$, p_i completes exactly one operation on $\mathcal{O}$ and, in fact, just completes it.

We note that, for any reasonable definition of the time complexity of a nonblocking (or wait-free) implementation of $\mathcal{O}$, if the solo-termination time complexity of a deterministic implementation of $\mathcal{O}$ is at least k, then the time complexity of any deterministic nonblocking (or wait-free) implementation of $\mathcal{O}$ is also at least k.

3. The lower bound. This section has three parts. Section 3.1 illustrates the main ideas of our lower bound technique for the special case of implementing an increment object. Our lower bound applies not just to implementations of the increment object, but to implementations of a large class of objects, which we call perturbable objects. Section 3.2 defines the class of perturbable objects. Section 3.3 proves the lower bound for any implementation of any perturbable object.

3.1. The intuition. To illustrate the main ideas of the lower bound proof, we provide below a proof sketch for a simple case of the full result. Consider any deterministic implementation of an increment object $\mathcal{O}$,[2] shared by $p_1, \ldots, p_n$, from swap objects.[3] We will prove that the space and time complexity of the implementation are at least $n-1$. (The full result is more general in three ways: it applies to randomized implementations; it applies to implementations of any perturbable object, not just the increment object; and it applies even if base objects include resettable consensus objects.)

[2] An *increment object* supports *increment* and *read* operations. The *increment* operation adds 1 to the state and returns *ack*. The *read* operation returns the state without affecting it.

[3] A *swap object* supports *read* and *write* operations. The *write* v operation changes the state to v and returns the previous state. The *read* operation returns the state without modifying it.

Consider Scenario 0 in which p_n initiates a read operation on $\mathcal{O}$ and runs alone. We claim that p_n accesses some base object, before completing this read operation on $\mathcal{O}$. For a proof, suppose the claim is false. Then p_n cannot distinguish Scenario 0 from Scenario $0'$ in which some process completes an increment operation on $\mathcal{O}$ before p_n starts taking steps. Yet, correctness requires p_n's read operation to return different values in Scenarios 0 and $0'$. This contradiction implies the claim. Let B_1 denote the first base object that p_n accesses.

To force p_n to access a second base object, the idea is to schedule other processes, before scheduling p_n, in such a way that they render the information in B_1 not much useful for p_n. Consequently, later when p_n runs and accesses B_1, it will not learn enough to determine the response to its read operation on $\mathcal{O}$; thus, p_n is forced to access a second base object. The details are as follows. Let Scenario 1 depict an execution in which processes other than p_n take steps until some process, say, p_{i_1}, writes B_1.[4] If p_n runs after Scenario 1, it accesses B_1 because, until accessing B_1, this scenario is indistinguishable to p_n from Scenario 0. More significantly, we show below that, when running after Scenario 1, p_n accesses some base object, besides B_1, before completing its first read operation on $\mathcal{O}$. To see this, consider another scenario, Scenario $1'$, which is the same as Scenario 1 with the following exception: just before p_{i_1} writes B_1, some process p_l other than p_{i_1} and p_n completes many increment operations on $\mathcal{O}$, and after this p_{i_1} takes a step and writes B_1. Clearly, B_1's value is the same at the end of Scenarios 1 and $1'$. Now extend each of Scenarios 1 and $1'$ by letting p_n take steps. If p_n accesses *only* B_1, it is clear that the two scenarios would be indistinguishable to p_n. Yet, since many more increments operations are completed in Scenario $1'$ than in Scenario 1, p_n's read operation on $\mathcal{O}$ must return different values in the two scenarios. It follows that, in Scenario 1 (and in Scenario $1'$), p_n must access a second base object before completing its read operation on $\mathcal{O}$. Let B_2 denote this base object.

To force p_n to access a third base object, we repeat the above trick and render the information in B_2 not much useful for p_n. Specifically, consider Scenario 2 consisting of the following three execution fragments, taking place in this order: (i) the prefix of Scenario 1 (described above) up to, but excluding the write of B_1 by p_{i_1}, (ii) the steps of processes other than p_{i_1} and p_n until some process, say, p_{i_2}, writes B_2, and (iii) the write of B_1 by p_{i_1}. If p_n were to run after Scenario 2, it accesses B_1 and B_2 because, until accessing B_2, this scenario is indistinguishable to p_n from Scenario 1. We claim that p_n accesses some base object, besides B_1 and B_2, before completing its first read operation on $\mathcal{O}$. The justification is as in the previous paragraph: if the claim is false, p_n cannot distinguish Scenario 2 from Scenario $2'$, where Scenario $2'$ is similar to Scenario 2 except that some process p_l other than p_{i_1}, p_{i_2}, and p_n completes many increment operations on $\mathcal{O}$ just before the write steps of p_{i_2} and p_{i_1} on B_2 and B_1, respectively. This is a contradiction since p_n's read operation on $\mathcal{O}$ must return different values in Scenarios 2 and $2'$.

Repeating the above argument, we construct successively Scenarios $3, \ldots, n-2$ with the property that, if p_n runs alone after Scenario k, it accesses at least $k+1$ distinct base objects before completing its first read operation on $\mathcal{O}$. The lower bound of $n-1$ on the space and time complexity of the implementation is immediate from the existence of Scenario $n-2$. (We cannot proceed any further than Scenario $n-2$ because processes other than p_n are all already used up: they play the roles

[4] It is possible that such an execution does not exist. In order not to obscure the basic intuition, we address this possibility only in the formal proof, presented in section 3.3.

of $p_{i_1}, \ldots, p_{i_{n-2}}$ or as a process that does increment operations just before the write steps of $p_{i_1}, \ldots, p_{i_{n-2}}$.)

In summary, the crux is to ensure that p_n gets no useful information from $B_1, B_2, \ldots, B_{n-2}$; that is, $B_1, B_2, \ldots, B_{n-2}$ are rendered useless to p_n. This is accomplished by "using up" p_{i_j} to render B_j useless. This technique of using up one new process for each additional shared object to be rendered useless, in the context of space complexity lower bounds, was used earlier by Burns and Lynch [BL93].

We turn the above ideas into a rigorous inductive proof in section 3.3. To understand the correspondence between that proof and the above informal argument, note that Scenario 2 described above has three parts: a schedule involving $p_1, \ldots, p_{n-1}$, followed by the write steps of p_{i_2} and p_{i_1} (on objects B_2 and B_1, respectively), followed by the steps of only p_n. More generally, if we extended the argument to Scenarios $3, 4, \ldots$, Scenario k would consist of three parts, where the first part is a schedule involving $p_1, \ldots, p_{n-1}$, the second part is the write steps of $p_{i_k}, p_{i_{k-1}}, \ldots, p_{i_1}$ on some base objects $B_k, B_{k-1}, \ldots, B_1$, and the third part is the steps of only p_n. Roughly speaking, these three parts of Scenario k' correspond to the schedules Λ, Σ, and Π, respectively, in Definition 3.1 in section 3.2, and to the schedules Λ_k, Σ_k, and Π_k, respectively, of the proof in section 3.3. We caution the reader that this correspondence is not exact but is close enough to help the reader understand how the formal proof works.

3.2. Perturbable types. The key property of the increment object exploited in the above proof is the following: it is possible to create a new scenario by scheduling the steps of some process p_l immediately before that of $p_{i_k}, p_{i_{k-1}}, \ldots, p_{i_1}$ in such a way that p_n is forced to distinguish the new scenario from the older one. Below we state an abstract version of this property (which suffices for our proof technique to work), and call any type that has this property a *perturbable type*.

DEFINITION 3.1. Type T is perturbable for n processes, for initial state s *if for every linearizable and solo-terminating randomized implementation of an object $\mathcal{O}$ of type T, initialized to s and shared by processes $p_1, \ldots, p_n$, there exists an assignment of operation sequences to input variables $oplist_1, \ldots, oplist_n$ such that the following statement holds:*

If Λ, Σ, and Π are any schedules that satisfy the following four conditions,

- $\text{PSET}(\Lambda) \subseteq \{p_1, \ldots, p_{n-1}\}$;
- $\text{PSET}(\Sigma)$ *is a proper subset of* $\{p_1, \ldots, p_{n-1}\}$ *and each process appears at most once in* Σ;
- $\text{PSET}(\Pi) = \{p_n\}$;
- *in* $\Lambda\Sigma\Pi$, p_n*'s first operation on* $\mathcal{O}$ *just completes and returns some response res;*

then, for some $p_l \in \{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$, *there is a schedule* $\gamma \in (\{p_l\} \times \text{COINSPACE})^*$ *such that, in* $\Lambda\gamma\Sigma\Pi$, *either* p_n*'s first operation on* $\mathcal{O}$ *does not complete or it returns a response different from res.*

3.3. The main result. We prove that the space complexity of any randomized solo-terminating implementation and the time complexity of any deterministic solo-terminating implementation of any perturbable object, shared by n processes, are both at least $n-1$ if base objects are restricted to be (any combination of) resettable consensus objects and historyless objects, such as registers, test and set objects, and swap registers.

THEOREM 3.2. *Suppose that type T is perturbable for n processes for some initial state s. Consider any randomized implementation of an object of type T, initialized to*

There are schedules Λ_k, Σ_k, Π_k such that the following conditions hold:

(1) Λ_k, $\Sigma_k \in (\{p_1, p_2, \ldots, p_{n-1}\} \times \text{COINSPACE})^*$, and $\Pi_k \in (\{p_n\} \times \text{COINSPACE})^*$. That is, Λ_k and Σ_k do not contain p_n and Π_k contains no process other than p_n.
(2) $|\Sigma_k| = |\text{PSET}(\Sigma_k)| = k$. That is, k distinct processes take one step each in Σ_k.
(3) In $\Lambda_k\Sigma_k\Pi_k$, p_n accesses exactly k distinct base objects and p_n's first operation on $\mathcal{O}$ has either not completed or just completed.
(4) Let $\mathcal{S}_k$ be the set of base objects that p_n accesses in $\Lambda_k\Sigma_k\Pi_k$. Let $\mathcal{P}_k = \{p_1, p_2, \ldots, p_{n-1}\} - \text{PSET}(\Sigma_k)$ and γ be any schedule in $(\mathcal{P}_k \times \text{COINSPACE})^*$. Then, $\Lambda_k\Sigma_k \approx_{\mathcal{S}_k} \Lambda_k\gamma\Sigma_k$.

FIG. 3.1. *Statement S_k.*

s and shared by processes $p_1, \ldots, p_n$, from resettable consensus objects and historyless objects. If the implementation is linearizable and solo-terminating, then

(1) *its space complexity is at least $n - 1$;*
(2) *if the implementation is deterministic, its solo-termination time complexity is at least $n - 1$.*

Proof. Let $\mathcal{O}$ be the implemented object, and let C_0 be the initial configuration of $(p_1, \ldots, p_n; \mathcal{O})$ obtained by assigning to *op-list*$_1, \ldots,$ *op-list*$_n$ the operation sequences mentioned in Definition 3.1. The crux of the proof lies in the following claim: For all k, $0 \le k \le n-1$, Statement S_k, presented in Figure 3.1, is true. We prove this claim by induction. Below, we let $S_k : j$ denote the jth part of Statement S_k.

Base case. Let $\Lambda_0 = \Sigma_0 = \Pi_0 = \epsilon$ (ϵ is the empty sequence). It is easy to verify that all of $S_0 : 1$-4 are true. Hence, we have the base case.

Induction step. Suppose $0 \le k \le n-2$ and S_k is true. Let $\Lambda_k, \Sigma_k, \Pi_k$ be so defined as to make Statement S_k true. Let $\mathcal{S}_k$ denote the set of base objects that p_n accesses in $\Lambda_k\Sigma_k\Pi_k$, and let $\mathcal{P}_k$ denote $\{p_1, p_2, \ldots, p_{n-1}\} - \text{PSET}(\Sigma_k)$. We show that S_{k+1} is true through the following steps.

(1) By $S_k : 3$, in $\Lambda_k\Sigma_k\Pi_k$, p_n's first operation on $\mathcal{O}$ has either not completed or just completed. Let $\pi \in (\{p_n\} \times \text{COINSPACE})^*$ be such that, in $\Lambda_k\Sigma_k\Pi_k\pi$, p_n just completes its first operation on $\mathcal{O}$, returning some value. Since the implementation is solo-terminating, π exists.
(2) *Claim* 1. $\pi \neq \epsilon$ and in $\Lambda_k\Sigma_k\Pi_k\pi$, p_n accesses a base object not in $\mathcal{S}_k$.

Proof. Suppose the claim is false. Recall that $\mathcal{P}_k = \{p_1, p_2, \ldots, p_{n-1}\} - \text{PSET}(\Sigma_k)$. Since $|\text{PSET}(\Sigma_k)| = k$ and $k \le n-2$, $\mathcal{P}_k$ is nonempty. For all $p_l \in \mathcal{P}_k$ and $\gamma \in (\{p_l\} \times \text{COINSPACE})^*$, we assert that $\Lambda_k\Sigma_k\Pi_k\pi \approx_{p_n} \Lambda_k\gamma\Sigma_k\Pi_k\pi$. This assertion follows from the facts below: (i) $\Lambda_k\Sigma_k \approx_{p_n} \Lambda_k\gamma\Sigma_k$ (since the schedules $\Lambda_k\Sigma_k$ and $\Lambda_k\gamma\Sigma_k$ do not contain p_n), (ii) $\Lambda_k\Sigma_k \approx_{\mathcal{S}_k} \Lambda_k\gamma\Sigma_k$ (by $S_k : 4$), (iii) the schedule $\Pi_k\pi$ contains only p_n, and (iv) the only base objects accessed by p_n in $\Lambda_k\Sigma_k\Pi_k\pi$ are the ones in $\mathcal{S}_k$ (by our assumption that Claim 1 is false).

The above assertion, together with the definition of π from step 1, implies that p_n's first operation on $\mathcal{O}$ completes and returns the same response in $\Lambda_k\gamma\Sigma_k\Pi_k\pi$ as in $\Lambda_k\Sigma_k\Pi_k\pi$. This contradicts Definition 3.1. (To see the contradiction, substitute Λ, Σ, Π in the definition with $\Lambda_k, \Sigma_k, \Pi_k\pi$, respectively, and note that the conditions in the definition hold because of the induction hypothesis.) Hence, we have Claim 1. ∎

(3) DEFINITION 3.3. *Define* π_{k+1}, B_{k+1}, *and* Π_{k+1} *as follows*:

- π_{k+1} *is the shortest prefix of* π *such that, in* $\Lambda_k\Sigma_k\Pi_k\pi_{k+1}$, p_n *accesses a base object not in* $\mathcal{S}_k$ *(by Claim* 1, π_{k+1} *exists).*
- B_{k+1} *is the unique base object not in* $\mathcal{S}_k$ *that* p_n *accesses in* $\Lambda_k\Sigma_k\Pi_k\pi_{k+1}$.
- $\Pi_{k+1} = \Pi_k\pi_{k+1}$.

Claim 2. $\Pi_{k+1} \in (\{p_n\} \times \text{COINSPACE})^*$.

Proof. Since Π_k and π_{k+1} are both from $(\{p_n\} \times \text{COINSPACE})^*$, we have $\Pi_{k+1} \in (\{p_n\} \times \text{COINSPACE})^*$. □

(4) *Claim* 3. There exist $\lambda_{k+1} \in (\mathcal{P}_k \times \text{COINSPACE})^*$ and $[p_{i_{k+1}}, t_{k+1}] \in \mathcal{P}_k \times \text{COINSPACE}$ such that, for all $\gamma \in ((\mathcal{P}_k - \{p_{i_{k+1}}\}) \times \text{COINSPACE})^*$, $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}] \approx_{B_{k+1}} \Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$.

Proof. The following observation will be used many times in the proof:

Observation O1. For all $\gamma \in ((\mathcal{P}_k - \{p_{i_{k+1}}\}) \times \text{COINSPACE})^*$, process $p_{i_{k+1}}$ accesses the same base object and applies the same operation in the last step of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$ as in the last step of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$.

This observation follows from the fact that γ does not contain $p_{i_{k+1}}$. In the following, we pick λ_{k+1} and $[p_{i_{k+1}}, t_{k+1}]$ based on the type of B_{k+1}, and in each case prove that our choice of λ_{k+1} and $[p_{i_{k+1}}, t_{k+1}]$ satisfies Claim 3. In the rest of the proof, γ denotes an arbitrary schedule in $((\mathcal{P}_k - \{p_{i_{k+1}}\}) \times \text{COINSPACE})^*$.

Case 1. B_{k+1} is a historyless object.

Subcase 1a. There is some nonempty schedule $\lambda \in (\mathcal{P}_k \times \text{COINSPACE})^*$ such that the last step in $\Lambda_k\lambda$ is a nontrivial operation on B_{k+1}. Define λ_{k+1} and $[p_{i_{k+1}}, t_{k+1}]$ so that $\lambda = \lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$. By O1, $p_{i_{k+1}}$ performs the same nontrivial operation on B_{k+1} in the last step of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$ as in the last step of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$. This, together with Proposition 2.1, gives Claim 3.

Subcase 1b. There is no such λ.

Define λ_{k+1} to be ϵ and $[p_{i_{k+1}}, t_{k+1}]$ to be any element of $\mathcal{P}_k \times \text{COINSPACE}$. It follows from the subcase in consideration that no nontrivial operation is performed on B_{k+1} in the last $|\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]|$ steps of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$ and in the last $|\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]|$ steps of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. Therefore, by Proposition 2.1, $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}] \approx_{B_{k+1}} \Lambda_k \approx_{B_{k+1}} \Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. Hence, we have Claim 3.

Case 2. B_{k+1} is a resettable consensus object.

Subcase 2a. There is some nonempty schedule $\lambda \in (\mathcal{P}_k \times \text{COINSPACE})^*$ such that the last step in $\Lambda_k\lambda$ is a reset operation on B_{k+1}. Define λ_{k+1} and $[p_{i_{k+1}}, t_{k+1}]$ so that $\lambda = \lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$. By O1, $p_{i_{k+1}}$ performs a reset on B_{k+1} in the last step of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$, just as it does in the last step of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$. Hence, we have Claim 3.

Subcase 2b. There is no such λ. However, there is some nonempty schedule $\lambda' \in (\mathcal{P}_k \times \text{COINSPACE})^*$ such that the last step in $\Lambda_k\lambda'$ is a propose operation on B_{k+1}.

Define λ_{k+1} to be λ' and $[p_{i_{k+1}}, t_{k+1}]$ to be any element of $\mathcal{P}_k \times \text{COINSPACE}$. Let σ be the state of B_{k+1} at the end of $\Lambda_k\lambda_{k+1}$. Since Subcase 2a is not applicable, it follows that B_{k+1} is not reset in the last $|\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]|$ steps of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$ and in the last $|\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]|$ steps of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. Thus, B_{k+1}'s state is σ at the end of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$ and at the end of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. Hence, we have Claim 3.

Subcase 2c. Neither λ nor λ' exists.
Define λ_{k+1} to be ϵ and $[p_{i_{k+1}}, t_{k+1}]$ to be any element of $\mathcal{P}_k \times$ COINSPACE. It follows from the subcase under consideration that no reset or propose operation is performed on B_{k+1} in the last $|\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]|$ steps of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$ and in the last $|\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]|$ steps of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. Therefore, B_{k+1}'s state at the end of $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]$ is the same as its state at the end of $\Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. Hence, we have Claim 3.

This completes the proof of Claim 3. □

(5) DEFINITION 3.4. *Let λ_{k+1} and $[p_{i_{k+1}}, t_{k+1}]$ be as in Claim* 3. *Define Λ_{k+1} and Σ_{k+1} as follows:*
- $\Lambda_{k+1} = \Lambda_k\lambda_{k+1}$.
- $\Sigma_{k+1} = [p_{i_{k+1}}, t_{k+1}]\Sigma_k$.

Claim 4. Λ_{k+1}, $\Sigma_{k+1} \in (\{p_1, p_2, \ldots, p_{n-1}\} \times$ COINSPACE$)^*$.

Proof. By definition, $\lambda_{k+1} \in (\mathcal{P}_k \times$COINSPACE$)^*$ and $[p_{i_{k+1}}, t_{k+1}] \in \mathcal{P}_k \times$COINSPACE, where $\mathcal{P}_k = \{p_1, p_2, \ldots, p_{n-1}\} -$ PSET(Σ_k). This, together with $S_k : 1$, implies the claim. □

Claim 5. $|\Sigma_{k+1}| = |$PSET$(\Sigma_{k+1})| = k + 1$.

Proof. This claim follows from the definition of Σ_{k+1} as $[p_{i_{k+1}}, t_{k+1}]\Sigma_k$ and the following two facts: (i) $|\Sigma_k| = |$PSET$(\Sigma_k)| = k$ (by $S_k : 2$), and (ii)$[p_{i_{k+1}}, t_{k+1}] \in \mathcal{P}_k \times$ COINSPACE and $\mathcal{P}_k$ does not include any process from PSET(Σ_k). □

(6) *Claim* 6. Let $\mathcal{P}_{k+1} = \{p_1, p_2, \ldots, p_{n-1}\} -$ PSET(Σ_{k+1}). Let γ be any schedule from $(\mathcal{P}_{k+1} \times$ COINSPACE$)^*$. Then we have the following:
- (a) $\Lambda_{k+1}[p_{i_{k+1}}, t_{k+1}] \approx_{B_{k+1}} \Lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$.
- (b) $\Lambda_{k+1}\Sigma_{k+1} \approx_{B_{k+1}} \Lambda_{k+1}\gamma\Sigma_{k+1}$.
- (c) $\Lambda_{k+1}\Sigma_{k+1} \approx_{\mathcal{S}_k} \Lambda_k\Sigma_k \approx_{\mathcal{S}_k} \Lambda_{k+1}\gamma\Sigma_{k+1}$.

Proof. Part (a) of this claim is a rephrasing of Claim 3. The proof of the other two parts of this claim will use Observation O1, stated earlier in the proof of Claim 3.

By construction, $p_{i_{k+1}} \notin$ PSET(Σ_k). By definition of γ, PSET$(\gamma) \cap$ PSET$(\Sigma_k) = \emptyset$. It follows that $\Lambda_{k+1}[p_{i_{k+1}}, t_{k+1}] \approx_{\text{PSET}(\Sigma_k)} \Lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]$. From this, the fact that each process in PSET(Σ_k) appears only once in Σ_k and $|\Sigma_k| = k$, we conclude that the sequence of base objects accessed and the operations applied in the last k steps of $\Lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]\Sigma_k$ are identical to the sequence of base objects accessed and the operations applied in the last k steps of $\Lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]\Sigma_k$. This, together with part (a) of the claim, implies part (b).

It follows from the induction hypothesis $S_k : 4$ that $\Lambda_k\lambda_{k+1}[p_{i_{k+1}}, t_{k+1}]\Sigma_k \approx_{\mathcal{S}_k} \Lambda_k\Sigma_k \approx_{\mathcal{S}_k} \Lambda_k\lambda_{k+1}\gamma[p_{i_{k+1}}, t_{k+1}]\Sigma_k$. This, together with prior definitions of Λ_{k+1} as $\Lambda_k\lambda_{k+1}$ and Σ_{k+1} as $[p_{i_{k+1}}, t_{k+1}]\Sigma_k$, gives part (c) of the claim. □

(7) *Claim* 7.
- (a) Let $\mathcal{S}_{k+1}$ be the set of base objects that p_n accesses in $\Lambda_{k+1}\Sigma_{k+1}\Pi_{k+1}$. Then $\mathcal{S}_{k+1} = \mathcal{S}_k \cup \{B_{k+1}\}$ and $|\mathcal{S}_{k+1}| = k + 1$.
- (b) In $\Lambda_{k+1}\Sigma_{k+1}\Pi_{k+1}$, p_n's first operation on $\mathcal{O}$ has either not completed or just completed.

Proof. We make the following observations: (1) $\Lambda_k\Sigma_k \approx_{p_n} \Lambda_{k+1}\Sigma_{k+1}$ (since neither $\Lambda_k\Sigma_k$ nor $\Lambda_{k+1}\Sigma_{k+1}$ contains p_n); (2) $\Lambda_k\Sigma_k \approx_{\mathcal{S}_k} \Lambda_{k+1}\Sigma_{k+1}$ (this is part (c) of Claim 6); (3) by definition of $\mathcal{S}_k$, $\mathcal{S}_k$ is exactly the set of base objects that p_n accesses in $\Lambda_k\Sigma_k\Pi_k$; and (4) by definition of π_{k+1}, in $\Lambda_k\Sigma_k\Pi_k\pi_{k+1}$, it is only in the last step that p_n accesses a base object not in $\mathcal{S}_k$ (this base object is B_{k+1}), and p_n's first

operation on $\mathcal{O}$ has either not completed or just completed. These observations imply part (b) of the claim and that $\mathcal{S}_{k+1} = \mathcal{S}_k \cup \{B_{k+1}\}$. This, together with $|\mathcal{S}_k| = k$ (by induction hypothesis), implies $|\mathcal{S}_{k+1}| = k + 1$. □

We have proved all four parts of Statement S_{k+1}: part 1 in Claims 4 and 2, Part 2 in Claim 5, part 3 in Claim 7; part 4 follows from parts (b) and (c) of Claim 6 and Claim 7(a). This completes the induction step and hence the proof of Statement S_k for all $0 \leq k \leq n-1$.

We now proceed to prove Theorem 3.2. The first part of the theorem is immediate from part 3 of Statement S_{n-1}. To obtain the second part of the theorem, observe that a deterministic implementation can be viewed as a randomized implementation for which COINSPACE is a singleton set. Since Statement S_k $(0 \leq k \leq n-1)$ is proved for any nonempty countable COINSPACE, Statement S_{n-1} is true for any deterministic implementation. By part 3 of Statement S_{n-1}, in $\Lambda_{n-1}\Sigma_{n-1}\Pi_{n-1}$, p_n accesses $n-1$ base objects and has either not completed or just completed its first operation on $\mathcal{O}$. This implies that the solo-termination time complexity is at least $n-1$. Hence, we have the theorem. □

4. Examples of perturbable types. We show that the following common types of objects are perturbable for n processes: modulo k counter for any $k \geq 2n$, increment object, fetch&add, k-valued compare&swap for any $k \geq n$, LL/SC bit, and single-writer snapshot. It follows from Theorem 3.2 that the space complexity of a randomized implementation or the time complexity of a deterministic implementation of any of these objects from resettable consensus objects and historyless objects is at least $n-1$.

4.1. Modulo counter and related objects. A *modulo k counter* supports *increment* and *read* operations. The states are $0, 1, \ldots, k-1$. The *increment* operation adds 1 to the state (modulo k) and returns *ack*. The *read* operation returns the state without affecting it. The following proposition is immediate from the linearizability requirement.

PROPOSITION 4.1. *Let $\mathcal{O}$ be a modulo k counter, initialized to 0. Let E be a finite execution of $(p_1, \ldots, p_n; \mathcal{O})$ such that in the configuration C at the end of E, process p_n has no pending operation on $\mathcal{O}$. Suppose p_n runs alone from C and performs a read operation on $\mathcal{O}$. If the number of completed increments in E is at least v and the sum in E of the number of completed increments and the number of pending increments is at most v', then the value returned by the read of p_n is in the range $[v, v'] \bmod k$.*

LEMMA 4.2. *For all $k \geq 2n$, modulo k counter is perturbable for n processes, for any initial state.*

Proof. Without loss of generality, we prove the lemma for initial state 0. Consider any linearizable and solo-terminating randomized implementation of a modulo k counter $\mathcal{O}$, initialized to 0 and shared by processes $p_1, \ldots, p_n$. For $1 \leq i \leq n-1$, let *op-list*$_i$ be an infinite sequence of *increment* operations, and *op-list*$_n$ be an infinite sequence of *read* operations. Let Λ, Σ, and Π be any schedules that satisfy the four conditions listed in Definition 3.1.

Let p_l be any process in $\{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$, and $\gamma \in (\{p_l\} \times \text{COINSPACE})^*$ be the shortest schedule such that there are exactly n more completed increment operations on $\mathcal{O}$ in $\Lambda\gamma$ than in Λ. Since the implementation is solo-terminating, γ exists. We now make the following observations:

(1) If a process completes an increment on $\mathcal{O}$ in the last $|\Sigma|$ steps of $\Lambda\Sigma$, then it has no pending increment on $\mathcal{O}$ in $\Lambda\Sigma$. Furthermore, no process completes more than one increment on $\mathcal{O}$ in the last $|\Sigma|$ steps of $\Lambda\Sigma$.

This follows from the fact that each process appears at most once in the schedule Σ.

(2) For any execution E, let $\mathrm{NP}(E)$ denote the number of pending increment operations on $\mathcal{O}$ in E. The sum of $\mathrm{NP}(\Lambda\Sigma)$ and the number of increments that completed in the last $|\Sigma|$ steps of $\Lambda\Sigma$ is at most $n-1$.
This follows from (1) and the fact that $\textsc{pset}(\Lambda\Sigma) \subseteq \{p_1, \ldots, p_{n-1}\}$.

(3) The sum of $\mathrm{NP}(\Lambda\gamma\Sigma)$ and the number of increments that completed in the last $|\Sigma|$ steps of $\Lambda\gamma\Sigma$ is at most $n-1$.
This also follows from (1) and the fact that $\textsc{pset}(\Lambda\gamma\Sigma) \subseteq \{p_1, \ldots, p_{n-1}\}$.

(4) For any execution E, let $\mathrm{NC}(E)$ denote the number of completed increment operations on $\mathcal{O}$ in E. Let $\mathrm{NC}(\Lambda) = v$. In $\Lambda\Sigma\Pi$, the value res, which p_n's first operation on $\mathcal{O}$ (which is a read) returns, is in the range $[v, v+n-1] \bmod k$.
This follows from Proposition 4.1 and the following two chains of inequalities:

$$\mathrm{NC}(\Lambda\Sigma) \geq \mathrm{NC}(\Lambda) = v,$$

$$\begin{aligned}\mathrm{NC}(\Lambda\Sigma) + \mathrm{NP}(\Lambda\Sigma) &= \mathrm{NC}(\Lambda) + \mathrm{NP}(\Lambda\Sigma) \\ &\quad + \text{number of increments that completed in} \\ &\quad\quad \text{the last } |\Sigma| \text{ steps of } \Lambda\Sigma \\ &\leq v + n - 1 \qquad \text{(by (2))}.\end{aligned}$$

(5) In $\Lambda\gamma\Sigma\Pi$, if p_n's first operation on $\mathcal{O}$ completes, it returns a value in the range $[v+n, v+2n-1] \bmod k$.
This follows from Proposition 4.1 and the following two chains of inequalities:

$$\mathrm{NC}(\Lambda\gamma\Sigma) \geq \mathrm{NC}(\Lambda\gamma) = v + n,$$

$$\begin{aligned}\mathrm{NC}(\Lambda\gamma\Sigma) + \mathrm{NP}(\Lambda\gamma\Sigma) &= \mathrm{NC}(\Lambda\gamma) + \mathrm{NP}(\Lambda\gamma\Sigma) \\ &\quad + \text{number of increments that completed in} \\ &\quad\quad \text{the last } |\Sigma| \text{ steps of } \Lambda\gamma\Sigma \\ &\leq (v+n) + (n-1) \qquad \text{(by (3))}.\end{aligned}$$

Since $k \geq 2n$, the range $[v, v+n-1] \bmod k$ and the range $[v+n, v+2n-1] \bmod k$ are disjoint. This, together with (4) and (5), implies Lemma 4.2. □

An *increment object* is a special case of a modulo k counter for $k = \infty$. Since the finiteness of k is not used in the proofs of Proposition 4.1 or Lemma 4.2, we have the following result.

LEMMA 4.3. *An increment object is perturbable for n processes, for any initial state.*

A *fetch&add object* supports the operation fetch&add(v), for any integer v. The states are integers. The fetch&add(v) operation adds v to the state and returns the previous state. Proceeding analogously as in the proof of Lemma 4.2, with $k = \infty$ and the operations *increment*, *read* replaced by *fetch&add*(1), *fetch&add*(0), we obtain the following lemma.

LEMMA 4.4. *A fetch&add object is perturbable for n processes, for any initial state.*

4.2. Compare&swap. A *k-valued compare&swap object* supports the operations *read* and $c\&s(u, v)$ for all $u, v \in \{1, 2, \ldots, k\}$. The states are $1, 2, \ldots, k$. The effect of $c\&s(u, v)$ depends on whether or not the state is u: if the state is u, $c\&s(u, v)$

changes the state to v and returns *true*; otherwise it returns *false* without affecting the state. We say a compare&swap operation is *successful* if it returns *true*.

PROPOSITION 4.5. *Let C be any reachable configuration of $(p_1, \ldots, p_n; \mathcal{O})$, where $\mathcal{O}$ is a k-valued compare&swap object. Suppose that process p_l has no pending operations on $\mathcal{O}$ in C. For any $w \in \{1, 2, \ldots, k\}$, if p_l runs alone from C, completing the sequence of operations read, c&s$(1, w)$, c&s$(2, w)$, $\ldots$, c&s(k, w), then one of the following is true:*

(1) *One of the c&s operations of p_l returns true.*
(2) *Some operation on $\mathcal{O}$ that was pending in C is linearized after the read and before the last c&s operation of p_l.*

Proof. Let v be the value returned by the read operation of p_l. Suppose that statement (1) in the proposition is false. Since c&s(v, w), which is one of the n c&s operations that p_l performed following the read, did not return true, some pending operation must have taken effect after the read and before the c&s(v, w). ∎

PROPOSITION 4.6. *Let C be any reachable configuration of $(p_1, \ldots, p_n; \mathcal{O})$, where $\mathcal{O}$ is a k-valued compare&swap object. Suppose that process p_l has no pending operations on $\mathcal{O}$ in C. Let $w \in \{1, 2, \ldots, k\}$ and suppose that p_l runs alone from C, completing the following sequence of operations n times: read, c&s$(1, w)$, c&s$(2, w)$, $\ldots$, c&s(k, w). Then, at least one of the c&s operations returns true.*

Proof. The proposition follows by successive application of Proposition 4.5 and the observation that there can be at most $n-1$ pending operations on $\mathcal{O}$ in C. ∎

LEMMA 4.7. *For all $k \geq n$, k-valued compare&swap object is perturbable for n processes for any initial state.*

Proof. Consider any linearizable and solo-terminating randomized implementation of a k-valued compare&swap object $\mathcal{O}$, initialized to any value and shared by processes $p_1, \ldots, p_n$. For any $1 \leq j \leq k$, let α_j denote the operation sequence *read*, c&s$(1, j)$, c&s$(2, j)$, $\ldots$, c&s(k, j). Let β denote the operation sequence $\alpha_1^n \alpha_2^n \cdots \alpha_k^n$, where α_i^m denotes the sequence α_i repeated m times. Thus, $|\alpha_j| = k + 1$ and $|\beta| = nk(k + 1)$. For all $1 \leq i \leq n - 1$, initialize the input variable *op-list*$_i$ to the infinite sequence $\beta\beta\beta, \ldots$, and initialize *op-list*$_n$ to the infinite sequence of *read* operations. Let Λ, Σ, and Π be any schedules that satisfy the four conditions listed in Definition 3.1.

Let p_l be any process in $\{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$. If p_l has any pending operation on $\mathcal{O}$ at the end of Λ, let $\gamma' \in (\{p_l\} \times \text{COINSPACE})^*$ be such that p_l just completes that operation in $\Lambda\gamma'$. Otherwise let $\gamma' = \epsilon$. Thus, at the end of $\Lambda\gamma'$, p_l has no pending operation on $\mathcal{O}$, and any pending operations have to be from processes in $\{p_1, \ldots, p_{n-1}\} - \{p_l\}$. Let $P \subseteq \{p_1, \ldots, p_{n-1}\} - \{p_l\}$ be the set of processes that have pending operations on $\mathcal{O}$ at the end of $\Lambda\gamma'$.

Let Q be the set of processes that initiate a new operation on $\mathcal{O}$ in the last $|\Sigma|$ steps of $\Lambda\gamma'\Sigma$. Since $p_l \in \{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$, we have $Q \subseteq \{p_1, \ldots, p_{n-1}\} - \{p_l\}$. Furthermore, since each process appears at most once in Σ, if a process has a pending operation in $\Lambda\gamma'$, then that process cannot initiate a new operation on $\mathcal{O}$ in the last $|\Sigma|$ steps of $\Lambda\gamma'\Sigma$. In other words, $P \cap Q = \emptyset$. From this and the fact $P, Q \subseteq \{p_1, \ldots, p_{n-1}\} - \{p_l\}$, we have $|P| + |Q| \leq n - 2$. That is, the sum of the number of pending operations on $\mathcal{O}$ in $\Lambda\gamma'$ and the number of operations on $\mathcal{O}$ initiated in the last $|\Sigma|$ steps of $\Lambda\gamma'\Sigma$ is at most $n - 2$. Let V be the set of all v such that a c&s$(v, *)$ operation[5] on $\mathcal{O}$ is either pending in $\Lambda\gamma'$ or initiated in the last $|\Sigma|$ steps of $\Lambda\gamma'\Sigma$. From the above, $|V| \leq n - 2$.

[5] An asterisk in a field indicates that we do not care what the value of that field is.

Recall from (the fourth condition in) Definition 3.1 that res is the value returned by p_n's first operation on $\mathcal{O}$ in $\Lambda\Sigma\Pi$. Let $w \in \{1, 2, \ldots, n\}$ be such that $w \notin V$ and $w \neq res$. Let $\gamma'' \in (\{p_l\} \times \text{COINSPACE})^*$ be the shortest schedule such that, in $\Lambda\gamma'\gamma''$, we have the following: (i) p_l has no pending operations, (ii) there are at least $n(k+1)$ completed operations on $\mathcal{O}$ (by p_l) in the last $|\gamma''|$ steps, and (iii) the sequence of $n(k+1)$ most recent completed operations of p_l on $\mathcal{O}$ is α_w^n. The definition of $op\text{-}list_l$ and the fact that the implementation is solo-terminating imply that γ'' exists. We now make the following observations:

(1) In $\Lambda\gamma'\gamma''$, the most recent $n(k+1)$ operations of p_l on $\mathcal{O}$ includes a *successful* compare&swap operation of the form c and $s(*, w)$. (Let OP denote any such operation.)

Proof. In $\Lambda\gamma'\gamma''$, the most recent $n(k+1)$ operations of p_l on $\mathcal{O}$ are the operations in α_w^n. All of the compare&swap operations in α_w^n are of the form $c\&s(*, w)$ and, by Proposition 4.6, at least one of these succeeds. □

(2) Consider any linearization of $\Lambda\gamma'\gamma''\Sigma$. If OP′ is a successful compare&swap operation that is linearized after OP, then OP′ must be of the form $c\&s(*, w)$.

Proof. We prove this assertion by contradiction. Let OP′ be the first successful compare&swap operation that is of the form $c\&s(*, x)$, for some $x \neq w$, to be linearized after OP. Since OP is successful and each successful compare&swap operation that is linearized after OP and before OP′ is of the form $c\&s(*, w)$, the value of the object is w immediately before OP′.

There are three cases to consider: (i) OP′ is an operation from p_l that follows OP, (ii) OP′ is an operation which is pending in $\Lambda\gamma'$, or (iii) OP′ is an operation which is initiated in the last $|\Sigma|$ steps of $\Lambda\gamma'\gamma''\Sigma$. In case (i), by definition of γ'' and OP, OP′ is of the form $c\&s(*, w)$, a contradiction. In cases (ii) and (iii), by definitions of V and w, OP′ $= c\&s(v, *)$ for some $v \neq w$. Since the value of the object is w immediately before OP′, the fact $v \neq w$ implies that OP′ $= c\&s(v, *)$ cannot be successful, which is a contradiction. □

(3) In $\Lambda\gamma'\gamma''\Sigma\Pi$, if p_n's first operation on $\mathcal{O}$ (which is a read operation) completes, it returns a value different from res.

Proof. Since OP is successful and is of the form $c\&s(*, w)$, the value of $\mathcal{O}$ immediately after OP is w. By the previous observation, in $\Lambda\gamma'\gamma''\Sigma$, every successful compare&swap linearized after OP is also of the form $c\&s(*, w)$.

Therefore, in $\Lambda\gamma'\gamma''\Sigma\Pi$, if p_n's first operation on $\mathcal{O}$ (which is a read operation) completes, it returns w. But w, by definition, is different from res. Hence, we have the observation. □

Lemma 4.7 is immediate from the last observation. □

4.3. LL/SC bit. An *n-process load-link store-conditional* (LL/SC) *bit* supports the operations LL and $SC(b)$ for $b = 0, 1$. The states are pairs (v, S) for all $v \in \{0, 1\}$ and $S \subseteq \{1, 2, \ldots, n\}$. The operation LL from process p_i, when applied in state (v, S), returns v and changes the state to (v, S'), where $S' = S \cup \{i\}$. The operation $SC(b)$ from process p_i, when applied in state (v, S), has the following effect: if $i \in S$, the state changes to $(b, \emptyset)$ and *true* is returned; otherwise the state is not affected and *false* is returned. We say an SC operation is *successful* if it returns *true*.

PROPOSITION 4.8. *Let C be any reachable configuration of $(p_1, \ldots, p_n; \mathcal{O})$, where $\mathcal{O}$ is an n-process LL/SC bit. Suppose that process p_l has no pending operations on $\mathcal{O}$ in C. If p_l runs alone from C, completing an LL operation and then an $SC(b)$ operation (for any $b \in \{0, 1\}$), then one of the following is true:*

(1) *The $SC(b)$ operation of p_l returns true.*

(2) *Some SC operation on $\mathcal{O}$ that was pending in C is linearized after the LL and before the SC(b) of p_l.*

Proof. The proposition follows from the specification of n-process LL/SC bit. ∎

PROPOSITION 4.9. *Let C be any reachable configuration of $(p_1, \ldots, p_n; \mathcal{O})$, where $\mathcal{O}$ is an n-process LL/SC bit. Suppose that process p_l has no pending operations on $\mathcal{O}$ in C. For $b \in \{0, 1\}$, suppose further that p_l runs alone from C, completing the following sequence of operations n times: LL, SC(b). Then at least one of the SC(b) operations returns true.*

Proof. The proposition follows by repeated application of Proposition 4.8 and the observation that there can be at most $n - 1$ pending operations on $\mathcal{O}$ in C. ∎

LEMMA 4.10. *LL/SC bit is perturbable for n processes for any initial state.*

Proof. Consider any linearizable and solo-terminating randomized implementation of an LL/SC bit $\mathcal{O}$, initialized to any value and shared by processes $p_1, \ldots, p_n$. For $j \in \{0, 1\}$, let α_j denote the sequence $LL, SC(j), LL, SC(j), \ldots, LL, SC(j)$ that has a total of $2n$ operations (n LL operations and n $SC(j)$ operations). For all $1 \leq i \leq n-1$, initialize the input variable *op-list*$_i$ to the infinite sequence $\alpha_0, \alpha_1, \alpha_0, \alpha_1, \alpha_0, \alpha_1, \ldots$ and initialize *op-list*$_n$ to the infinite sequence of LL operations. Let Λ, Σ, and Π be any schedules that satisfy the four conditions listed in Definition 3.1.

Let p_l be any process in $\{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$. If p_l has any pending operation on $\mathcal{O}$ at the end of Λ, let $\gamma' \in (\{p_l\} \times \text{COINSPACE})^*$ be such that p_l just completes that operation in $\Lambda\gamma'$. Otherwise let $\gamma' = \epsilon$. Thus, at the end of $\Lambda\gamma'$, p_l has no pending operation on $\mathcal{O}$, but other processes may. Any such pending operations have to be from processes in $\{p_1, \ldots, p_{n-1}\} - \{p_l\}$.

Recall from (the fourth condition in) Definition 3.1 that *res* is the value returned by p_n's first operation on $\mathcal{O}$ in $\Lambda\Sigma\Pi$. Let $w = 1 - res$. Let $\gamma'' \in (\{p_l\} \times \text{COINSPACE})^*$ be the shortest schedule such that, in $\Lambda\gamma'\gamma''$, we have the following: (i) p_l has no pending operations, (ii) there are at least $2n$ completed operations on $\mathcal{O}$ (by p_l) in the last $|\gamma''|$ steps, and (iii) the sequence of $2n$ most recent operations of p_l on $\mathcal{O}$ is α_w. The definition of *op-list*$_l$ and the fact that the implementation is solo-terminating imply that γ'' exists. We now make the following observations:

(1) In $\Lambda\gamma'\gamma''$, the most recent $2n$ operations of p_l on $\mathcal{O}$ include a successful $SC(w)$ operation. (Let OP denote any such operation.)

Proof. Consider the sequence α_w of the $2n$ most recent (alternating LL and $SC(w)$) operations of p_l on $\mathcal{O}$. By Proposition 4.9, at least one of these $SC(w)$ operations succeeds. ∎

(2) Consider any linearization of $\Lambda\gamma'\gamma''\Sigma$. Let OP$' = SC(v)$ be any successful operation that is linearized after OP. Then $v = w$.

Proof. If OP$'$ is linearized after OP, there are three cases to consider: (i) OP$'$ is an operation from p_l that follows OP, (ii) OP$'$ is an operation which is pending in $\Lambda\gamma'$, or (iii) OP$'$ is an operation which is initiated in the last $|\Sigma|$ steps of $\Lambda\gamma'\gamma''\Sigma$. In the following we show that the observation holds in all cases.

Case (i). OP$'$ is an operation from p_l that follows OP.

Since $p_l \in \{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$, p_l has no step in the last $|\Sigma|$ steps of $\Lambda\gamma'\gamma''\Sigma$. From this and the definition of γ'', the $2n$ most recent operations from p_l in $\Lambda\gamma'\gamma''\Sigma$ are LL, $SC(w)$, LL, $SC(w)$, …, LL, $SC(w)$. By definition, OP is one of these $2n$ operations. Thus, if OP$' = SC(v)$ is an SC operation from p_l that follows OP, then v must equal w.

Cases (ii) and (iii). OP$'$ is pending in $\Lambda\gamma'$ or OP$'$ is initiated in the last $|\Sigma|$

steps of $\Lambda\gamma'\gamma''\Sigma$.

Let p_i be the process that invoked $\text{OP}' = SC(v)$. Consider the most recent LL operation from p_i that preceded OP'. Let OP'' denote this operation. We assert that in both Case (ii) and Case (iii), OP'' completed in Λ. In the next two paragraphs we prove this assertion for the two cases.

In Case (ii), since OP' is pending in $\Lambda\gamma'$, OP'' must have completed in $\Lambda\gamma'$. Since $\text{PSET}(\gamma'\gamma'') = \{p_l\}$ and $p_l \neq p_i$ (because, unlike p_i, p_l has no pending operation in $\Lambda\gamma'$), it follows that p_i completed OP'' in Λ.

In Case (iii), since each process appears at most once in Σ, if p_i initiated OP' in the last $|\Sigma|$ steps of $\Lambda\gamma'\gamma''\Sigma$, then it follows that p_i completed OP'' in $\Lambda\gamma'\gamma''$. Further, since $\text{PSET}(\gamma'\gamma'') = \{p_l\}$ and $p_l \neq p_i$ (because $p_l \in \{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$), it follows that p_i completed OP'' in Λ.

Since OP did not even begin in Λ, it follows that OP'' is linearized before OP. Thus, we have the following situation: p_i applied the LL operation OP'' and then the SC operation OP'; p_l's successful SC operation OP is linearized after OP'' and before OP'. By the specification of LL/SC bit, OP' must return *false*. This contradicts the premise that OP' is successful. Thus Cases (ii) and (iii) cannot arise. □

(3) In $\Lambda\gamma'\gamma''\Sigma\Pi$, if p_n's first operation on $\mathcal{O}$ (which is an LL operation) completes, it returns a response different from res.

Proof. Since $\text{OP} = SC(w)$ is successful, the value of $\mathcal{O}$ immediately after OP is w. By the previous observation, in $\Lambda\gamma'\gamma''\Sigma$, every successful SC linearized after OP is also of the form $SC(w)$. Therefore, in $\Lambda\gamma'\gamma''\Sigma\Pi$, p_n's first operation on $\mathcal{O}$ (which is an LL operation) returns w. Since $w = 1 - res$, we have $w \neq res$. Hence, we have the observation. □

Lemma 4.10 is immediate from the last observation. □

4.4. Single-writer snapshot. An *n-process single-writer binary snapshot object* [AAD+93, And93] supports the operations *read* and *write* v for $v \in \{0, 1\}$. The states are $[v_1, v_2, \ldots, v_n]$, where $v_1, v_2, \ldots, v_n$ are from $\{0, 1\}$. A *write* x operation from process p_i, when applied in state $[v_1, v_2, \ldots, v_n]$, changes the state to $[v_1, \ldots, v_{i-1}, x, v_{i+1}, \ldots, v_n]$ and returns *ack*. The *read* operation, when applied in state $[v_1, v_2, \ldots, v_n]$, returns $[v_1, v_2, \ldots, v_n]$ without affecting the state.

LEMMA 4.11. *Single-writer binary snapshot object is perturbable for n processes for any initial state.*

Proof. Consider any linearizable and solo-terminating randomized implementation of an n-process single-writer binary snapshot object $\mathcal{O}$, initialized to any value and shared by processes $p_1, \ldots, p_n$. For $1 \leq i \leq n-1$, let $\textit{op-list}_i$ be an infinite sequence of alternating *write 0* and *write* 1 operations. Let $\textit{op-list}_n$ be an infinite sequence of *read* operations. Let Λ, Σ, and Π be any schedules that satisfy the four conditions listed in Definition 3.1.

Recall from (the fourth condition in) Definition 3.1 that res is the value returned by p_n's first operation on $\mathcal{O}$ in $\Lambda\Sigma\Pi$. Let $res = [v_1, v_2, \ldots, v_n]$. Let p_l be any process in $\{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$. Let $\gamma \in (\{p_l\} \times \text{COINSPACE})^*$ be the shortest schedule such that, in $\Lambda\gamma$, p_l just completed writing $1 - v_l$. Since the implementation is solo-terminating, γ exists. Further, since $p_l \in \{p_1, \ldots, p_{n-1}\} - \text{PSET}(\Sigma)$, p_l has no step in the last $|\Sigma|$ steps of $\Lambda\gamma\Sigma$. Therefore, if p_n's first operation in $\Lambda\gamma\Sigma\Pi$ (which is a read operation) completes and returns $[w_1, w_2, \ldots, w_n]$, w_l must equal $1 - v_l$. It follows that $res \neq [w_1, w_2, \ldots, w_n]$. Hence, we have Lemma 4.11. □

5. Applications. In section 4, we showed that every type in set A is perturbable for n processes, for any initial state, where $A = \{$modulo k counter (for any $k \geq 2n$), increment, fetch&add, k-valued compare&swap (for any $k \geq n$), LL/SC bit, single-writer snapshot$\}$ (see Lemmas 4.2, 4.3, 4.4, 4.7, 4.10, and 4.11). From this and Theorem 3.2, we have the following.

THEOREM 5.1. *Let A be the set of types defined above. Consider any randomized implementation of an object belonging to a type in A, initialized to any state and shared by processes $p_1, \ldots, p_n$, from resettable consensus objects and historyless objects. If the implementation is linearizable and solo-terminating, then*

(1) *its space complexity is at least $n - 1$;*
(2) *if the implementation is deterministic, its solo-termination time complexity is at least $n - 1$.*

The above result does not address the complexity of implementing modulo k counter when $k < 2n$, or of implementing k-valued compare&swap when $k < n$. We discuss these cases below. The following corollary is a simple consequence of Lemma 4.2.

COROLLARY 5.2. *For all $k \geq 1$, modulo k counter is perturbable for $\lfloor k/2 \rfloor$ processes for any initial state.*

From Corollary 5.2 and Theorem 3.2, we have the following.

COROLLARY 5.3. *For any positive integer k, consider any randomized implementation of modulo k counter, initialized to any state and shared by processes $p_1, \ldots, p_{\lfloor k/2 \rfloor}$, from resettable consensus objects and historyless objects. If the implementation is linearizable and solo-terminating, then*

(1) *its space complexity is at least $\lfloor k/2 \rfloor - 1$;*
(2) *if the implementation is deterministic, its solo-termination time complexity is at least $\lfloor k/2 \rfloor - 1$.*

We observe that the time or space complexity grows monotonically with the number of processes sharing the implementation. This observation, together with Corollary 5.3, gives the following.

THEOREM 5.4. *For any $k \leq 2n$, consider any randomized implementation of modulo k counter, initialized to any state and shared by processes $p_1, \ldots, p_n$, from resettable consensus objects and historyless objects. If the implementation is linearizable and solo-terminating, then*

(1) *its space complexity is at least $\lfloor k/2 \rfloor - 1$;*
(2) *if the implementation is deterministic, its solo-termination time complexity is at least $\lfloor k/2 \rfloor - 1$.*

Using Lemma 4.7 and reasoning as above, we have the following.

THEOREM 5.5. *For any $k \leq n$, consider any randomized implementation of k-valued compare&swap, initialized to any state and shared by processes $p_1, \ldots, p_n$, from resettable consensus objects and historyless objects. If the implementation is linearizable and solo-terminating, then*

(1) *its space complexity is at least $k - 1$;*
(2) *if the implementation is deterministic, its solo-termination time complexity is at least $k - 1$.*

Acknowledgments. We thank Faith Fich and Nir Shavit, and the SICOMP and PODC '96 referees for many helpful suggestions on how to improve the proof and the general presentation of the paper.

REFERENCES

[AAD+93] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. Assoc. Comput. Mach., 40 (1993), pp. 873–890.

[And93] J. Anderson, *Composite registers*, Distrib. Comput., 6 (1993), pp. 141–154.

[Asp90] J. Aspnes, *Time and space efficient randomized consensus*, in Proceedings of the Ninth ACM Symposium on Principles of Distributed Computing, Quebec City, Canada, 1990, pp. 325–332.

[Aum97] Y. Aumann, *Efficient asynchronous consensus with the weak adversary scheduler*, in Proceedings of the 16th Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, pp. 209–218.

[BL93] J. E. Burns and N. A. Lynch, *Bounds on shared memory for mutual exclusion*, Inform. and Comput., 107 (1993), pp. 171–184.

[Cha96] T. D. Chandra, *Polylog randomized wait-free consensus*, in Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, PA, 1996, pp. 166–175.

[CJT98] T. D. Chandra, P. Jayanti, and K. Y. Tan, *A polylog time wait-free construction for closed objects*, in Proceedings of the 17th Annual ACM Symposium on Principles of Distributed Computing, Puerto Vallarta, Mexico, 1998, pp. 287–296.

[FHS93] F. Fich, M. Herlihy, and N. Shavit, *On the space complexity of randomized synchronization*, in Proceedings of the 12th Annual ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 241–249.

[FHS98] F. Fich, M. Herlihy, and N. Shavit, *On the space complexity of randomized synchronization*, J. Assoc. Comput. Mach., 45 (1998), pp. 843–862.

[Her88] M. P. Herlihy, *Impossibility and universality results for wait-free synchronization*, in Proceedings of the Seventh ACM Symposium on Principles of Distributed Computing, Toronto, Canada, 1988, pp. 276–290.

[Her91] M. P. Herlihy, *Wait-free synchronization*, ACM Transactions on Programming Languages and Systems, 13 (1991), pp. 124–149.

[HS93] M. P. Herlihy and N. Shavit, *The asynchronous computability theorem for t-resilient tasks*, in Proceedings of the 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 111–120.

[HW90] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Transactions on Programming Languages and Systems, 12 (1990), pp. 463–492.

[Plo89] S. Plotkin, *Sticky bits and universality of consensus*, in Proceedings of the Eighth ACM Symposium on Principles of Distributed Computing, Edmonton, Canada, 1989, pp. 159–175.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 457–474

EFFICIENT SEARCH FOR APPROXIMATE NEAREST NEIGHBOR IN HIGH DIMENSIONAL SPACES*

EYAL KUSHILEVITZ†, RAFAIL OSTROVSKY‡, AND YUVAL RABANI†

Abstract. We address the problem of designing data structures that allow efficient search for approximate nearest neighbors. More specifically, given a database consisting of a set of vectors in some high dimensional Euclidean space, we want to construct a space-efficient data structure that would allow us to search, given a query vector, for the closest or nearly closest vector in the database. We also address this problem when distances are measured by the L_1 norm and in the Hamming cube. Significantly improving and extending recent results of Kleinberg, we construct data structures whose size is polynomial in the size of the database and search algorithms that run in time nearly linear or nearly quadratic in the dimension. (Depending on the case, the extra factors are polylogarithmic in the size of the database.)

Key words. nearest neighbor search, data structures, random projections

AMS subject classification. 68Q25

PII. S0097539798347177

1. Introduction.

Motivation. Searching for a nearest neighbor among a specified database of points is a fundamental computational task that arises in a variety of application areas, including information retrieval [32, 33], data mining [20], pattern recognition [8, 14], machine learning [7], computer vision [4], data compression [18], and statistical data analysis [10]. In many of these applications the database points are represented as vectors in some high dimensional space. For example, latent semantic indexing is a recently proposed method for textual information retrieval [9]. The semantic contents of documents, as well as the queries, are represented as vectors in $\mathbb{R}^d$, and proximity is measured by some distance function. Despite the use of dimension reduction techniques such as principal component analysis, vector spaces of several hundred dimensions are typical. Multimedia database systems, such as IBM's QBIC [16] or MIT's Photobook [31], represent features of images and queries similarly. In such applications, the mapping of attributes of objects to coordinates of vectors is heuristic, and so is the choice of metric. Therefore, an approximate search is just as good as an exact search and is often used in practice.

The problem. Let $\mathcal{V}$ be some (finite or infinite) vector space of dimension d, and let $\|\cdot\|$ be some norm (Minkowsky distance function) for $\mathcal{V}$. Given a database consisting of n vectors in $\mathcal{V}$, a slackness parameter $\epsilon > 0$, and a query vector q, a $(1+\epsilon)$-approximate nearest neighbor of q is a database vector a such that for any other database vector b, $\|q-a\| \leq (1+\epsilon)\|q-b\|$. We consider the following problem.

*Received by the editors November 13, 1998; accepted for publication (in revised form) August 17, 1999; published electronically June 3, 2000. A preliminary version of this paper appeared in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 614–623.

http://www.siam.org/journals/sicomp/30-2/34717.html

†Computer Science Department, Technion—IIT, Haifa 32000, Israel (eyalk@cs.technion.ac.il, rabani@cs.technion.ac.il). Part of this work was done while the third author was visiting Bell Communications Research. Work at Technion supported by BSF grant 96-00402, by a David and Ruth Moskowitz Academic Lecturship award, and by grants from the S. and N. Grand Research Fund, the Smoler Research Fund, and the Fund for the Promotion of Research at Technion.

‡Bell Communications Research, MCC-1C365B, 445 South Street, Morristown, NJ 07960-6438 (rafail@bellcore.com).

Given such a database and $\epsilon > 0$, design a data structure $\mathcal{S}$ and a $(1+\epsilon)$-approximate nearest neighbor search algorithm (using $\mathcal{S}$).

We aim at efficient construction of $\mathcal{S}$ as well as quick lookup. Our performance requirements are the following. (i) The algorithm for constructing $\mathcal{S}$ should run in time polynomial in n and d (and thus the size of $\mathcal{S}$ is polynomial in n and d). (ii) The search algorithm should improve significantly over the naïve (brute force) $O(dn)$ time exact algorithm. More precisely, we aim at search algorithms using (low) polynomial in d and $\log n$ arithmetic operations.

Our results. We obtain results for the Hamming cube ($\{0,1\}^d$ with the L_1 norm) for Euclidean spaces ($\mathbb{R}^d$ with the L_2 norm) and for ℓ_1^d ($\mathbb{R}^d$ with the L_1 norm). Our results for the cube generalize to vector spaces over any finite field; thus we can handle a database of documents (strings) over any finite alphabet. Our results for Euclidean spaces imply similar bounds for distance functions used in latent semantic indexing (these are not metrics, but their square root is Euclidean).

Our data structures are of size $(dn)^{O(1)}$. For the d-dimensional Hamming cube as well as for ℓ_1^d, our search algorithm runs in time $O(d\text{poly}\log(dn))$ (the logarithmic factors are different in each case). For d-dimensional Euclidean spaces, our search algorithm runs in time $O(d^2\text{poly}\log(dn))$. (The big-Oh notation hides factors polynomial in $1/\epsilon$.)

Our algorithms are probabilistic. They succeed with any desired probability (at the expense of time and space complexity). We have to make precise the claims for success: The algorithm that constructs $\mathcal{S}$ succeeds with high probability, and if successful, $\mathcal{S}$ is good for every possible query. If $\mathcal{S}$ has been constructed successfully, then given any query, the search algorithm succeeds to find an approximate nearest neighbor with high probability, and this probability can be increased as much as we like *without* modifying $\mathcal{S}$ (just by running the search algorithm several times). An alternative, weaker guarantee is to construct a data structure that is good for most queries. Our algorithms provide the stronger guarantee. (This means that they can work when the queries are generated by an adversary that has access to the random bits used in the construction of the data structure.) Much of the difficulty arises from this requirement.

Related work. In computational geometry, there is a vast amount of literature on proximity problems in Euclidean spaces, including nearest neighbor search and the more general problem of point location in an arrangement of hyperplanes. We dare not attempt to survey but the most relevant papers to our work.

There are excellent solutions to nearest neighbor search in low (two or three) dimensions. For more information see, e.g., [30]. In high dimensional space, the problem was first considered by Dobkin and Lipton [11]. They showed an exponential in d search algorithm using (roughly) a double-exponential in d (summing up time and space) data structure. This was improved and extended in subsequent work of Clarkson [5], Yao and Yao [35], Matoušek [28], Agarwal and Matoušek [1], and others, all requiring query time exponential in d. Recently, Meiser [29] obtained a polynomial in d search algorithm using an exponential in d size data structure.

For approximate nearest neighbor search, Arya et al. [3] gave an exponential in d time search algorithm using a linear size data structure. Clarkson [6] gave a search algorithm with improved dependence on ϵ. Recently, Kleinberg [24] gave two algorithms that seem to be the best results for large d prior to this work. The first algorithm searches in time $O(d^2 \log^2 d + d \log^2 d \log n)$ but requires a data structure of size $O(n \log d)^{2d}$. The second algorithm uses a small data structure (nearly linear

in dn) and takes $O(n + d \log^3 n)$ time (so it beats the brute force search algorithm).

Independently of our work, Indyk and Motwani [21] obtained several results on essentially the same problems as we discuss here. Their main result gives an $O(d\text{poly}\log(dn))$ search algorithm using a data structure of size $O\left(n(1/\epsilon)^{O(d)}\right.$ $\left.\text{poly}\log(dn)\right)$ for Euclidean and other norms. They obtain this result using space partitions induced by spheres, and bucketing. In comparison with our work, they use exponential in d (but not in $1/\epsilon$) storage in contrast with our polynomial in d storage. Their search time is better than ours for the Euclidean case and similar for the L_1 norm. They also point out that dimension reduction techniques, such as those based on random projections, can be used in conjunction with their other results to get polynomial size data structures which are good for any single query with high probability. However, the data structure always fails on some queries (so an adversary with access to the random bits used in the construction can present a bad query).

Also related to our work are constructions of hash functions that map "close" elements to "close" images. In particular, *nonexpansive* hash functions guarantee that the distance between images is at most the distance between the original elements. However, such families are known only for one-dimensional points (Linial and Sasson [27]). For d-dimensional points, Indyk et al. [22] construct hash functions that increase the distance by a bounded additive term (d or $\sqrt{d}$ depending on the metric). These results do not seem useful for approximate nearest neighbor search as they can increase a very small distance δ to a distance which is much larger than $(1+\epsilon)\delta$. Dolev et al. [13, 12] construct hash functions that map all elements at distance at most ℓ to "close" images. These constructions, too, do not seem useful for approximate nearest neighbor search, because the construction time is exponential in ℓ.

Our methods. Our data structure and search algorithm for the hypercube is based on an inner product test. Similar ideas have been used in a cryptographic context by [19] and as a matter of folklore to design equality tests (see, e.g., [25]). Here, we refine the basic idea to be sensitive to distances. For Euclidean spaces, we reduce the problem essentially to a search in several hypercubes (along with random sampling to speed up the search). The reduction uses projections onto random lines through the origin. Kleinberg's algorithms are also based on a test using random projections. His test relies on the relative positions of the projected points. In contrast, our test is based on the property that the projection of any vector maintains, in expectation, its (properly scaled) length. This property underlies methods of distance preserving embeddings into low dimensional spaces, like the Johnson–Lindenstrauss lemma [23] (see also Linial, London, and Rabinovich [26]). The problem with these techniques is that when applied directly, they guarantee correct answers to most queries but not to all possible queries. In order to overcome this difficulty, we resort to the theory of Vapnik–Chervonenkis (or VC-) dimension [34] to show the existence of a small finite sample of lines that closely imitate the entire distribution for any vector. A clustering argument and another sampling argument allow us to use this sample to reduce our problem to the cube. Similar ideas work for the L_1 norm too (but we need to project onto the axes rather than onto random lines).

Notation. We denote by n the number of database points, by q the query point, and by ϵ the slackness (i.e., in reply to q we must return a database point whose distance from q is within a factor of $1 + \epsilon$ of the minimum distance from q to any database point).

Metric spaces. We consider the following metric spaces. The d-dimensional Hamming cube Q_d is the set $\{0, 1\}^d$ of cardinality 2^d, endowed with the Hamming distance

H. For $a, b \in Q_d$, $H(a,b) = \sum_i |a_i - b_i|$. For a finite set F, let $x, y \in F^d$. (That is, x, y are words of length d over the alphabet F.) The *generalized Hamming distance* $\mathcal{H}(x,y)$ is the number of dimensions where x differs from y. The Euclidean space ℓ_2^d is $\mathbb{R}^d$ endowed with the standard L_2 distance. The space ℓ_1^d is $\mathbb{R}^d$ endowed with the L_1 distance.

2. Approximate nearest neighbors in the hypercube. In this section we present an approximate nearest neighbors algorithm for the d-dimensional cube. That is, all database points and query points are in $\{0,1\}^d$ and distances are measured by Hamming distance. The first idea behind our algorithm for the hypercube is to design a separate test for each distance ℓ. Given a query q, such a test either returns a database vector at distance at most $(1+\epsilon)\ell$ from q, or informs that there is no database vector at distance ℓ or less from q. Given such a test, we can perform approximate nearest neighbor search by using binary search over $\ell \in \{1, 2, \ldots, d\}$ (and also checking distance 0—this can be done using any reasonable dictionary data structure).

We begin by defining a test, which we later use in the construction of our data structure. A β-test τ is defined as follows. We pick a subset C of coordinates of the cube by choosing each element in $\{1, 2, \ldots, d\}$ independently at random with probability β. For each of the chosen coordinates i we pick independently and uniformly at random $r_i \in \{0,1\}$. For $v \in Q_d$, define the value of τ at v, denoted $\tau(v)$ as follows:

$$\tau(v) = \sum_{i \in C} r_i \cdot v_i \pmod 2.$$

Equivalently, the test can be viewed as picking a vector $\vec{R} \in \{0,1\}^d$ in a way that each entry gets the value 0 with "high" probability (i.e., $1 - \frac{\beta}{2}$) and the value 1 with "low" probability (i.e., $\beta/2$). With this view, the value of the test on $v \in Q_d$ is just its inner product with $\vec{R}$ modulo 2.[1]

Let q be a query, and let a, b be two database points with $H(q,a) \le \ell$ and $H(q,b) > (1+\epsilon)\ell$. We claim that for $\beta = \frac{1}{2\ell}$ the above test distinguishes between a and b with constant probability. More formally, let $\beta = \frac{1}{2\ell}$, and let $\Delta(u,v) = \Pr_{\vec{R}}[\tau(u) \neq \tau(v)]$. Then we have the following lemma.

LEMMA 2.1. *There is an absolute constant $\delta_1 > 0$, such that for any $\epsilon > 0$ there is a constant $\delta_2 > \delta_1$ (depending on ϵ only), such that $\Delta(q,a) \le \delta_1$ and $\Delta(q,b) \ge \delta_2$. (In what follows we denote by δ the constant $\delta_2 - \delta_1$.)*

Proof. For any $u, v \in Q_d$ with $H(u,v) = k$ we have $\Delta(u,v) = \frac{1}{2}(1 - (1 - \frac{1}{2\ell})^k)$ (if none of the k coordinates where $u_i \neq v_i$ is chosen to be in C, then $\tau(u) = \tau(v)$; if at least one such coordinate, j, is in C, then, for every way of fixing all other choices, exactly one of the two choices for r_j will give $\tau(u) \neq \tau(v)$). Note that $\Delta(u,v)$ is monotonically increasing with k. We set $\delta_1 = \frac{1}{2}(1 - (1 - \frac{1}{2\ell})^\ell)$ and $\delta_2 = \frac{1}{2}(1 - (1 - \frac{1}{2\ell})^{(1+\epsilon)\ell})$. Thus $\delta_2 - \delta_1 = \frac{1}{2}\left[(1 - \frac{1}{2\ell})^\ell - (1 - \frac{1}{2\ell})^{(1+\epsilon)\ell}\right] = \Theta(1 - e^{-\epsilon/2})$. ∎

The above lemma implies that, for q, a, and b as above, a single test can get a small (constant) bias towards making the correct decision as to which point is closer to q. To amplify this bias we use several such tests as explained below (see Lemma 2.2).

The data structure. Our data structure $\mathcal{S}$ consists of d substructures $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_d$ (one for each possible distance). Fix $\ell \in \{1, 2, \ldots, d\}$. We now describe $\mathcal{S}_\ell$. Let

[1] It is common to use the inner product for "equality tests." However, these tests just distinguish the equal u, v from the nonequal u, v, but they lose all information on the distance between u and v. In our test, by appropriately choosing the value β, we can obtain some distance information.

M and T be positive integers which we specify later. $\mathcal{S}_\ell$ consists of M structures $\mathcal{T}_1, \ldots, \mathcal{T}_M$. So fix $i \in \{1, 2, \ldots, M\}$. Structure $\mathcal{T}_i$ consists of a list of T $\frac{1}{2\ell}$-tests $\vec{R}_1, \vec{R}_2, \ldots, \vec{R}_T \in \{0,1\}^d$, and a table of 2^T entries (one entry for each possible outcome of the sequence of T tests). Each entry of the table either contains a database point or is empty.

We construct the structure $\mathcal{T}_i$ as follows. We pick independently at random T $\frac{1}{2\ell}$-tests $t_1, \ldots, t_T$ (defined by $\vec{R}_1, \vec{R}_2, \ldots, \vec{R}_T \in \{0,1\}^d$). For $v \in Q_d$, let its *trace* be the vector $t(v) = (t_1(v), \ldots, t_T(v)) \in \{0,1\}^T$. Let δ_1 and δ be the constants from Lemma 2.1. An entry corresponding to $z \in \{0,1\}^T$ contains a database point v with $H(t(v), z) \leq (\delta_1 + \frac{1}{3}\delta)T$, if such a point exists (any such point, if more than one exists), and otherwise the entry is empty. This completes the specification of $\mathcal{S}$ (up to the choice of T and M). Notice that in a straightforward implementation, the size of $\mathcal{S}$ is $O(d \cdot M \cdot (dT + 2^T \log n))$ (we also need to keep the original set of points, and this takes dn space), and it takes $O(d \cdot M \cdot (dTn + 2^T n))$ time to construct $\mathcal{S}$.

LEMMA 2.2. *Let q be a query, and let a, b be two database points with $H(q,a) \leq \ell$ and $H(q,b) > (1+\epsilon)\ell$. Consider a structure $\mathcal{T}_i$ in $\mathcal{S}_\ell$, and let δ_1, δ_2, and δ be as in Lemma 2.1. Then the following hold.*

- $\Pr[H(t(q), t(a)) > (\delta_1 + \frac{1}{3}\delta)T] \leq e^{-\frac{2}{9}\delta^2 T}$.
- $\Pr[H(t(q), t(b)) < (\delta_2 - \frac{1}{3}\delta)T] \leq e^{-\frac{2}{9}\delta^2 T}$.

Proof. The proof follows immediately by plugging in the success probabilities from Lemma 2.1 in the following Chernoff bounds. For a sequence of m independently and identically distributed (i.i.d.) 0-1 random variables $X_1, X_2, \ldots, X_m$, $\Pr[\sum X_i > (p+\gamma)m] \leq e^{-2m\gamma^2}$, and $\Pr[\sum X_i < (p-\gamma)m] \leq e^{-2m\gamma^2}$, where $p = \Pr[X_i = 1]$ (see [2, Appendix A]). ∎

Our goal is to show that we can answer every possible query "correctly." This is formalized by the following definition.

DEFINITION 2.3. *For $q \in Q_d$, ℓ, and $\mathcal{T}_i$ in $\mathcal{S}_\ell$, we say that $\mathcal{T}_i$* fails at *q if there exists a database point a with $H(q,a) \leq \ell$ (or a database point b with $H(q,b) > (1+\epsilon)\ell$), such that $H(t(q), t(a)) > (\delta_1 + \frac{1}{3}\delta)T$ (or $H(t(q), t(b)) < (\delta_2 - \frac{1}{3}\delta)T$, respectively). We say that $\mathcal{S}$* fails at *q if there exists ℓ, such that more than $\mu M / \log d$ structures $\mathcal{T}_i$ in $\mathcal{S}_\ell$ fail (where μ is a constant that affects the search algorithm). We say that $\mathcal{S}$* fails *if there exists $q \in Q_d$ such that $\mathcal{S}$ fails at q.*

The following theorem bounds the probability that $\mathcal{S}$ fails at any given query q.

THEOREM 2.4. *For every $\gamma > 0$, if we set $M = (d + \log d + \log \gamma^{-1}) \log d / \mu$ and $T = \frac{9}{2}\delta^{-2} \ln(2en \log d / \mu)$, then, for any query q, the probability that $\mathcal{S}$ fails at q is at most $\gamma 2^{-d}$.*

Proof. For any ℓ, $\mathcal{T}_i$ in $\mathcal{S}_\ell$ and database point a, the probability that $H(t(q), t(a))$ is not within the desired range (i.e., $\leq (\delta_1 + \frac{1}{3}\delta)T$ if $H(q,a) \leq \ell$ or $\geq (\delta_2 - \frac{1}{3}\delta)T$ if $H(q,a) > (1+\epsilon)\ell$ or anything otherwise) is at most $e^{-\frac{2}{9}\delta^2 T} = \frac{\mu}{2en \log d}$, by Lemma 2.2. Summing over the n database points, the probability that $\mathcal{T}_i$ fails is at most $\frac{\mu}{2e \log d}$. Therefore, the expected number of $\mathcal{T}_i$s that fail is at most $\frac{\mu M}{2e \log d}$. By standard Chernoff bounds (see [2, Appendix A]), for independent 0-1 random variables $X_1, X_2, \ldots, X_m$, setting $X = \sum_i X_i$ and denoting by E the expectation of X, $\Pr[X > (1+\beta)E] < \left(e^\beta / (1+\beta)^{(1+\beta)}\right)^E$. Thus the probability that more than $\frac{\mu M}{\log d}$ of the $\mathcal{T}_i$s fail is less than $2^{-\mu M / \log d} = \gamma / d 2^d$. Summing over all d possible values of ℓ completes the proof. ∎

We conclude the following corollary.

COROLLARY 2.5. *The probability that $\mathcal{S}$ fails is at most γ.*

Proof. Sum the bound from the above theorem over 2^d possible queries q. □

Notice that using the values from Theorem 2.4 for M and T and assuming that γ and μ are absolute constants, we get that the size of $\mathcal{S}$ is $O(\epsilon^{-2}d^3 \log d(\log n + \log\log d) + d^2 \log d(n \log d)^{O(\epsilon^{-2})})$, and the construction time is essentially this quantity times n.

2.1. The search algorithm. Our search algorithm assumes that the construction of $\mathcal{S}$ is successful. By Corollary 2.5, this happens with probability $1-\gamma$. Given a query q, we do a binary search to determine (approximately) the minimum distance ℓ to a database point. A step in the binary search consists of picking one of the structures $\mathcal{T}_i$ in $\mathcal{S}_\ell$ uniformly at random, computing the trace $t(q)$ of the list of tests in $\mathcal{T}_i$, and checking the table entry labeled $t(q)$. The binary search step succeeds if this entry contains a database point, and otherwise it fails. If the step fails, we restrict the search to larger ℓs, and otherwise we restrict the search to smaller ℓs. The search algorithm returns the database point contained in the last nonempty entry visited during the binary search.

LEMMA 2.6. *For any query q, the probability that the binary search uses a structure $\mathcal{T}_i$ that fails at q is at most μ.*

Proof. The binary search consists of $\log d$ steps, each examining a different value ℓ. As we are assuming that $\mathcal{S}$ did not fail, the probability that for any given ℓ the random $\mathcal{T}_i$ in $\mathcal{S}_\ell$ that we pick fails is at most $\mu/\log d$. Summing over the $\log d$ steps completes the proof. □

LEMMA 2.7. *If all the structures used by the binary search do not fail at q, then the distance from q to the database point a returned by the search algorithm is within a $(1+\epsilon)$-factor of the minimum distance from q to any database point.*

Proof. Denote the minimum distance from q to any database point by $\ell_{\min}$. If $\ell < \ell_{\min}/(1+\epsilon)$, then no database point is within distance $(1+\epsilon)\ell$ of q, and therefore all the binary search steps that visit ℓ in this range fail. On the other hand, all the binary search steps that visit ℓ in the range $\ell \geq \ell_{\min}$ succeed. Therefore, the binary search ends with ℓ such that $\ell_{\min}/(1+\epsilon) \leq \ell \leq \ell_{\min}$. At that point, the database point a returned has $H(t(q), t(a)) \leq (\delta_1 + \frac{1}{3}\delta)T$. Any database point b with $H(q,b) > (1+\epsilon)\ell$ has $H(t(q), t(b)) > (\delta_2 - \frac{1}{3}\delta)T > (\delta_1 + \frac{1}{3}\delta)T$. Therefore, $H(q,a) \leq (1+\epsilon)\ell \leq (1+\epsilon)\ell_{\min}$. □

Lemmas 2.6 and 2.7 imply the main result of this section which is the following theorem.

THEOREM 2.8. *If $\mathcal{S}$ does not fail, then for every query q the search algorithm finds a $(1+\epsilon)$-approximate nearest neighbor with probability at least $1-\mu$ using $O(\epsilon^{-2}d(\log n + \log\log d + \log\frac{1}{\mu})\log d)$ arithmetic operations.*

Proof. The success probability claim is immediate from the above lemmas. The number of operations follows from the fact that we perform $\log d$ binary search steps. Each step requires computing the value of T β-tests. Each β-test requires computing the sum of at most d products of two elements. □

Remark. Some improvements of the above implementation are possible. For example, note that the value of M was chosen so as to guarantee that with constant probability no mistake is made throughout the binary search. Using results of [17], a binary search can still be made in $O(\log d)$ steps even if there is a constant probability of error at each step. This allows choosing M which is smaller by an $O(\log d)$ factor and getting the corresponding improvement in the size of $\mathcal{S}$ and the time required to construct it.

3. Approximate nearest neighbors in Euclidean spaces. In this section we present our algorithm for Euclidean spaces. The main idea underlying the solution for Euclidean spaces is to reduce the problem to the problem on the cube solved in the previous section. In fact, we produce several cubes, and the search involves several cube searches.

Notation. Let $x \in \mathbb{R}^d$, and let $\ell > 0$. Denote by $\mathcal{B}(x, \ell)$ the closed ball around x with radius ℓ; i.e., the set $\{y \in \mathbb{R}^d \mid \|x - y\|_2 \leq \ell\}$. Denote by $\mathcal{D}(x, \ell)$ the set of database points contained in $\mathcal{B}(x, \ell)$.

The main tool in reducing the search problem in Euclidean space to search problems in the cube is the following embedding lemma. The proof of this lemma follows standard arguments. We defer the proof to the appendix.

LEMMA 3.1. *There exists a constant $\lambda > 0$, such that for every $\delta > 0$, $\beta > 0$, $\ell > 0$, positive integer d, and $x \in \mathbb{R}^d$, the following holds. There is an embedding $\eta = \eta(x, \ell, \delta, \beta)$, $\eta : \mathbb{R}^d \hookrightarrow Q_k$ with the following properties.*

1. $$k = \text{poly}\left(\delta^{-1}\right) \cdot \left(d \log^2 d + d \log d \log \delta^{-1} + \log \beta^{-1}\right).$$

2. *For every $y \in \mathcal{B}(x, \ell)$, and for every $z \in \mathbb{R}^d$,*

$$((1 - O(\delta))m - O(\delta))k \leq H(\eta(y), \eta(z)) \leq ((1 + O(\delta))m' + O(\delta))k,$$

where

$$m = \kappa \cdot \max\left\{\frac{\|y - z\|_2}{\ell}, \lambda\right\},$$

$$m' = \kappa \cdot \min\left\{\frac{\|y - z\|_2}{\ell}, \delta^{-1}\right\},$$

$$\kappa = \Theta\left(1/(1 + \delta^{-1})\sqrt{\log(\delta^{-1})}\right).$$

Furthermore, there is a probabilistic algorithm that computes in polynomial time, with success probability at least $1 - \beta$, an embedding η with the above properties. The algorithm computes a representation of η as $O(dk)$ rational numbers. Using this representation, for every rational $y \in \mathbb{R}^d$, we can compute $\eta(y)$ using $O(dk)$ arithmetic operations.

The data structure. Our data structure $\mathcal{S}$ consists of a substructure $\mathcal{S}_a$ for every point a in the database. Each $\mathcal{S}_a$ consists of a list of all the other database points, sorted by increasing distance from a, and a structure $\mathcal{S}_{a,b}$ for each database point $b \neq a$. Fix a and b, and let $\ell = \|a - b\|_2$ be the distance from a to b. (For simplicity, we'll assume that different bs have different distances from a.) The structure $\mathcal{S}_{a,b}$ consists of (1) a representation of an embedding η (as in Lemma 3.1), (2) a Hamming cube data structure, and (3) a positive integer.

We construct $\mathcal{S}_{a,b}$ as follows. Set $\delta = \epsilon/O(1)$. Part (1) of $\mathcal{S}_{a,b}$ is a representation of $\eta(a, \ell, \delta, \beta)$ (β to be determined below), which we compute by Lemma 3.1. Let k be the dimension of the target of η. Part (2) of $\mathcal{S}_{a,b}$ is an approximate nearest neighbor search structure for Q_k, with the database consisting of the images under η of the points in $\mathcal{D}(a, \ell)$, and the slackness parameter being δ. Part (3) of $\mathcal{S}_{a,b}$ is the number of database points in $\mathcal{D}(a, \ell)$. This completes the specification of $\mathcal{S}_{a,b}$ (up to the choice of β, and of the error probability γ allowed in the cube data structure construction).

DEFINITION 3.2. *We say that* $\mathcal{S}_{a,b}$ fails *if the embedding* η *does not satisfy the properties stipulated in Lemma* 3.1*, or if the construction of the cube data structure fails. We say that* $\mathcal{S}$ fails *if there are* a, b *such that* $\mathcal{S}_{a,b}$ *fails.*

LEMMA 3.3. *For every* $\zeta > 0$*, setting* $\beta = \zeta/n^2$ *and* $\gamma = \zeta/n^2$ *(where* β *is the parameter of Lemma* A.3*, and* γ *is the parameter of Corollary* 2.5*) the probability that* $\mathcal{S}$ *fails is at most* ζ*.*

Proof. Sum up the failure probabilities from Lemma 3.1 and Corollary 2.5 over all the structures we construct. □

Our data structure $\mathcal{S}$ requires $O(n^2 \cdot \text{poly}(1/\epsilon) \cdot d^2 \cdot \text{poly}\log(dn/\epsilon) \cdot (n \log(d \log n/\epsilon))^{O(\epsilon^{-2})})$ space (we have $O(n^2)$ structures, and the dominant part of each is the k-dimensional cube structure). The time to construct $\mathcal{S}$ is essentially its size times n (again, the dominant part is constructing the cube structures).

3.1. The search algorithm. As with the cube, our search algorithm assumes that the construction of $\mathcal{S}$ succeeds. This happens with probability at least $1 - \zeta$, according to Lemma 3.3. Given a query q, we search some of the structures $\mathcal{S}_{a,b}$ as follows. We begin with any structure $\mathcal{S}_{a_0,b_0}$, where a_0 is a database point and b_0 is the farthest database point from a_0. Let $\ell_0 = \|a_0 - b_0\|_2$. Then $\mathcal{D}(a_0, \ell_0)$ contains the entire database. We proceed by searching $\mathcal{S}_{a_1,b_1}$, $\mathcal{S}_{a_2,b_2}$, ..., where a_{j+1}, b_{j+1} are determined by the results of the search in $\mathcal{S}_{a_j,b_j}$.

So fix j. We describe the search in $\mathcal{S}_{a_j,b_j}$. Let $\ell_j = \|a_j - b_j\|_2$. Let η be the embedding stored in $\mathcal{S}_{a_j,b_j}$. We compute $\eta(q)$, a node of the k-dimensional cube. We now search for a $(1+\delta)$-approximate nearest neighbor for $\eta(q)$ in the cube data structure stored in $\mathcal{S}_{a_j,b_j}$ (allowing failure probability μ). Let the output of this search be (the image of) the database point a'. If $\|q - a'\|_2 > \frac{1}{10}\ell_{j-1}$, we stop and output a'. Otherwise, we pick T database points uniformly at random from $\mathcal{D}(a_j, \ell_j)$, where T is a constant. Let a'' be the closest among these points to q. Let a_{j+1} be the closest to q between a_j, a' and a'', and let b_{j+1} be the farthest from a_{j+1} such that $\|a_{j+1} - b_{j+1}\|_2 \le 2\|a_{j+1} - q\|_2$. (We find b_{j+1} using binary search on the sorted list of database points in $\mathcal{S}_{a_{j+1}}$.) If no such point exists, we abort the search and we output a_{j+1}.

Before going into the detailed analysis of the search algorithm, let us try to motivate it. Our test gives a good approximation for the distance if ℓ is "close" to the true minimum distance between q and a database point. Thus, ℓ can be viewed as the scale with which we measure distances. If the scale is too large, we cannot make the right decision. However, we are able to detect that ℓ is too large, and in such a case we reduce it. This guarantees that if we start with ℓ_0 and the nearest neighbor is at distance $\ell_{\min}$, the search will terminate in $O(\log \frac{\ell_0}{\ell_{\min}})$ iterations. This quantity may be enormous compared with d and $\log n$. To speed up the search (i.e., have the number of iterations independent of the ratio of distances), we add random sampling from the points $\mathcal{D}(a_j, \ell_j)$. Using random sampling guarantees that not only the distances reduce but also that the number of database points to consider decreases quickly. This guarantees that the number of iterations is $O(\log n)$.

The following lemmas formulate the progress made by each step. For the analysis, let $a_{\min}$ be the closest point in the database to q and let $\ell_{\min}$ be its distance.

LEMMA 3.4. *For every* $j \ge 0$*,* $a_{\min} \in \mathcal{D}(a_j, \ell_j)$*.*

Proof. $\mathcal{D}(a_j, \ell_j)$ contains all the database points whose distance from a_j is at most $2\|q - a_j\|_2$. In particular (by triangle inequality), it contains all the database points whose distance from q is at most $\|q - a_j\|_2 \ge \ell_{\min}$. Therefore, it contains $a_{\min}$. □

LEMMA 3.5. *For every $j \geq 1$, if ℓ_j is such that $\delta^{-1}\ell_j < \ell_{\min}$, then for every $a \in \mathcal{D}(a_j, \ell_j)$ we have $\|q - a\|_2 \leq \ell_{\min}(1+\delta)$.*

Proof. By the assumptions (and since we have $a_{\min} \in \mathcal{D}(a_j, \ell_j)$), the distance from q to a is at most $\ell_{\min} + \ell_j < \ell_{\min}(1+\delta)$. □

LEMMA 3.6. *For every $j \geq 1$, $\|q - a'\|_2 \leq \max\{(1+\epsilon)\ell_{\min}, \frac{1}{10}\ell_{j-1}\}$ with probability at least $1 - \mu$.*

Proof. As $a_{\min} \in \mathcal{D}(a_{j-1}, \ell_{j-1})$, we claim that our search of $\mathcal{S}_{a_{j-1}, b_{j-1}}$ returns a' whose distance from q is at most $(1+\epsilon)\max\{\ell_{\min}, \lambda\ell_{j-1}\}$ with probability at least $1 - \mu$ (we set λ such that $(1+\epsilon)\lambda \leq 1/10$). The reason is that, by Lemma 3.1,

$$H(\eta(q), \eta(a)) \leq ((1 + O(\delta))\kappa \max\{\ell_{\min}/\ell_{j-1}, \lambda\} + O(\delta))k.$$

Therefore, the search algorithm returns a point b_c such that

$$H(\eta(q), b_c) \leq (1+\delta)((1 + O(\delta))\kappa \max\{\ell_{\min}/\ell_{j-1}, \lambda\} + O(\delta))k.$$

Using Lemma 3.1 again, this point b_c is the image of a point a' whose distance from q satisfies

$$\|q - a'\|_2 \leq (1 + O(\delta)) \max\{\ell_{\min}, \lambda\ell_{j-1}\} + O(\delta)\ell_{j-1}. \qquad \square$$

LEMMA 3.7. *For every $j \geq 1$, $\mathcal{B}(q, \|q - a_j\|_2)$ contains at most $\frac{1}{2}|\mathcal{D}(a_{j-1}, \ell_{j-1})|$ database points with probability at least $1 - 2^{-T}$.*

Proof. First notice that $\mathcal{B}(q, \|q - a_j\|_2)$ contains database points from $\mathcal{D}(a_{j-1}, \ell_{j-1})$ only. Let ξ be such that $\mathcal{B}(q, \xi)$ contains exactly half the points of $\mathcal{D}(a_{j-1}, \ell_{j-1})$. (For simplicity, we assume that the distances from q to the database points are all distinct.) Each database point in the random sample we pick has probability $\frac{1}{2}$ to be in $\mathcal{B}(q, \xi)$. Therefore, the probability that $a'' \notin \mathcal{B}(q, \xi)$ is at most 2^{-T}. □

LEMMA 3.8. *For all $j \geq 1$, $\mathcal{D}(a_j, \ell_j) \subset \mathcal{B}(q, \|q - a_{j-1}\|_2)$ with probability at least $1 - \mu$.*

Proof. Let $a \in \mathcal{D}(a_j, \ell_j)$. By the triangle inequality, $\|q - a\|_2 \leq \ell_j + \|q - a_j\|_2 \leq 3\|q - a_j\|_2$. By Lemma 3.6, $3\|q - a_j\|_2 \leq \frac{3}{10}\ell_{j-1}$. Since $\ell_{j-1} \leq 2\|q - a_{j-1}\|_2$, the lemma follows. □

COROLLARY 3.9. *For every $j \geq 2$, $|\mathcal{D}(a_j, \ell_j)| \leq \frac{1}{2}|\mathcal{D}(a_{j-2}, \ell_{j-2})|$ with probability at least $1 - (\mu + 2^{-T})$.*

THEOREM 3.10. *If $\mathcal{S}$ does not fail, then for every query q the search algorithm finds a $(1+\epsilon)$-approximate nearest neighbor using expected* $\mathrm{poly}(1/\epsilon)d^2\mathrm{poly}\log(dn/\epsilon)$ *arithmetic operations.*[2]

Proof. Corollary 3.9 says that within two iterations of the algorithm, with constant probability the number of database points in the current ball, $\mathcal{D}(a_j, \ell_j)$, is reduced by a factor of 2. Hence, within expected $O(\log n)$ iterations the search ends.

If the search ends because $\|q - a'\|_2 > \frac{1}{10}\ell_{j-1}$, then by Lemma 3.6 it must be that $\|q - a'\|_2 \leq (1+\epsilon)\ell_{\min}$. Otherwise, the search ends because no database point $b \neq a_j$ satisfies: $\|a_j - b\|_2 \leq 2\|a_j - q\|_2$. In this case, $a_j = a_{\min}$, because $\|a_j - a_{\min}\|_2 \leq \|a_j - q\|_2 + \|a_{\min} - q\|_2 \leq 2\|a_j - q\|_2$. In either case, the search produces a $(1+\epsilon)$-approximate nearest neighbor.

As for the search time, we have $O(\log n)$ iterations. In each iteration we perform $O(dk) = O(\mathrm{poly}(1/\epsilon) \cdot d^2 \cdot \mathrm{poly}\log(dn/\epsilon))$ operations to compute $\eta(q)$; then, we execute a search in the k-cube, which by Theorem 2.8 takes $O(\mathrm{poly}(1/\epsilon) \cdot d \cdot \mathrm{poly}\log(dn/\epsilon))$ operations. □

[2] Alternatively, we can demand a deterministic bound on the number of operations, if we are willing to tolerate a vanishing probability error in the search.

4. Extensions. In what follows we discuss some other metrics for which our methods (with small variations) apply.

Generalized Hamming metric. Assume that we have a finite alphabet Σ and consider the generalized cube Σ^d. For $x, y \in \Sigma^d$, the *generalized Hamming distance* $\mathcal{H}(x, y)$ is the number of dimensions where x differs from y. The case $\Sigma = \{0, 1\}$ is the Hamming cube discussed in section 2. Here we argue that the results in that section extend to the case of arbitrary Σ. For convenience, assume that $\Sigma = \{0, 1, \ldots, p-1\}$ for a prime p. (We can always map the elements of Σ to such a set, perhaps somewhat larger, without changing the distances.) It suffices to show that a generalization of the basic test used in section 2 has the same properties. The generalized test works as follows: pick each element in $\{1, 2, \ldots, d\}$ independently at random with probability $1/(2\ell)$. For each chosen coordinate i, pick independently and uniformly at random $r_i \in \Sigma$. (For every i which is not chosen put $r_i = 0$.) The test is defined by $\tau(x) = \sum_{i=1}^{d} r_i x_i$, where multiplication and summation are done in $GF[p]$. As before, for any two vectors $x, y \in \Sigma^d$, let $\Delta(x, y) \stackrel{\triangle}{=} \Pr[\tau(x) \neq \tau(y)]$. If $\mathcal{H}(x, y) = k$, then $\Delta(x, y) = \frac{p-1}{p} \cdot (1 - (1 - \frac{1}{2\ell})^k)$. Therefore, the difference δ in the probabilities between the case of vectors with Hamming distance at most ℓ and the case of vectors with Hamming distance at least $(1+\epsilon)\ell$ is $\frac{p-1}{p} \cdot \left[(1 - \frac{1}{2\ell})^{\ell} - (1 - \frac{1}{2\ell})^{(1+\epsilon)\ell}\right]$. δ is minimized at $p = 2$, so we do better with a larger alphabet. Notice that the number of possible traces here is p^T, so this construction gives a polynomial size data structure if and only if p is a constant. In the next paragraph we mention how to handle nonconstant p-s.

L_1 norm for finite alphabet. Consider, again, the case $\Sigma = \{0, 1, \ldots, p-1\}$ and define the distance between x and y as $d(x, y) \stackrel{\triangle}{=} \sum_{i=1}^{d} |x_i - y_i|$. The first observation is that this case is reducible to the case of the Hamming cube as follows. Map any $x \in \Sigma^d$ into $\hat{x} \in \{0, 1\}^{d(p-1)}$ by replacing every coordinate $x_i \in \{0, 1, \ldots, p-1\}$ of x by $p-1$ coordinates of $\hat{x}$. These are x_i ones followed by $(p - 1 - x_i)$ zeros. Observe that indeed $d(x, y) = H(\hat{x}, \hat{y})$. Therefore, we can apply the cube construction and algorithm to $\hat{x}$. If p is not a constant, this solution is not satisfactory, because it blows up not only the data structure size (by a factor polynomial in p), but also the search time (by a factor of p at least). Our second observation is that we can restrict the blowup in search time to a factor of $O(\log p)$. This is because we do not really have to map x into $\hat{x}$. The contribution of $x_i \in \{0, 1, \ldots, p-1\}$ to $\tau(\hat{x})$ is just the sum modulo 2 of (at most $p-1$) r_j-s. As there are only p possible sums (and *not* 2^p) they can all be precomputed and stored in a dictionary using $O(p)$ space. (Notice that this needs to be done for each test and for each coordinate.) To compute the value of the test on a query, we need to sum up the contributions of the coordinates. For each coordinate, it takes at most $O(\log p)$ time to retrieve the desired value (because operations on values in $\{0, 1, \ldots, p-1\}$ take that much time). The same ideas can be used to handle the generalized Hamming metric for nonconstant alphabets. Map each coordinate x_i into p binary coordinates, which are all 0s except for a 1 in position $x_i + 1$. The Hamming distance in the (dp)-cube is twice the original distance. For a given test, there are only p different values to consider for each original coordinate, so we can precompute them and retrieve them as before.

The space ℓ_1^d. The construction and algorithm are similar to those for the Euclidean case. The only difference is in the embedding η to the cube. Instead of projecting the points onto random unit vectors, we project them onto the original coordinates. Let w, δ, λ, ℓ be as in the Euclidean case. For the ith coordinate, we place $S = 2d(1 + 1/\delta)/\delta\lambda$ equally spaced points between $w_i - (1 + 1/\delta)\ell$ and $w_i + (1 + 1/\delta)\ell$.

These partition the line into $S+1$ (finite or infinite) segments. We number them 0 through S from left to right. Now for every $x \in \mathbb{R}^d$ we define $\eta(x)$ to be the vector with entries in $\{0, 1, \ldots, S\}$, such that $\eta(x)_i$ is the number of the segment that contains x_i. Using Lemma A.4, we show the equivalent of Lemma A.5 for ℓ_1^d.

LEMMA 4.1. *For every x such that $\|x-w\|_1 \le \ell$, for every $y \in \mathbb{R}^d$,*

$$(1-\delta)m\frac{d}{\delta\lambda} \le \|\eta(x)-\eta(y)\|_1 \le (1+\delta)m'\frac{d}{\delta\lambda},$$

where $m = \max\left\{\frac{\|x-y\|_1}{\ell}, \lambda\right\}$, *and* $m' = \min\left\{\frac{\|x-y\|_1}{\ell}, \delta^{-1}\right\}$.

Proof. We show the case $\lambda\ell \le \|x-y\|_1 \le \ell/\delta$. The other two cases are similar. Notice that in this case, for every i, $x_i, y_i \in [w_i - (1+\delta^{-1})\ell, w_i + (1+\delta-1)\ell]$. By Lemma A.4, for every i,

$$-1 + \|x_i - y_i\|S/2(1+\delta^{-1})\ell \le \|\eta(x)_i - \eta(y)_i\| \le 1 + \|x_i - y_i\|S/2(1+\delta^{-1})\ell.$$

Thus, summing over the d coordinates,

$$-d + \|x-y\|_1 S/2(1+\delta^{-1})\ell \le \|\eta(x)-\eta(y)\|_1 \le d + \|x-y\|_1 S/2(1+\delta^{-1})\ell.$$

As $d = \delta\lambda S/2(1+\delta^{-1}) \le \delta\|x-y\|_1 S/2(1+\delta^{-1})\ell$, the lemma follows. □

We use the construction for the finite alphabet L_1 norm to handle the embedded instance. The remainder of the argument is identical to the Euclidean case. Notice, however, that given a query q, computing $\eta(q)$ is more efficient than in the Euclidean case: In each coordinate, we can use a binary search to determine the segment in $O(\log S) = O(\log(d/\epsilon))$ operations. Projecting is trivial and takes $O(1)$ operations per coordinate. In the Euclidean case, the search time is dominated by the calculation of $\eta(q)$. Thus, here it goes down to $O(d\text{poly}\log(dn/\epsilon))$.

Further comments. The iterative search procedure described in section 3.1 is quite general and can be used in any metric space; i.e., the problem of finding an approximate nearest neighbor reduces to the following problem. Given a query q and a distance estimate ℓ, either return an approximate nearest neighbor of q, or return a data point at distance at most $\ell/10$ from q. Of course, the latter problem might be hard to solve in an arbitrary metric space.

Appendix. Proof of Lemma 3.1. The proof of the embedding lemma follows two parts. First, we use a low distortion embedding of ℓ_2^d into ℓ_1^k (where, for fixed ϵ, $k = O(d\log^2 d + \log n)$). Second, we use an embedding of ℓ_1^k into $Q_{O(k)}$. This embedding maintains low distortion on certain distances, as stipulated by the lemma.

It is known (see, for example, [15] and references therein) that the projection ℓ_2^d onto $O(d)$ random unit vectors, properly scaled, gives a low distortion embedding of ℓ_2^d into $\ell_1^{O(d)}$ (the distortion $1+\epsilon$ drops to 1 as the constant hidden by the big-Oh notation grows). Using the arguments in section 4 (which in turn use the second part of the proof here), we could prove a version of the embedding lemma. The dimension of the cube would have to grow by a factor of $O(d/\log^2 d)$. Thus the size of the data structure would grow significantly (yet the search time would improve by a factor of $O(\log d)$).

To avoid this blowup in space, and because we require explicit bounds on the dependency on ϵ and on the error probability, we give here another argument for the low distortion of embedding ℓ_2^d into ℓ_1^k via random projections. The dimesion k of the target space is somewhat worse than in [15]. However, the details of the analysis allow

us to save significantly on the embedding into the cube. (In fact, it is likely that our analysis of the dimension can be improved by a factor of $O(\log d)$, thus matching the search time required using [15].)

It is also known (see, for example, [26]) that for a finite set P of points in ℓ_1^d, rounding each point to a grid point, and then representing each coordinate in unary (padded by leading zeros), gives an embedding of P into a hypercube that approximately preserves relative distances (the approximation depends on the fineness of the grid). We argue here (a straightforward argument) that such a construction approximately preserves relative distances among an infinite number of point pairs (as required by the embedding lemma). We also argue that for these pairs the grid need not be too fine, so the dimension of the cube is small. (Notice that distances in Q_k vary by a factor of k at most. On the other hand, in an arbitrary finite set P of points in ℓ_1^d the distances can vary by an arbitrarily large factor. Thus, it is impossible to bound the dimension of the cube uniformly for all sets P.)

We now proceed with the proof. Fix x and ℓ. Let D, S, and L be parameters that we fix later (we will set $k = DS$). We map the points in $\mathcal{D}(x, \ell)$ into the $(D \cdot S)$-dimensional cube as follows: We pick a set of D i.i.d. unit vectors $\{z_1, \ldots, z_D\}$ from the uniform (Haar) measure on the unit sphere and project the points in $\mathcal{D}(x, \ell)$ onto each of these vectors; i.e., for every $a \in \mathcal{D}(x, \ell)$ and for every $z \in \{z_1, \ldots, z_D\}$ we compute the dot product $a \cdot z$. For every $z \in \{z_1, \ldots, z_D\}$ we place S equally spaced points in the interval $[x \cdot z - L, x \cdot z + L]$. We call these points *cutting points*. Each vector z and cutting point c determine a single coordinate of the cube. For any $a \in \mathcal{D}(x, \ell)$, the value of this coordinate is 0 if $a \cdot z \leq c$, and it is 1 otherwise.[3] Altogether, we get a mapping of x into a point in the cube $\{0, 1\}^{D \cdot S}$.

The following lemma analyzes the distribution of length when projecting a fixed (unit) vector on a random (unit) vector.

LEMMA A.1. *Let X be the length of the projection of a unit vector onto a random unit vector drawn from the uniform measure on the unit sphere. Then, for every $\delta > 0$ there exist $\alpha = \Theta(\delta)$ and $\alpha' = \Theta(\delta^{3/2})$ such that the following hold.*

1. $$\alpha_0 \stackrel{\triangle}{=} \Pr\left[X < \sqrt{\frac{\delta}{d}}\right] < \alpha;$$

2. $$\alpha_\infty \stackrel{\triangle}{=} \Pr\left[X > \sqrt{\frac{\log(1/\delta)}{d}}\right] < \alpha;$$

3. *For every $j \geq 1$ such that $(1+\delta)^j \sqrt{\delta/d} \leq \sqrt{\log(\delta)^{-1}/d}$,*

$$\alpha_j \stackrel{\triangle}{=} \Pr\left[(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}} \leq X \leq (1+\delta)^j \sqrt{\frac{\delta}{d}}\right] \geq \alpha'.$$

Proof. Let $\mathcal{S}_d(r)$ denote the sphere of radius r in $\mathbb{R}^d$ centered at the origin. Its surface area, $\mathcal{A}_d(r)$, is given by $\mathcal{A}_d(r) = 2\pi^{d/2} r^{d-1}/\Gamma(d/2) = \mathcal{A}_d(1) r^{d-1}$, where Γ is

[3] Note that if the cutting points are $c_1 \leq c_2 \leq \cdots c_S$, then the S coordinates obtained for a point a by comparing $a \cdot z$ to the cutting points are always of the following form: j 0s followed by $S - j$ 1s. In other words, only $S + 1$ out of the 2^S combinations of 0s and 1s are possible. This observation can be used to get certain improvements in the efficiency of our algorithm. See section 4 for details.

the so-called Gamma function.[4] By "rotating" the space we can view the experiment of projecting a fixed vector on a random vector as if we project a random vector on the axis $x_1 = 1$. Therefore, the probabilities that we need to estimate are just "slices" of the sphere. In particular, consider the set of points $\{x \in \mathcal{S}_d(1) \mid x_1 \in (\tau - \omega, \tau)\}$ (with $\omega, \tau - \omega > 0$). The surface area of this set is lower bounded by $\omega \cdot \mathcal{A}_{d-1}(r)$, where $r = \sqrt{1-\tau^2}$. By symmetry, the same is true for $\{x \in \mathcal{S}_d(1) \mid x_i \in (-\tau, -\tau + \omega)\}$.

To compute the probability of the desired event we compare the area of the slice with the area of the whole sphere. Note that $\mathcal{A}_{d-1}(1)/\mathcal{A}_d(1) = \Theta(\sqrt{d})$. Plug in $\tau = \tau(j) = (1+\delta)^j\sqrt{\delta/d}$ and $\omega = \omega(j) = \tau(j) - \tau(j-1) = \delta(1+\delta)^{j-1}\sqrt{\delta/d}$. Put $\xi = \xi(j) = \delta(1+\delta)^{2j}$; thus $r^2 = 1 - \tau^2 = 1 - \xi/d$ and $\omega = \delta\sqrt{\xi/d}/(1+\delta)$. We get that

$$\begin{aligned}
\Pr\left[(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}} \le X \le (1+\delta)^j\sqrt{\frac{\delta}{d}}\right] &\ge 2\omega\mathcal{A}_{d-1}(\sqrt{1-\tau^2})/\mathcal{A}_d(1) \\
&= 2\omega\mathcal{A}_{d-1}(1)\cdot(\sqrt{1-\tau^2})^{d-2}/\mathcal{A}_d(1) \\
&= \Theta\left(\frac{\delta}{1+\delta}\sqrt{\xi}\left(1-\frac{\xi}{d}\right)^{\frac{d-2}{2}}\right) \\
&= \Omega(\delta^{3/2}),
\end{aligned}$$

where the last equality follows from the fact that in the range of j that interests us, $1 \le (1+\delta)^{j-1} < (1+\delta)^j \le \sqrt{\delta^{-1}\log(1/\delta)}$. This shows the third claim. Similar arguments show the first two claims. □

COROLLARY A.2. *Using the above notation, there is an absolute constant b such that*

$$\sum_{j=1}^{j_{\max}}\left(\alpha_j(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\right) \le E[X] \le b\sqrt{\frac{\delta}{d}} + \sum_{j=0}^{j_{\max}}\left(\alpha_j(1+\delta)^j\sqrt{\frac{\delta}{d}}\right). \qquad \square$$

In what follows, we denote by b' the constant $b' = E[X]\sqrt{d}$.

The next lemma analyzes the lengths distribution with respect to a series of D projections.

LEMMA A.3. *Let δ, α, α' be as in Lemma A.1. Let $\varphi, \beta > 0$. Set*

$$D = \frac{c}{\varphi^2}(8(d+1)\log(4(d+1))(\log(8(d+1)) + \log\log(4(d+1)) + \log\varphi^{-1}) + \log\beta^{-1})$$

for some absolute constant c. Let $z_1, \ldots, z_D$ be i.i.d. unit vectors from the uniform distribution on the unit sphere. Then, with probability at least $1-\beta$, the following holds. For every $x, y \in \mathbb{R}^d$ define

$$I_0 = \left\{i;\ |(x-y)\cdot z_i| < \sqrt{\frac{\delta}{d}}\|x-y\|_2\right\},$$

$$I_\infty = \left\{i;\ |(x-y)\cdot z_i| > \sqrt{\frac{\log(1/\delta)}{d}}\|x-y\|_2\right\},$$

[4] For integer d the Gamma function is given by

$$\Gamma(d/2) \triangleq \begin{cases} (\frac{d-2}{2})! & d \text{ even}, \\ \frac{(d-2)(d-4)\cdots 1}{2^{(d-1)/2}}\sqrt{\pi} & d \text{ odd}. \end{cases}$$

$$I_j = \left\{ i;\ (1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2 \le |(x-y)\cdot z_i| \le (1+\delta)^{j}\sqrt{\frac{\delta}{d}}\|x-y\|_2 \right\},$$

where $j = 1, 2, \ldots, j_{\max}$, *with* $j_{\max}$ *the largest possible such that* $I_{j_{\max}} \cap I_\infty = \emptyset$.[5] *Then*

1. $|I_0|, |I_\infty| < (\alpha + \varphi)D$; *and*
2. *for* $j = 1, 2, \ldots, j_{\max}$, $\quad (\alpha_j - \varphi)D \le |I_j| \le (\alpha_j + \varphi)D$.

Proof. Consider the following range space over the set of vectors in the unit sphere. Every pair of points $x, y \in \mathbb{R}^d$ defines several ranges: a range of vectors z such that $|(x-y)\cdot z| < \sqrt{\delta/d}\|x-y\|_2$, a range such that $|(x-y)\cdot z_i| > \sqrt{\log(1/\delta)/d}\|x-y\|_2$, and ranges such that $(1+\delta)^{j-1}\sqrt{\delta/d}\|x-y\|_2 \le |(x-y)\cdot z_i| \le (1+\delta)^{j}\sqrt{\delta/d}\|x-y\|_2$ for $j = 1, 2, \ldots, j_{\max}$. Each of these ranges is a Boolean combination of at most four (closed or open) half-spaces. Therefore, the VC-dimension of this range space is at most $8(d+1)\log(4(d+1))$ (see [2]). The lemma follows from the fact that a random subset of the unit sphere of size D is a φ-sample with probability at least $1-\beta$. ∎

LEMMA A.4. *Let* $L, \psi > 0$. *Let* $\sigma = [-L, L]$ *be a segment of the real line. Set* $S = \lceil \frac{1}{\psi} \rceil$. *Let* $-L = p_1 < p_2 < \cdots < p_S = L$ *be equally spaced points in* σ (*i.e.,* $p_j = -L + 2L(j-1)/(S-1)$). *Then, every segment* $[\sigma_1, \sigma_2) \subset \sigma$ *contains at least* $(-\psi + (\sigma_2 - \sigma_1)/2L)S$ *such points and at most* $(\psi + (\sigma_2 - \sigma_1)/2L)S$ *such points.*

Proof. The number of points in $[\sigma_1, \sigma_2)$ is approximately proportional to the measure of this segment (under the uniform measure on σ). It might be at worst one point below or one point above the exact proportion. ∎

Let $w \in \mathbb{R}^d$ and let $\ell > 0$. Fix L, φ, β, ψ. (Thus D and S are fixed.) Consider the following (random) embedding $\eta : \mathbb{R}^d \hookrightarrow Q_{DS}$: Let $z_1, \ldots, z_D$ be the random vectors in Lemma A.3, and Let $p_1, \ldots, p_S$ be the points in Lemma A.4. For $x \in \mathbb{R}^d$, $\eta(x) = \eta(x)_{11}\eta(x)_{12} \cdots \eta(x)_{ij} \cdots \eta(x)_{DS}$, where $\eta(x)_{ij} = 0$ if $(x-w)\cdot z_j \le p_i$, and $\eta(x)_{ij} = 1$ otherwise. We are now ready to restate and prove the embedding lemma.

LEMMA A.5. *Let* $\lambda > 0$ *be a sufficiently small constant. Set*

$$L = (1+\delta^{-1})\ \ell\sqrt{\log(1/\delta)/d}.$$

Set $\varphi = \delta\alpha'$ *and* $\psi = \delta^2\lambda/2(1+\delta^{-1})$.

Then, for η *the following holds with probability at least* $1-\beta$. *For every* $x \in B(w, \ell) \subseteq \mathbb{R}^d$ *and* $y \in \mathbb{R}^d$

$$((1-O(\delta))m - O(\delta))DS \le H(\eta(x), \eta(y)) \le ((1+O(\delta))m' + O(\delta))DS,$$

where

$$m = \kappa \cdot \max\left\{ \frac{\|x-y\|_2}{\ell}, \lambda \right\},$$

$$m' = \kappa \cdot \min\left\{ \frac{\|x-y\|_2}{\ell}, \delta^{-1} \right\},$$

and $\kappa = b'/2(1+\delta^{-1})\sqrt{\log(1/\delta)}$.

Proof. We prove the lemma for the case $\lambda\ell \le \|x-y\|_2 \le \ell/\delta$. The proofs of the two extreme cases are similar. To analyze the distance in the cube, $H(\eta(x), \eta(y))$, we notice that this distance is influenced by the distribution of the projection lengths

[5] For simplicity we assume that these sets form a partition of the space; otherwise, there are minor changes in the constants.

$|(x-y)\cdot z_1|,\ldots,|(x-y)\cdot z_D|$ among the sets I_j (Lemma A.3 guarantees that this distribution is "nice"); the error in estimating $|(x-y)\cdot z_i|$ for each set I_j, and, for each z_i, the error caused by discretizing the projection length with the S cutting points (i.e., the value ψ of Lemma A.4). In what follows we assume that everything went well. That is, we avoided the probability β that Lemma A.3 fails.

First we prove the lower bound. Consider I_j for $1 \le j \le j_{\max}$. By Lemma A.3, at least $(\alpha_j - \varphi)D$ of the projections $|(x-y)\cdot z_i|$ are in the set I_j. For each such z_i, by the definition of I_j, we have that $|(x-y)\cdot z_i| \ge (1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2$. Every point p_k $(1 \le k \le S)$ such that p_k is between $(x-w)\cdot z_i$ and $(y-w)\cdot z_i$ contributes 1 to the Hamming distance. Lemma A.4 shows that the number of such points is at least $(-\psi + ((1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2)/2L)S$, provided that both $(x-w)\cdot z_i$ and $(y-w)\cdot z_i$ are contained in the segment $[-L, L]$. As $x \in B(w,\ell)$ and by the triangle inequality $y \in B(w,(1+\delta^{-1})\ell)$, the corresponding projections are not contained in the segment $[-L,L]$ only if they fall in the set I_∞. For each vector this happens with probability at most α, by Lemma A.1. Thus the probability that both vectors fall in this segment is at least $1-2\alpha$.

For the lower bound we can ignore the bad events: the is for which $|(x-y)\cdot z_i|$ falls in I_0 and I_∞, as well as the is for which $(x-w)\cdot z_i$ or $(y-w)\cdot z_i$ fall outside $[-L,L]$. These contribute nonnegative terms to the distance. We get that $H(\eta(x),\eta(y))$ is at least

$$\sum_{j=1}^{j_{\max}} \left(-\psi + \frac{(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L} \right) \cdot (\alpha_j - \varphi)DS - 2\alpha DS.$$

As $\varphi = \delta\alpha'$ and $\alpha_j > \alpha'$ we get that $\varphi < \delta\alpha_j$ and so $(\alpha_j - \varphi) > (1-\delta)\alpha_j$. Also, note that ψ is at most δ times the other term: This term is minimized at $j=1$ and $\|x-y\|_2 = \lambda\ell$, and in this case

$$\frac{(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L} = \frac{\lambda\ell\sqrt{\frac{\delta}{d}}}{2(1+\delta^{-1})\ell\sqrt{\log(1/\delta)/d}} \ge \delta\lambda/2(1+\delta^{-1}).$$

By Corollary A.2,

$$\begin{aligned}\sum_{j=1}^{j_{\max}} \left(\alpha_j(1+\delta)^{j-1}\sqrt{\frac{\delta}{d}}\right) &= \frac{1}{1+\delta}\sum_{j=1}^{j_{\max}} \left(\alpha_j(1+\delta)^{j}\sqrt{\frac{\delta}{d}}\right) \\ &\ge \frac{1}{1+\delta}\left(E[X] - (1+o(1))b\sqrt{\frac{\delta}{d}}\right) \\ &= (1-O(\delta))b'\sqrt{\frac{1}{d}}.\end{aligned}$$

Combining everything we get that the lower bound is at least

$$\begin{aligned}&\left(\frac{(1-O(\delta))b'\sqrt{1/d}\|x-y\|_2}{2L} - 2\alpha\right) DS \\ &\qquad = \left((1-O(\delta))\frac{b'}{2(1+\delta^{-1})\sqrt{\log(1/\delta)}} \cdot \frac{\|x-y\|_2}{\ell} - O(\delta)\right) DS.\end{aligned}$$

Now we show the upper bound. By Lemma A.4, at most $(\alpha_j + \varphi)D$ of the projections $|(x-y)\cdot z_i|$ are in the set I_j for $1 \le j \le j_{\max}$, and at most $(\alpha+\varphi)D$ for I_0 and I_∞. If $|(x-y)\cdot z_i|$ is in I_j for $0 \le j \le j_{\max}$, then $|(x-y)\cdot z_i| \le (1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2$. By Lemma A.4, the contribution of z_i to the Hamming distance is at most $(\psi + ((1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2)/2L)S$, provided (as before) that $(x-w)\cdot z_i$ and $(y-w)\cdot z_i$ are contained in the segment $[-L, L]$. The latter happens with probability at least $1-2\alpha$. With the remaining probability, the contribution of z_i is no more than S.

If z_i is in I_∞, we have no bound on the distance between $x\cdot z_i$ and $y\cdot z_i$, but the contribution of z_i to the Hamming distance is no more than S. Summing this up, we get an upper bound of at most

$$\sum_{j=0}^{j_{\max}} \left(\psi + \frac{(1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L}\right)\cdot(\alpha_j+\varphi)DS + 2\alpha DS + (\alpha_\infty+\varphi)DS.$$

As before, the choice of parameters implies that $(\alpha_j+\varphi) \le (1+\delta)\alpha_j$ and $\psi \le \delta\cdot\frac{(1+\delta)^j\sqrt{\frac{\delta}{d}}\|x-y\|_2}{2L}$. Using the lower bound in Corollary A.2,

$$\begin{aligned}\sum_{j=0}^{j_{\max}}\left(\alpha_j(1+\delta)^j\sqrt{\frac{\delta}{d}}\right) &= \alpha_0\sqrt{\frac{\delta}{d}} + \sum_{j=1}^{j_{\max}}\left(\alpha_j(1+\delta)^j\sqrt{\frac{\delta}{d}}\right)\\ &\le O(\delta)\sqrt{\frac{\delta}{d}} + E[X]\\ &= (1+O(\delta^{3/2}))b'\sqrt{\frac{1}{d}}.\end{aligned}$$

We get that the Hamming distance is at most

$$\left(\frac{(1+O(\delta))\sqrt{1/d}\|x-y\|_2}{2L} + (3+\delta)\alpha\right)DS =$$
$$\left((1+O(\delta))\frac{b'}{2(1+\delta^{-1})\sqrt{\log\delta^{-1}}}\cdot\frac{\|x-y\|_2}{\ell} + O(\delta)\right)DS. \qquad \square$$

Acknowledgments. We thank Shai Ben-David, Nati Linial, and Avi Wigderson for helpful comments on earlier versions of this work.

REFERENCES

[1] P.K. Agarwal and J. Matoušek, *Ray shooting and parametric search*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 517–526.

[2] N. Alon and J. Spencer, *The Probabilistic Method*, John Wiley and Sons, New York, 1992.

[3] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Wu, *An optimal algorithm for approximate nearest neighbor searching in fixed dimensions*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 573–582.

[4] J.S. Beis and D.G. Lowe, *Shape indexing using approximate nearest-neighbor search in high-dimensional spaces*, in Procceedings of the IEEE Conference on Computer Vision and Pattern Recogognition, Japan, 1997, pp. 1000–1006.

[5] K. Clarkson, *A randomized algorithm for closest-point queries*, SIAM J. Comput., 17 (1988), pp. 830–847.
[6] K. Clarkson, *An algorithm for approximate closest-point queries*, in Proceedings of the 10th Annual ACM Symposium on Computational Geometry, Stony Brook, New York, 1994, pp. 160–164.
[7] S. Cost and S. Salzberg, *A weighted nearest neighbor algorithm for learning with symbolic features*, Machine Learning, 10 (1993), pp. 57–67.
[8] T.M. Cover and P.E. Hart, *Nearest neighbor pattern classification*, IEEE Trans. Inform. Theory, 13 (1967), pp. 21–27.
[9] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman, *Indexing by latent semantic analysis*, J. Amer. Soc. Inform. Sci., 41 (1990), pp. 391–407.
[10] L. Devroye and T.J. Wagner, *Nearest neighbor methods in discrimination*, in Handbook of Statistics, Vol. 2, P.R. Krishnaiah and L.N. Kanal, eds., North Holland, Amsterdam, 1982.
[11] D. Dobkin and R. Lipton, *Multidimensional searching problems*, SIAM J. Comput., 5 (1976), pp. 181–186.
[12] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas, *Neighborhood preserving hashing and approximate queries*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 251–259.
[13] D. Dolev, Y. Harari, and M. Parnas, *Finding the neighborhood of a query in a dictionary*, in Proceedings of the 2nd Israel Symposium on the Theory of Computing and Systems, IEEE Computer Society Press, Los Alamitos, CA, 1993, pp. 33–42.
[14] R.O. Duda and P.E. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973.
[15] T. Figiel, J. Lindensrauss, and V.D. Milman, *The dimension of almost spherical sections of convex bodies*, Acta Math., 139 (1977), pp. 53–94.
[16] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P.Yanker, *Query by image and video content: The QBIC system*, IEEE Computer, 28 (1995), pp. 23–32.
[17] U. Feige, D. Peleg, P. Raghavan, and E. Upfal, *Computing with unreliable information*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, Baltimore, MD, 1990, pp. 128–137.
[18] A. Gersho and R.M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, Dordrecht, The Netherlands, 1991.
[19] O. Goldreich and L. Levin, *A hard-core predicate for all one-way functions*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 25–32.
[20] T. Hastie and R. Tibshirani, *Discriminant adaptive nearest neighbor classification*, in 1st International ACM Conference on Knowledge Discovery and Data Mining, ACM, New York, 1995.
[21] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 604–613.
[22] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, *Locality-preserving hashing in multidimensional spaces*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 618–625.
[23] W.B. Johnson and J. Lindenstrauss, *Extensions of Lipschitz mappings into Hilbert space*, Contemp. Math., 26 (1984), pp. 189–206.
[24] J. Kleinberg, *Two algorithms for nearest-neighbor search in high dimensions*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 599–608.
[25] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[26] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.
[27] N. Linial and O. Sasson, *Non-expansive hashing*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 509–518.
[28] J. Matoušek, *Reporting points in halfspaces*, in Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, IEEE Computer Society Press, Los Alamitos, CA, 1991, pp. 207–215.
[29] S. Meiser, *Point location in arrangements of hyperplanes*, Inform. and Comput., 106 (1993), pp. 286–303.

[30] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[31] A. Pentland, R.W. Picard, and S. Sclaroff, *Photobook: tools for content-based manipulation of image databases*, in Proceedings of the SPIE Conference on Storage and Retrieval of Image and Video Databases II, San Jose, CA, SPIE Proceedings 2185, Bellingham, WA, 1994, pp. 34–37.

[32] G. Salton, *Automatic Text Processing*, Addison-Wesley, Reading, MA, 1989.

[33] A.W.M. Smeulders and R. Jain, *Proceedings of the* 1*st Workshop on Image Databases and Multi-Media Search*, World Sci. Ser. Software Engrg. and Knowledge Engrg. 8, World Scientific, River Edge, NJ, 1996.

[34] V.N. Vapnik and A.Y. Chervonenkis, *On the uniform convergence of relative frequencies of events to their probabilities*, Theory Probab. Appl., 16 (1971), pp. 264–280.

[35] A.C. Yao and F.F. Yao, *A general approach to d-dimension geometric queries*, in Proceedings of the 17th Annual ACM Symposium on Theory of Computing, Providence, RI, 1985, pp. 163–168.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 475–485

WHEN HAMMING MEETS EUCLID: THE APPROXIMABILITY OF GEOMETRIC TSP AND STEINER TREE*

LUCA TREVISAN†

Abstract. We prove that the traveling salesman problem (MIN TSP) is Max SNP-hard (and thus NP-hard to approximate within some constant $r > 1$) even if all cities lie in a Euclidean space of dimension $\log n$ (n is the number of cities) and distances are computed with respect to any l_p norm. The running time of recent approximation schemes for geometric MIN TSP is doubly exponential in the number of dimensions. Our result implies that this dependence is necessary unless NP has subexponential algorithms. As an intermediate step, we also prove the hardness of approximating MIN TSP in Hamming spaces.

Finally, we prove a similar, but weaker, inapproximability result for the Steiner minimal tree problem (MIN ST).

The reduction for MIN TSP uses error-correcting codes; the reduction for MIN ST uses the integrality property of MIN-CUT linear programming relaxations. The only previous inapproximability results for metric MIN TSP involved metrics where all distances are 1 or 2.

Key words. computational complexity, combinatorial optimization, hardness of approximation, traveling salesman problem, Steiner tree problem

AMS subject classifications. 68Q25, 90C27

PII. S0097539799352735

1. Introduction.

Given a metric space and a set U of points into it, the metric traveling salesman problem (MIN TSP) is to find a closed tour of shortest total length visiting each point exactly once, while the metric Steiner minimum tree problem (MIN ST) is to find the minimum length tree connecting all the points of U; the tree can possibly contain points not in U, which are called "Steiner points."

Both are classical and well-studied problems. Important special cases arise when the metric space is $\mathcal{R}^k$ and the distance is computed according to the ℓ_1 norm (the *rectilinear* case) or the ℓ_2 norm (the *Euclidean* case).

We establish the first nonapproximability results for this class of problems. As an intermediate step, we use the fact that they are also hard to approximate in *Hamming spaces*. The approximability of the Hamming versions of MIN TSP seems to have never been considered before. The hardness of approximating this problem is one of the main technical results of this paper. The hardness of approximating MIN ST in Hamming spaces has been studied for its application to problems in computational biology (specifically, reconstructing evolutionary trees), but it has not been linked to the hardness of the problem in geometric norms.

We now state and discuss our results for MIN TSP and MIN ST.

1.1. The traveling salesman problem.

Interest in the MIN TSP started during the 1930s. A book by Lawler et al. [28] contains an extensive survey of results on MIN TSP. Here we will review only the results that are relevant to the present paper.

*Received by the editors March 3, 1999; accepted for publication (in revised form) October 28, 1999; published electronically June 3, 2000. A preliminary version of this paper appeared in the proceedings of the 29th ACM Symposium on Theory of Computing [37]. This work was done while the author was at the University of Geneva, at the University of Rome "La Sapienza" and while visiting the Technical University of Catalonia (UPC).

http://www.siam.org/journals/sicomp/30-2/35273.html

†Department of Computer Science, 462 CSB, Mail Code 0401, Columbia University, New York, NY 10027-6902 (luca@cs.columbia.edu).

The MIN TSP is NP-hard even if the cities are restricted to lie in $\mathcal{R}^2$ and the distances are computed according to the ℓ_2 norm [15, 31]. Due to such a negative result, research concentrated on developing good heuristics. Recall that an r-approximate algorithm ($r > 1$) is a polynomial-time heuristic that is guaranteed to deliver a tour whose cost is at most r times the optimum cost. A 3/2-approximate algorithm that works for any metric space is due to Christofides [9]. For more than twenty years, no improvement of this bound had been found, even in the restricted case of geometric metrics.

In the late 1980s, the discovery of the theory of Max SNP-hardness [35] gave a tool for understanding this lack of results. Indeed, Papadimitriou and Yannakakis [34] proved that the MIN TSP is Max SNP-hard even when restricted to metric spaces (as we shall see later, the result also holds for a particularly restricted class of metric spaces). As later shown by Arora et al. [6], this implies that there exists a constant $\epsilon > 0$ such that metric MIN TSP cannot be approximated to within a factor $(1 + \epsilon)$ in polynomial time, unless P = NP. The complexity of approximating MIN TSP in the case of geometric metrics remained a major open question. In his Ph.D. thesis, Arora noted that proving the Max SNP-hardness of Euclidean MIN TSP in $\mathcal{R}^2$ should be very difficult but that this could perhaps be done in $\mathcal{R}^{k(n)}$ for sufficiently large $k(n)$ [2, Chapter 9]. In [18], Grigni, Koutsoupias, and Papadimitriou proved that the restriction of the MIN TSP to shortest paths metrics of planar graphs can be approximated to within $(1 + \epsilon)$ in time $n^{O(1/\epsilon)}$. Such an approximation algorithm is called a *polynomial time approximation scheme* (PTAS) (the result was later generalized to the case of shortest paths metrics of planar weighted graphs in [5]). This result led Grigni, Koutsoupias, and Papadimitriou [18] to conjecture that Euclidean MIN TSP has a PTAS in $\mathcal{R}^2$. They again posed the question of determining the approximability of the problem for higher dimensions. In a recent breakthrough, Arora [3] developed a PTAS for the MIN TSP in $\mathcal{R}^2$ under any ℓ_p metric. Such an algorithm also works in higher dimensional spaces and, in particular, it runs in time $n^{\tilde{O}((\log^{d-2} n)/\epsilon^{d-1})}$ in $\mathcal{R}^d$. A similar approximation scheme (but only for spaces of dimension 2) was also found later by Mitchell [29]. Arora has subsequently improved the running time of his approximation scheme [4]; his new scheme runs in nearly-linear time for any fixed number of dimensions. Specifically, the algorithm of [4] finds a $(1 + \epsilon)$-approximate solution in $\mathcal{R}^d$ in time $n(\log n)^{O\left((\sqrt{d}/\epsilon)^d\right)}$. An additional improvement is due to Rao and Smith [36]. Their algorithm runs in time $(\sqrt{d}/\epsilon)^{O(d(\sqrt{d}/s)^{d-1})}n + O(dn \log n)$. The dependence of the running time on the number of dimensions is, however, still doubly exponential.

This is a typical occurrence of the "curse of dimensionality," a phenomenon empirically observed in several cases in computational geometry, that is, the fact that the complexity of a geometric problem grows exponentially or more in the number of dimensions of the space. In some cases (e.g., nearest neighbor search [26, 27, 21]) clever algorithmic solutions can be developed to avoid this exponential growth. In a preliminary version of [3], Arora asked whether an approximation scheme for geometric MIN TSP exists for any arbitrary number of dimensions.

Our results. In this paper we essentially answer negatively to these questions. We prove that MIN TSP in $\mathcal{R}^{\log n}$ is Max SNP-hard for any ℓ_p metric. It follows from our results that there cannot be a PTAS for these problems (unless P= NP) and that there cannot be $(1 + \epsilon)$-approximate algorithms in $\mathcal{R}^d$ running in time $n^{O(\mathrm{poly}(d,1/\epsilon))}$ for any $\epsilon > 0$, unless $\mathsf{NP} \subseteq \mathsf{DTIME}(n^{O(\mathrm{poly}\log n)})$.

The Max SNP-hardness is proved by means of a reduction from the version of the

metric MIN TSP that was shown to be Max SNP-hard in [34]. In such metric spaces, any pair of points is either at distance 1 or 2, and an additional technical condition holds. The reduction uses a mapping (see Lemma 3.5) of the metric spaces of [34] into Hamming spaces and the observation (see Proposition 2.3) that for elements of $\{0,1\}^n$ a "gap" in the Hamming distance is preserved if distances are computed according to an ℓ_p metric. Our mapping of the metric spaces of [35] into Hamming spaces does not preserve distances up to negligible distortion. (In fact we suspect that such kind of mapping would be provably impossible.) Instead, our mapping introduces a fairly high (yet constant) distortion but satisfies an additional condition: cities at distance 1 are mapped into cities at distance $\approx D_1$; cities at distance 2 are mapped into cities at distance $\approx D_2$, and D_2 is larger than D_1 by a multiplicative constant factor. This is sufficient to make the mapping be an *approximation preserving reduction*. Our mapping uses *error-correcting codes*, a tool that is very seldom used in order to design approximation-preserving reductions.

The minimum k-cities traveling salesman problem (MIN k-TSP) and the minimum degree-restricted Steiner tree problem (two problems mentioned in Arora's papers on approximation schemes for geometric problems [3, 4]) are generalizations of the MIN TSP. The hardness results that we prove for MIN TSP clearly extend to them.

1.2. The minimum Steiner tree problem. The MIN ST problem has an even longer history than the MIN TSP. Special cases were studied by Torricelli (in the 17th century) and by Gauss (see [20] for more details on the history of this problem). Recent results about this problem are similar to the ones for MIN TSP: exact optimization is NP-hard in $\mathcal{R}^2$ both in the rectilinear (ℓ_1) case [17] and in the Euclidean (ℓ_2) case [16]. Constant-factor approximation is achievable in any metric space (the best factor is 1.644 due to Karpinski and Zelikovsky [24]); in general metric spaces the problem is Max SNP-hard [7]. Arora [3, 4], Mitchell [29], and Rao and Smith [36] show how to extend their geometric TSP approximation schemes to geometric MIN ST. The running time of these approximation schemes are the same as reported in the previous section for TSP. See also the books by Hwang, Richards, and Winters [20] and Ivanov and Tuzhilin [22] and the web page maintained by Ganley [14] for extended surveys on the Steiner tree literature. No nonapproximability result was known for geometric versions of the Steiner tree problem. Steiner tree in Hamming spaces is known to be Max SNP-hard, though this result is usually expressed in terms of an equivalent problem in computational biology (see Wareham's Master's thesis [39] and also [23, 40]).

Our results. We prove the Max SNP-hardness of MIN ST in $\mathcal{R}^n$ under the ℓ_1 norm. We establish this result by means of a reduction from the MIN ST problem in Hamming spaces. The reduction is based on the following combinatorial result (Theorem 4.2): for an instance where all the points are in $\{0,1\}^n \subset \mathcal{R}^n$, there exists an optimum solution where all the Steiner points lie in $\{0,1\}^n$. We prove this fact using the *integrality property* of MIN-CUT linear programming relaxations.

1.3. Discussion. For Euclidean MIN TSP, there is still a slight slack between recent approximation schemes and our hardness result. Specifically, a running time $2^{(2^d)/\epsilon}\text{poly}(n)$ would be compatible with our results, but if we believe that NP does not admit subexponential algorithms (i.e., $\text{NP} \not\subseteq \text{DTIME}(2^{n^{o(1)}})$), then even a running time $2^{2^{o(d)}/\epsilon}\text{poly}(n)$ is infeasible. For a fixed ϵ, the approximation scheme of Rao and Smith [36] runs in time $(\sqrt{d}/\epsilon)^{O(d(\sqrt{d}/s)^{d-1})}n + O(dn\log n)$ which, for fixed ϵ, is

roughly $2^{2^{(d \log d)/2 + \log\log n}}$.

There is much more room for improvement for the MIN ST problem; however our results at least imply that the number of dimensions *does matter* in the running time of an approximation scheme for this geometric problem.

We feel that one important contribution of this paper is the recognition of Hamming spaces as a class of metric spaces that are somewhat "close" both to arbitrary metrics and to geometric metrics, while they also have a nice combinatorial structure. This combination of characteristics makes problems on Hamming metrics a useful "intermediate" step in reducing a combinatorial problem to a geometric problem.

Our mapping of bounded-degree graphs into Hamming spaces has been used by Crescenzi et al. [10] in order to prove the NP-hardness of the protein folding problem.

2. Preliminaries. We denote by $\mathcal{R}$ the set of real numbers. For an integer n we denote by $[n]$ the set $\{1, \ldots, n\}$. For a vector $a \in \mathcal{R}^n$ and an index $i \in [n]$, we denote by $a[i]$ the ith coordinate of a. The *weight* of a Boolean vector $a \in \{0,1\}^n$ is the number of nonzero entries.

Given an instance x of an optimization problem A, we will denote by $\mathsf{opt}_A(x)$ the cost of an optimum solution for x, and we will also typically omit the subscript. For a feasible solution y (usually a tour or a tree) of an instance x of an optimization problem A, we denote its cost by $\mathsf{cost}_A(x, y)$ or, more often, as $\mathsf{cost}(y)$. See, e.g., [8, 32] for formal definitions about optimization problems. In this paper we will use the notions of L-reduction and Max SNP-hardness. Max SNP is a class of constant-factor approximable optimization problems that includes MAX 3SAT; we refer the reader to [35] for the formal definition.

DEFINITION 2.1 (L-reduction). *An optimization problem A us said to be L-reducible to an optimization problem B if two constants α and β and two polynomial-time computable functions f and g exist such that*

1. *for an instance x of A, $x' = f(x)$ is an instance of B, and it holds $\mathsf{opt}_B(x') \leq \alpha \mathsf{opt}_A(x)$;*
2. *for an instance x of A, and a solution y' feasible for $x' = f(x)$, $y = g(x, y')$ is a feasible solution for x and it holds $|\mathsf{opt}_A(x) - \mathsf{cost}_A(x, y)| \leq \beta |\mathsf{opt}_B(x') - \mathsf{cost}_B(x', y')|$.*

We say that an optimization problem A is Max SNP-hard if all Max SNP-problems are L-reducible to A. From [6] it follows that if a problem A is Max SNP-hard, then there exists a constant $\epsilon > 0$ such that $(1+\epsilon)$-approximating A is NP-hard.

A function $d : U \times U \to \mathcal{R}$ is a *metric* if the following properties hold:

1. $d(u, v) \geq 0$ for all $u, v \in U$;
2. $d(u, v) = 0$ if and only if $u = v$;
3. $d(u, v) = d(v, u)$ for any $u, v \in U$ (symmetry);
4. $d(u, v) \leq d(u, z) + d(z, v)$ for any $u, v, z \in U$ (triangle inequality).

If all properties but (2) hold, then d is said to be a *semimetric*. Abusing notation, we will usually adopt the term "metric" for both metrics and semimetrics.

DEFINITION 2.2 ($(1,2) - B$ metrics). *For a positive integer B, a metric $d : U \times U \to \mathcal{R}$ is a $(1,2) - B$ metric if it satisfies the following properties:*

1. *for any $u, v \in U$, $u \neq v$, $d(u, v) \in \{1, 2\}$;*
2. *for any $u \in U$, at most B elements of U are at distance 1 from u.*

Papadimitriou and Yannakakis [34] have shown that a constant $B_0 > 0$ exists such that the MIN TSP is Max SNP-hard even when restricted to $(1,2) - B_0$ metrics.

For an integer $p \geq 1$, the ℓ_p norm in $\mathcal{R}^n$ is defined as

$$\|(u_1, \ldots, u_n)\|_p = \left(\sum_{i=1}^{n} |u_i|^p \right)^{(1/p)}.$$

The distance induced by the ℓ_p norm is defined as $d_p(u, v) = \|u - v\|_p$. For a positive integer n, we denote by d_H^n the Hamming metric in $\{0,1\}^n$ (we will usually omit the superscripts). We will make use of the following fact.

PROPOSITION 2.3. *Let $u, v \in \{0,1\}^n \subseteq \mathcal{R}^n$. Then $d_p(u, v) = d_H(u, v)^{1/p}$.*

Before starting with the presentation of our results, we make the following important caveat.

Remark 2.1. In some of the proofs of this paper we implicitly make the (unrealistic) assumption that arbitrary real numbers can appear in an instance and that arithmetic operations (including squared roots) can be computed over them in constant time. However, our results still hold if we instead assume that numbers are rounded and stored in a floating point notation using $O(\log n)$ bits. This fact follows from a modification of the argument used in [3] to reduce a general instance of Euclidean TSP or Steiner tree into an instance where coordinates are positive integers whose value is $O(n^2)$.

3. The Min TSP. Our hardness result is based on a "distance preserving" embeddings of $(1,2) - B$ metric spaces into Hamming spaces. We first define the kind of embedding we are looking for.

DEFINITION 3.1. *For an integer B, a $(1,2) - B$ metric space (U, d), an integer k and positive reals $D_1, D_2 > 0$ and $0 < \epsilon < 1/2$, we say that a mapping $f : U \to \{0,1\}^k$ is (k, D_1, D_2, ϵ)-good if for any $u, v \in U$*

1. *if $d(u, v) = 1$, then $D_1(1 - \epsilon) \leq d_H(f(u), f(v)) \leq D_1(1 + \epsilon)$;*
2. *if $d(u, v) = 2$, then $D_2(1 - \epsilon) \leq d_H(f(u), f(v)) \leq D_2(1 + \epsilon)$.*

In particular, if f is a $(k, D_2, D_1, 0)$-good embedding, then pairs at distance 2 are mapped into pairs at distance D_2 and pairs at distance 1 are mapped into pairs at distance D_1.

Recall that, for any integer h, if we let $n = 2^h$, then the *first-order Reed–Muller code* (which is also an *Hadamard code*) $H_n \subset \{0,1\}^n$ is a set of n binary strings of length n whose pairwise Hamming distance is $n/2$. The elements of H_n can be seen as the set of linear functions $l : \{0,1\}^h \to \{0,1\}$. See, e.g., [38, Chapter 18] and the references therein for more details. The set H_n is computable in time polynomial in n.

LEMMA 3.2. *There exists a polynomial time algorithm that on input a $(1,2) - B$ metric with n points, where n is a power of 2, finds a $((B+1)n, Bn/2, (B+1)n/2, 0)$-good embedding.*

Proof. Let $U = \{u_1, \ldots, u_n\}$. Recall that a $(1,2) - B$ metric (U, d) can be represented as an undirected graph $G = (U, E)$, where $\{u, v\} \in E$ if and only if $d(u, v) = 1$ (see [34]). Note that G has maximum degree B.

We claim that we can find in polynomial time a partition of E into $B + 1$ matchings $E_1, \ldots, E_{B+1}$. To prove this claim, it suffices to observe that the problem of partitioning the set of edges of a graph into disjoint matchings is a restatement of the *edge coloring problem*. In a graph of maximum degree B, an edge coloring with $B + 1$ colors can be found in polynomial time [19].

We now describe the embedding. Each node $u \in U$ is mapped into a string $f(u)$

that is the concatenation of $B+1$ strings $a_u^1, \ldots, a_u^{B+1} \in H_n$:

$$f(u) = a_u^1 \circ \cdots \circ a_u^{B+1}.$$

For a fixed $i \in \{1, \ldots, B+1\}$, the strings $\{a_u^i\}_{u \in U}$ are chosen arbitrarily in H_n such that $a_u^i = a_v^i$ if and only if $\{u, v\} \in E_i$. Since H_n can be generated in polynomial time in n, the overall construction can be carried out in $\mathrm{poly}(n)$ time.

Let us now compute the distance between two strings $f(u)$ and $f(v)$. There are two cases to be considered.

1. If $\{u, v\} \notin E$, then $a_u^i \neq a_v^i$ for all $i = 1, \ldots, B+1$, and so $d_H(f(u), f(v)) = (B+1) \cdot n/2$.
2. If $\{u, v\} \in E$, then $\{u, v\} \in E_j$ for some j, and we have $a_u^j = a_v^j$ and $a_u^i \neq a_v^i$ for $i \neq j$. It follows that $d_H(f(u), f(v)) = B \cdot n/2$. ∎

We also observe the following simpler result.

LEMMA 3.3. *There exists a polynomial time algorithm that on input a $(1,2) - B$ metric with n points finds a $((B+1)n, 2B, 2(B+1), 0)$-good embedding. Furthermore, any vector in the embedding has weight precisely $B+1$.*

Proof. Use the same construction of the proof of Lemma 3.2, but using the code $I_n \subset \{0,1\}^n$ composed of all the n vectors of length n having exactly one nonzero entry. Any two elements of such a code have distance precisely 2. ∎

Our next goal is to describe a good embedding that maps into Hamming spaces having a logarithmic number of dimensions. We will need error-correcting codes whose existence (and explicit construction) is guaranteed by the following result of Naor and Naor.

LEMMA 3.4 (see [30]). *For every $\epsilon > 0$ and positive integer n there is a collection $C_{n,\epsilon} \subseteq \{0,1\}^{k(n,\epsilon)}$ such that $|C_{n,\epsilon}| = n$, $k(n,\epsilon) = O((\log n)/\mathrm{poly}(\epsilon))$, and for any two elements $u, v \in C_{n,\epsilon}$ we have $k(n,\epsilon)(1/2 - \epsilon) \leq d_H(u,v) \leq k(n,\epsilon)(1/2 + \epsilon)$. Furthermore, there is a procedure that on input n and ϵ outputs $C_{n,\epsilon}$ in $\mathrm{poly}(n, 1/\epsilon)$ time.*

The connection between the results of [30] and the explicit construction of error correcting codes is made explicit in section 7 of [1]. The precise dependency of k on ϵ in [30] is of the form $k(n,\epsilon) = O(n/\epsilon^{4+\gamma})$, where $\gamma > 0$ can be an arbitrary positive constant. Some improvements are possible, but the actual dependency on ϵ is not important for our application.

LEMMA 3.5. *There exists a polynomial time algorithm that, for any B and $\gamma > 0$, on input a $(1,2) - B$ metric (U,d) with n points finds a $(k, DB/(B+1), D, \gamma)$-good embedding of U, where $k = O((\log Bn)/\mathrm{poly}(\gamma))$ and $D = k/2$.*

Proof. Use the argument in the proof of Lemma 3.2, but with the codes of Lemma 3.4 instead of the Hadamard code. ∎

THEOREM 3.6. *For any p there exists a constant $\epsilon_p > 0$ such that* MIN TSP *is* NP*-hard to approximate within a factor $(1 + \epsilon_p)$ even when restricted to ℓ_p spaces of logarithmic dimension.*

Proof. From [34] and [6] we have the following result: constants $B_0 > 0$ and $r_0 > 1$ exist such that, given an instance (U,d) of MIN TSP with a $(1,2) - B_0$ metric and n cities, and given the promise that either $\mathsf{opt}(U,d) = n$ or $\mathsf{opt}(U,d) \geq r_0 n$, it is NP-hard to decide which of the two cases holds.

Fix a constant γ such that

$$\frac{1-\gamma}{1+\gamma}\left(1 + \frac{(r_0 - 1)}{B_0}\right) > 1 + \frac{(r_0-1)}{2B_0} \stackrel{\mathrm{def}}{=} 1 + \epsilon_1.$$

Such a constant γ must exist since for $\gamma \to 0$ the left-hand side tends to a value strictly greater than the right-hand side.

Given an instance (U, d) of $(1,2) - B_0$ MIN TSP with n cities, we use Lemma 3.5 to map it into a Hamming space of dimension $k = O(\log n)$ using a $(k, DB_0/(B_0 + 1), D, \gamma)$-good embedding with $D = k/2$. Let $f : U \to \{0,1\}^k$ denote such embedding. We consider two cases.

- If $\mathsf{opt}(U, d) = n$, then an optimum solution for U will have cost at most $n \cdot D(B_0/(B_0 + 1)) \cdot (1 + \gamma)$ for U'.
- If $\mathsf{opt}(U, d) \geq nr_0$, then there can be no solution for U' of cost less than $nD(B_0/(B_0 + 1))(1 - \gamma) + (r_0 - 1)n(1/(B_0 + 1))D(1 - \gamma)$. Otherwise, the same solution would have cost less than nr_0 for U.

Distinguishing between the two cases is NP-hard; therefore it is NP-hard to approximate the target instance to within a factor

$$\frac{1-\gamma}{1+\gamma} \cdot \frac{(B_0 + r_0 - 1)/(B_0 + 1)}{B_0/(B_0 + 1)} \geq 1 + \epsilon_1.$$

It follows that it is NP-hard to approximate MIN TSP to within a factor $1 + \epsilon_1$ even in the special case of Hamming instances having a logarithmic number of dimensions. The same nonapproximability result holds for the case ℓ_1 metric space (since Hamming TSP is just a special case of TSP in ℓ_1 spaces). The result for general ℓ_p spaces follows by using Proposition 2.3. □

Remark 3.1. The claim of Theorem 3.6 asks for the cities to be in $\mathcal{R}^{\log n}$, rather than in $\mathcal{R}^{c \log n}$ as in the previous construction. However, we can add $(n^c - n)$ new cities, all at distance $1/n^{c+1}$ from a given one. This perturbs the optimum in a negligible way, and gives an instance with $N = n^c$ cities in $\mathcal{R}^{\log N}$.

Using techniques of Khanna et al. [25], the nonapproximability result of Theorem 3.6 implies that geometric MIN TSP in $\mathcal{R}^{\log n}$ under any ℓ_p norm is APX PB-hard (in particular, Max SNP-hard) under E-reductions and APX-complete under AP-reductions [11].

3.1. Additional remarks. Using the embedding of Lemma 3.3 in the proof of Theorem 3.6, one can prove the Max SNP-hardness of MIN TSP when restricted to Hamming instances with a constant bound on the weight of the points. Reducing from TSP(1,2) in graphs with maximum degree 3 (an NP-hard problem), it is possible to prove the NP-hardness of the Hamming TSP problem in instances where all points have weight precisely 4. This problem was not known to be NP-hard before. It is reasonable to conjecture that Hamming TSP is solvable in polynomial time for instances where all points have weight at most 2 and is NP-hard for instances where all points have weight 3. We do not know how to prove this conjecture.

4. The Min ST problem. The hardness of approximating MIN ST will be established by means of a reduction from the Hamming version of the problem.

The following result appears in [39, Theorem 45, part 3]. It is based on the observation that the NP-hardness proof of Hamming MIN TSP that appears in [12] yields an L-reduction from vertex cover in bounded degree graphs (a problem proved to be Max SNP-hard in [35]) to Hamming MIN ST.

THEOREM 4.1 (see [39]). MIN ST *is* Max SNP*-hard even when restricted to Hamming spaces.*

Our goal is to reduce the Steiner tree problem in Hamming spaces to the Steiner tree problem under the ℓ_1 distance. We note that for points in $\{0,1\}^n$ the ℓ_1 distance

equals the Hamming distance. However, the reduction is nontrivial since $\mathcal{R}^n$ contains many points that are not in $\{0,1\}^n$ and we have to argue that having much more choice for the Steiner nodes does not make the problem easier. The rectilinear Min ST problem looks very much like a *relaxation* of the Hamming Min ST problem; our reduction makes use of a *rounding scheme* proving that the relaxation does not change the optimum.

Theorem 4.2. *Let $U \subseteq \{0,1\}^n \subset \mathcal{R}^n$ be an instance of rectilinear* Min ST *all of whose points are in the Boolean cube. Let T be a feasible solution for U. Then it is possible to find in polynomial time (in the size of T) another solution T' such that $\mathsf{cost}(T') \leq \mathsf{cost}(T)$ and all the Steiner nodes of T' are in $\{0,1\}^n$.*

Before proving the theorem, we note the following interesting consequence.

Corollary 4.3. *For any instance $U \subseteq \{0,1\}^n$ of rectilinear* Min ST*, an optimum solution exists all of whose Steiner points are in $\{0,1\}^n$.*

We now prove Theorem 4.2.

Proof of Theorem 4.2. Let $S = \{s_1, \ldots, s_m\}$ be the set of Steiner points of T, and let E be the set of edges of T. For any $s_j \in S$ we will find a new point $s'_j \in \{0,1\}^n$, so that if we let T' be the tree obtained from T by substituting the s points with the corresponding s' points, the cost of T' is not greater than the cost of T. The latter statement is equivalent to

$$\begin{aligned}&\sum_{(s_j,u)\in E, u\in U} ||s_j - u||_1 + \sum_{(s_j,s_h)\in E} ||s_j - s_h||_1 \\ \geq &\sum_{(s'_j,u)\in E, u\in U} ||s'_j - u||_1 + \sum_{(s'_j,s'_h)\in E} ||s'_j - s'_h||_1.\end{aligned}$$

We will indeed prove something stronger, namely, that for any $i \in [n]$ it holds

$$\begin{aligned}&\sum_{(s_j,u)\in E, u\in U} |s_j[i] - u[i]| + \sum_{(s_j,s_h)\in E} |s_j[i] - s_h[i]| \\ \geq &\sum_{(s'_j,u)\in E, u\in U} |s'_j[i] - u[i]| + \sum_{(s'_j,s'_h)\in E} |s'_j[i] - s'_h[i]|.\end{aligned} \tag{4.1}$$

Let $i \in [n]$ be fixed; we must now find values of $s'_1[i], \ldots, s'_m[i] \in \{0,1\}$ such that (4.1) holds. We express as a linear program the problem of finding values of $s'_1[i], \ldots, s'_m[i]$ that minimize the right-hand side of (4.1). For any $j \in [m]$ we have a variable x_j (representing the value to be given to $s'_j[i]$) and for any edge $e = (a,b)$ such that at least one endpoint is in S, we have a variable y_e representing the length $|a[i] - b[i]|$. The linear program is as follows:

$$\begin{array}{lll}\text{(LP)} & \min & \sum_e y_e \\ & \text{subject to} & \\ & & y_e \geq x_j - x_h \quad \forall e = (s_j, s_h) \in E, \\ & & y_e \geq x_h - x_j \quad \forall e = (s_j, s_h) \in E, \\ & & y_e \geq x_j \quad \forall e = (s_j, u_h) \in E. u_h[i] = 0, \\ & & y_e \geq 1 - x_j \quad \forall e = (s_j, u_h) \in E. u_h[i] = 1, \\ & & x_j \geq 0, \\ & & y_e \geq 0.\end{array}$$

Setting $x_j = s_j[i]$ and $y_{(a,b)} = |a[i] - b[i]|$ yields a feasible solution, and its cost is the left-hand side of (4.1). Let (x^*, y^*) be an optimum solution for (LP). From the

previous observation we have that by setting $s'_j[i] = x^*_j$ we satisfy (4.1). It remains to be seen that (LP) has an optimum solution where all variables take value from $\{0,1\}$. This follows from the fact that (LP) is the linear programming relaxation of an undirected MIN-CUT problem, where all the u such that $u[i] = 0$ (respectively, $u[i] = 1$) are identified with the source (respectively, the sink), each s_j is a node, and the edges are like in T. It is well known (see, e.g., [33]) that a MIN-CUT linear programming relaxation has optimum 0/1 solutions, and that such a solution can be found in polynomial time. □

Remark 4.1. *There seems to be no natural analog of Theorem* 4.2 *in other norms. Even in* $\mathcal{R}^2$, *using the Euclidean metric, we have that the optimum solution of the instance* $\{(0,0),(1,0),(0,1)\}$ *must use a Steiner point not in* $\{0,1\}^2$.

THEOREM 4.4. *Rectilinear* MIN ST *is* Max SNP*-hard.*

Proof. We reduce from Hamming MIN ST. The reduction leaves the instance unchanged. For an instance $U \subseteq \{0,1\}^n$, we let $\mathsf{opt}_H(U)$ (respectively, $\mathsf{opt}_R(U)$) be the cost of an optimum solution for U, when seen as an instance of Hamming MIN ST (respectively, of rectilinear MIN ST). Clearly, we have that $\mathsf{opt}_R(U) \leq \mathsf{opt}_H(U)$. Given a solution T for U, we find a solution T' as in Theorem 4.2. Since in $\{0,1\}^n$ the distance induced by the ℓ_1 norm equals the Hamming distance, we have that $\mathsf{cost}_H(T') = \mathsf{cost}_R(T') \leq \mathsf{cost}_R(T)$. We have an L-reduction with $\alpha = \beta = 1$. □

5. Conclusions and open questions. We do not know how to extend our nonapproximability result for MIN ST to the Euclidean case. Arora [3] notes that, by inspecting the way his algorithm works, it is possible to claim that, for any instance of Euclidean MIN ST, there exists a near-optimal solution where the Steiner points lie in some well-specified positions (either at "portals" or in positions chosen at the bottom of the recursion). This observation could perhaps be a starting point.

We do not have explicit estimations of the constants into which it is hard to approximate geometric MIN TSP and rectilinear MIN ST. The constant for MIN TSP should be only slightly smaller than the corresponding constant for the $(1,2) - B$ case. An explicit nonapproximability factor has been estimated by Engebresten [13] for the latter problem, and it is very close to 1. The constant for MIN ST should be slightly better. Finding much stronger estimations (comparable to the 3/2 bound of Christofides [9] and the 1.644 bound of Karpinski and Zelikovsky [24]) is an open and challenging question.

Acknowledgments. I am grateful to Sanjeev Arora, Madhu Sudan, László Babai, and Janos Körner for very interesting and helpful discussions. I thank Ray Greenlaw for having read a preliminary version of this paper and given several useful remarks. I wish to thank Jens Lagergren, Tao Jiang, and Todd Wareham for providing me references and reprints about their work in computational biology, and for explaining to me the relation with the Hamming–Steiner tree problem.

REFERENCES

[1] N. ALON, O. GOLDREICH, J. HÅSTAD, AND R. PERALTA, *Simple constructions of almost k-wise independent random variables*, Random Structures Algorithms, 3 (1992), pp. 289–304.

[2] S. ARORA, *Probabilistic Checking of Proofs and Hardness of Approximation Problems*, Ph.D. thesis, Department of Electrical Engineering and Computer Science, University of California at Berkeley, Berkeley, CA, 1994.

[3] S. ARORA, *Polynomial time approximation schemes for Euclidean TSP and other geometric problems*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1996, pp. 2–11.

[4] S. Arora, *Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems*, J. ACM, 45 (1998), pp. 753–782.
[5] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn, *Polynomial time approximation scheme for weighted planar graph TSP*, in Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, ACM, New York, 1998, pp. 33–41.
[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.
[7] M. Bern and P. Plassmann, *The Steiner tree problem with edge lengths* 1 *and* 2, Inform. Process. Lett., 32 (1989), pp. 171–176.
[8] D. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1993.
[9] N. Christofides, *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*, Tech. Rep., Carnegie-Mellon University, Pittsburgh, PA, 1976.
[10] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis, *On the complexity of protein folding*, in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, ACM, New York, 1998, pp. 597–603.
[11] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan, *Structure in approximation classes*, in Proceedings of the 1st Combinatorics and Computing Conference, Lecture Notes in Comput. Sci. 959, Springer-Verlag, New York, 1995, pp. 539–548.
[12] W. Day, D. Johnson, and D. Sankoff, *The computational complexity of inferring rooted phylogenies by parsimony*, Math. Biosci., 81 (1986), pp. 33–42.
[13] L. Engebresten, *An Explicit Lower Bound for TSP with Distances One and Two*, Tech. Rep. TR98-046, Electronic Colloquium on Computational Complexity, 1998.
[14] J. Ganley, *The Steiner Tree Page*, http://www.cs.virginia.edu/∼jlg8k/steiner.
[15] M. Garey, R. Graham, and D. Johnson, *Some NP-complete geometric problems*, in Proceedings of the 8th ACM Symposium on Theory of Computing, Hershey, PA, ACM, New York, 1976, pp. 10–22.
[16] M. Garey, R. Graham, and D. Johnson, *The complexity of computing Steiner minimal trees*, SIAM J. Appl. Math., 32 (1977), pp. 835–859.
[17] M. Garey and D. Johnson, *The rectilinear Steiner tree problem is NP-complete*, SIAM J. Appl. Math., 32 (1977), pp. 826–834.
[18] M. Grigni, E. Koutsoupias, and C. Papadimitriou, *An approximation scheme for planar graph TSP*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, IEEE, Piscataway, NJ, 1995, pp. 640–645.
[19] I. Holyer, *The NP-completeness of edge-coloring*, SIAM J. Comput., 10 (1981), pp. 718–720.
[20] F. Hwang, D. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, Amsterdam, 1992.
[21] P. Indyk and R. Motwani, *Approximate nearest neighbors: Towards removing the curse of dimensionality*, in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, ACM, New York, 1998, pp. 604–613.
[22] A. Ivanov and A. Tuzhilin, *Minimal Networks: The Steiner Problem and Its Generalizations*, CRC Press, Boca Raton, FL, 1994.
[23] T. Jiang and L. Wang, *On the complexity of multiple sequence alignment*, J. Comput. Biol., 1 (1994), pp. 337–348.
[24] M. Karpinski and A. Zelikovsky, *New approximation algorithms for the Steiner tree problems*, J. Comb. Optim., 1 (1997), pp. 1–19.
[25] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani, *On syntactic versus computational views of approximability*, SIAM J. Comput., 28 (1999), pp. 164–191.
[26] J. M. Kleinberg, *Two algorithms for nearest-neighbor search in high dimensions*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, ACM, New York, 1997, pp. 599–608.
[27] E. Kushilevitz, R. Ostrovsky, and Y. Rabani, *Efficient search for approximate nearest neighbor in high dimensional spaces*, in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, ACM, New York, 1998, pp. 614–623.
[28] E. Lawler, J. Lenstra, A. R. Kan, and D. Shmoys, *The Traveling Salesman Problem*, John Wiley, New York, 1985.
[29] J. S. B. Mitchell, *Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems*, SIAM J. Comput., 28 (1999), pp. 1298–1309.
[30] J. Naor and M. Naor, *Small-bias probability spaces: Efficient constructions and applications*, in Proceedings of the 22nd ACM Symposium on Theory of Computing, St. Louis, MO, IEEE, Piscataway, NJ, 1990, pp. 213–223.

[31] C. Papadimitriou, *Euclidean TSP is NP-complete*, Theoret. Comput. Sci., 4 (1977), pp. 237–244.
[32] C. Papadimitriou, *Computational Complexity*, Addison-Wesley, Reading, MA, 1994.
[33] C. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[34] C. Papadimitriou and M. Yannakakis, *The travelling salesman problem with distances one and two*, Math. Oper. Res., 18 (1993), pp. 1–11.
[35] C. H. Papadimitriou and M. Yannakakis, *Optimization, approximation, and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.
[36] S. Rao and W. Smith, *Approximating geometrical graphs via "spanners" and "banyans,"* in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, ACM, New York, 1998, pp. 540–550.
[37] L. Trevisan, *When Hamming meets Euclid: The approximability of geometric TSP and MST*, in Proceedings of the 29th ACM Symposium on Theory of Computing, Dallas, TX, ACM, New York, 1997, pp. 21–29.
[38] J. van Lint and R. Wilson, *A Course in Combinatorics*, Cambridge University Press, Cambridge, UK, 1992.
[39] H. Wareham, *On the Computational Complexity of Inferring Evolutionary Trees*, Master's thesis, Department of Computer Science, Memorial University of Newfoundland, Canada, 1993. Also available online from http://www.csr.uvic.ca/∼harold/.
[40] H. Wareham, *A simplified proof of the NP- and MAX SNP-hardness of multiple sequence tree alignment*, J. Comput. Biol., 2 (1995), pp. 509–514.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 486–510

SELF-STABILIZATION BY COUNTER FLUSHING*

GEORGE VARGHESE†

Abstract. A useful way to design simple and robust protocols is to make them self-stabilizing. A protocol is said to be self-stabilizing if it begins to exhibit correct behavior even after starting in an arbitrary state. We describe a simple technique for self-stabilization called *counter flushing*. We show how counter flushing helps us to understand and improve some existing distributed algorithms for tasks such as mutual exclusion and request-response protocols. We also use counter flushing to create *new* self-stabilizing protocols for propagation of information with feedback and resets. The resulting protocols are simple, require few changes from the nonstabilizing equivalents, and have fast stabilization times.

Key words. distributed algorithms, self-stabilization

AMS subject classifications. 68Q22, 68Q60, 68Q25

PII. S009753979732760X

1. Introduction.

As the world moves from an industrial economy to an information-based economy, we are dependent on networks and will become even more so. Thus it is important to design network protocols that are resilient to faults.

In the traditional approach to network fault-tolerance, the protocol designer enumerates the faults that the network will deal with—e.g., node and link crashes, bit errors on links—and adds recovery mechanisms for each such fault. This adds complexity as the mechanisms are not orthogonal and have subtle interactions. Also, there are a number of more obscure errors (e.g., memory corruption) that occur in real networks and are hard to anticipate and enumerate. Even if such faults occur rarely (say, once a month), it makes economic sense to have networks automatically recover from such faults.

These issues are illustrated by the crash of the original ARPANET protocol [Ros81, Per83]. The protocol was designed never to enter a state that contained three conflicting updates. Unfortunately, a malfunctioning node injected three such updates into the network and crashed. After this the network cycled continuously between the three updates until the problem was diagnosed a day later.

Self-stabilization. Ideally networks should recover by themselves, even from obscure faults. This paper describes a paradigm for designing what are known as *self-stabilizing* network protocols. We do so to make network protocols *simpler* and *more robust*.

We model a computer network as a set of nodes interconnected by communication links. A network protocol consists of a program for each network node. Each program consists of code and inputs as well as local state. The global state of the network consists of the local state of each node as well as the messages on network links. A network protocol is *self-stabilizing* if when started from an arbitrary state it exhibits

*Received by the editors September 22, 1997; accepted for publication (in revised form) September 7, 1999; published electronically June 3, 2000.
http://www.siam.org/journals/sicomp/30-2/32760.html
†Department of Computer Science, University of California, San Diego, La Jolla, CA 92093, and Department of Computer Science, Washington University, St. Louis, MO 63130 (varghese@cs.ucsd.edu). This research was supported in part by NSF grant NCR-9612853 and an ONR Young Investigator Award.

"correct" behavior after finite time. This definition allows arbitrary corruption of messages and node state variables in the initial state.

Note that we allow network *state* to be corrupted but not the *code* executing the protocol. This is reasonable because program code can be protected against arbitrary corruption of memory by redundancy techniques (e.g., checksumming) since code is rarely modified. However, it is not clear how one can detect corruption of network state by using redundancy techniques. The definition also seems to imply that faults can occur only once (i.e., when the network "starts"). However, the real assumption is, *The average period between faults is larger than the time the protocol takes to stabilize.*

The distributed algorithm literature also describes *Byzantine* fault models ([LSP82]) in which arbitrary faults can continuously occur. However, in Byzantine models, only a fraction of nodes are allowed to exhibit arbitrary behavior. In the self-stabilization model, *all* nodes are permitted to start with arbitrary initial states. Thus, the two models are orthogonal. In a practical setting the crucial difference is that the cost of stabilization is quite cheap while Byzantine solutions are expensive. For example, the self-stabilizing routing protocol in [Per83] is much cheaper than the Byzantine routing protocol of [Per88].

General techniques for self-stabilization. Self-stabilizing protocols were introduced by Dijkstra in [Dij74a]. [Sch93] contains a review of research in this area. While many ad hoc self-stabilizing protocols have been designed, there are few general techniques for self-stabilization. Katz and Perry [KP93] showed how to compile an arbitrary asynchronous protocol into a stabilizing equivalent. Their general transformation is expensive; hence more efficient (and possibly less general) techniques are needed. Techniques that transform any *locally checkable* protocol into a stabilizing equivalent are given in [AGV94, Var93]. However, local checking applies only to a subset of protocols that have a special property called *local checkability.*

Our paper describes a new general technique called *counter flushing* that is applicable to protocols that are not locally checkable. The abstract setting is that of a leader who periodically delivers a message to every network node (and sometimes to every link). By attaching a counter to the state of every node and to every message, and by using simple checks, we ensure that the protocol begins to work correctly regardless of initial messages and node states. Counter flushing can be applied to *total algorithms.* Total algorithms [Tel89] require the cooperation of all network nodes. Specifically, we apply counter flushing to token passing [Dij74a], propagation of information with feedback [Seg83], and network resets [AG94].

The rest of the paper is organized as follows. Section 2 describes our model. Section 3 describes how counter flushing works on ring topologies and shows how counter flushing can be used for token passing, request-response, and data link protocols. Section 4 describes how counter flushing works on trees by describing a stabilizing broadcast protocol called propagation of information with feedback. In section 5, we extend counter flushing to a general graph. We illustrate this by designing a stabilizing network reset protocol. In section 6, we present a uniform proof of stabilization and correctness of our three main protocols. The uniformity of proof emphasizes the unity behind the diversity of applications. We conclude in section 7.

2. Model. We restrict ourselves to message passing protocols. The network topology is modeled by a directed graph $G = (V, E)$, where $n = |V|$ denotes the number of network nodes and D the network diameter. Except for the case when we consider a ring (section 3), we assume the graph is symmetric—i.e., if $(i, j) \in E$, then $(j, i) \in E$. We assume a distinguished leader node $0 \in V$ that we often refer to as the

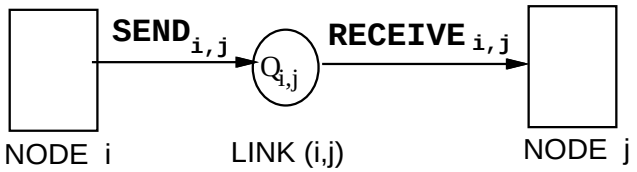


FIG. 1. *The rectangles represent node automata for nodes i and j with interfaces to send packets from i to j. The circle represents the link automaton from i to j whose state is captured by a queue, $Q_{i,j}$.*

root. Note that there are stabilizing protocols [AKM+93, Dol94, AK93] to construct a leader in $O(D)$ time. We model network nodes and links using input/output automata (IOA) [LT89].

An IOA is a state machine whose state transitions are given labels called *actions*. There are three kinds of actions. The environment affects the automaton through *input actions* which must be responded to in any state. The automaton affects the environment through *output actions*; these actions can be controlled by the automaton to occur only in certain states. *Internal actions* change the state of the automaton without affecting the environment.

Formally, an IOA is defined by a *state set* S, a *action set* A, an *action signature* Z (that classifies the action set into input, output, and internal actions), a *transition relation* $R \subseteq S \times A \times S$, and a set of *initial states* $I \subseteq S$. We mostly deal with *uninitialized IOA* for which $I = S$. An action a is said to be *enabled* in state s if there exist $s' \in S$ such that $(s, a, s') \in R$. Input actions are always enabled.

Nodes communicate with each other by sending to and receiving *packets* along links. Fix a packet alphabet P. Nodes and links are modeled by IOA called *node* and *link* automata, respectively. A node automaton (see Figure 1) for node i in graph G has output actions $(\text{SEND}_{i,j}(p), p \in P)$ to send a packet to every j such that $(i, j) \in E$; it also has input actions to receive packets $(\text{RECEIVE}_{j,i}(p), p \in P)$ for every j such that $(j, i) \in E$. Similarly, the link automaton for link $(i, j) \in E$ has input action $\text{SEND}_{i,j}(p)$ to receive packets[1] from i, and output action $\text{RECEIVE}_{i,j}(p)$ to deliver packets to node j (Figure 1).

Node automata are arbitrary but have *finite* state sets and have the appropriate interface actions to send and receive packets. Each link is modeled as a FIFO queue with *bounded storage*. Formally, the state of the link automaton for link (i, j) is a queue of packets $Q_{i,j}$ that is restricted to store no more than L_{max} packets. A formal specification of a *bounded data link* is in Figure 2. We use a bounded model because not much can be done with *unbounded* links in a stabilizing setting [DIM91], and real links are bounded anyway. We justify this model further at the end of this section.

A *network automaton* for graph $G = (E, V)$ is the composition of node automata for each $i \in V$ and link automata for each edge $(i, j) \in E$. *Composition* [LT89] produces a composite state machine; input and output actions of the same name are performed simultaneously. Thus when node i performs a $\text{SEND}_{i,j}(p)$ output action, the link between i and j performs a simultaneous input action (also $\text{SEND}_{i,j}(p)$) to store packet p. The state of the composition is the cross product of the states of every node and link automaton in the network automaton.

[1] The convention for action subscripts is that the first subscript always represents the sending node and the second the receiving node.

State of link $C_{i,j}$ is a queue of packets $Q_{i,j}$.
In all states, $|Q_{i,j}| \leq L_{max}$.

SEND$_{i,j}(p)$ (* node i sends a packet *)
Effects: Add p to the end of $Q_{i,j}$

RECEIVE$_{i,j}(p)$ (* node j receives a packet*)
Preconditions: p is the head of $Q_{i,j}$
Effects: Remove p from $Q_{i,j}$

For time complexity, assume that any packet p placed in $Q_{i,j}$ is delivered in 1 time unit.

FIG. 2. *Formal model of a bounded data link using a finite queue. Note the unusual time complexity assumption that is justified in the text.*

When an IOA "runs" it produces an execution. An *execution fragment* is an alternating sequence of states and actions $(s_0, a_1, s_1, \ldots)$, such that $(s_i, a_i, s_{i+1}) \in R$ for all $i \geq 0$. An execution fragment E is fair if any internal or output action that is continuously enabled eventually occurs. An *execution* is an execution fragment that begins with a start state and is fair.

We express the correctness of an automaton using a set of *legal executions* (*LE*) as in [DIM93]. Let *LE* be a set of executions of some automaton A. We say that automaton A stabilizes to *LE* if every execution of A has some suffix that is contained in *LE*. The *legal states* are the states that occur in legal executions. All the automata we will design in this paper will be *uninitialized IOA* whose set of initial states I is identical to its state set S. We do so by not specifying initial node or channel states. Note that the intuitive concept of self-stabilization is captured by the stabilization of an uninitialized automaton to a set of legal executions.

For time complexity, assume that every internal or output action that is continuously enabled at a node occurs in 1 unit of time. Thus node processing takes 1 time unit. However, we assume any packet stored on a link is delivered in 1 time unit. We say that A *stabilizes* to *LE in time* t if every execution of A has a suffix that occurs within time t and is contained in *LE*. The *stabilization time* from A to B is the infimum across all t such that A stabilizes to B in time t.

Our time complexity assumption for messages is reasonable for links, such as fiber links, in which packets are not queued on links. The assumption is completely unrealistic for channels that act like queues; a simple example is a link that is really a "network" which consists of internal switching nodes. However, since we expect our protocols (token ring, tree broadcast, and reset) to be used over networks with the former type of link, we believe this assumption is reasonable.

The time complexity assumption also seems to imply (see the description of algorithms in Figures 4, 5, and 8) that each node has to send a stabilization message at every step. In particular, this seems to follow because we have made the time complexity for nodes to send messages the same as the time complexity for a message to travel on a link. We choose this for simplicity in order to avoid having two parameters for the two times. A similar (but slightly more messy) analysis of the algorithms can be carried out in which the time to perform an internal node action is bounded by

some parameter t_n. The general model would allow stabilization messages to be sent at any reasonable interval, and would provide the usual tradeoff between message overhead and stabilization time.

There are several other methods of calculating time complexities for stabilizing protocols. These include methods in which time complexity is measured in terms of rounds in which every processor takes a step. Our time complexity measure is not directly comparable to these other measures.

Bounded links. In a stabilizing setting, if a link can store an unbounded number of packets, it is impossible to produce solutions (with bounded stabilization time) for most nontrivial tasks [DIM91]. Moreover, real links are bounded.

In other work, we have modeled bounded links as unit capacity data links (UDLs) that can store at most one packet at any instant. A UDL [Var93] is a model of a reliable data link protocol that delivers one packet at a time. The UDL model is appropriate for routing (and other) protocols that use an underlying reliable link protocol. However, many real protocols that work over very low error rate links (e.g., FDDI, ATM, Frame Relay [Tan93]) do not use an underlying data link protocol. Such links store a bounded number of packets because the link sender and receiver are *synchronous*. The receiver removes packets as fast as the sender inputs packets. However, there is no common clock for the entire network; *node processing is still asynchronous*.

We can model this using an asynchronous model like IOA if we only consider executions in which there are no more than L_{max} packets on each link in each state. In our bounded model we assume that any stored packet is delivered in constant time. For example, suppose the minimum packet size is 20 bytes, nodes transmit at 100 Mbit/sec and links are up to 10 miles long. Then $L_{max} = 30$ and (assuming speed of light limitations) any stored packet is delivered in 50 usec. Both numbers are constants that depend only on the maximum length of a link.

3. Counter flushing on rings. We start by showing how counter flushing can be used in rings. Our protocol is a message passing version of a shared memory protocol presented in [Dij74a]. Nodes $0, \ldots, n-1$ are arranged in a ring topology with a directed link $(i, i+1)$ for $i = 0, \ldots, n-1$. Addition on process indices is always implicitly mod$n$ so that $n - 1 + 1 = 0$. Node 0 is called the root.

In a ring, without the need for stabilization, mutual exclusion can easily be achieved by sending a special *Token* packet round the ring. Each node i would have a flag which would be set to true when a token arrives at i, and be set to false when node i sends its token to node $i + 1$. As long as the protocol starts in a state where there is exactly one token, this protocol will maintain a mutual exclusion property: no more than one node can have its token flag set to true in any state.

However, in a stabilizing setting, the token protocol can deadlock. We can start the token protocol in a state that does not contain a token. A simple way to avoid deadlock is to have nodes periodically retransmit tokens. But this introduces the possibility of a node receiving duplicate tokens. Thus we change the state of each node i (and each token) to have a counter instead of a flag. A node with counter value c can identify a token numbered c as being a duplicate. This is analogous to the use of sequence numbers in network protocols [Tan93].

However, counters cause new complications. In the initial state, the counter values may be arbitrary. Let c_{max} be the maximum number of counters that can be stored in the network in the initial state. Because of the initially bounded model, $c_{max} = |E|L_{max} + |V|$. We will show that if the size of the counter space is greater than c_{max},

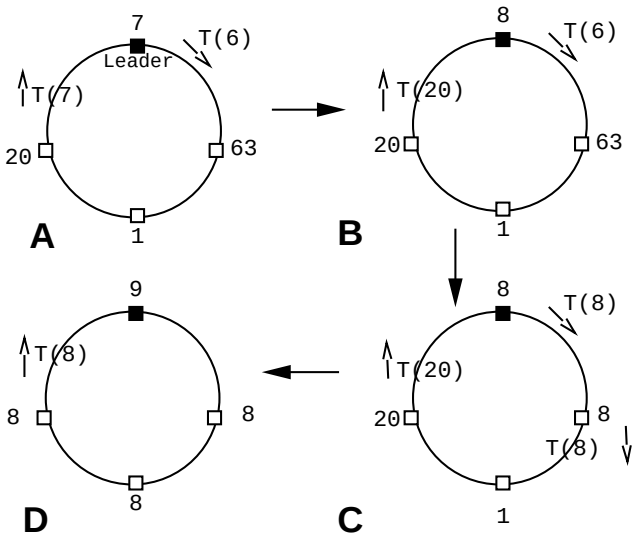


FIG. 3. *Progress to stabilization in a token ring with counters. Starting with an arbitrary state* (A) *we reach a state in which the root has a fresh value* (B). *The fresh value* (*i.e.,* 8) *moves round the ring* (C) *until we reach a legal state* (D).

then the protocol stabilizes in time proportional to the network diameter.

In Figure 3 the lower-left-hand corner (configuration D) describes a good global state of our protocol. There are four nodes—north, east, south, and west—and packets travel clockwise around the ring. North is the root. Each node other than north has a counter value of 8, and there is a token carrying the value 8 in transit from west to north.

Each node periodically retransmits its counter value in a token packet downstream. Thus mere receipt of a token packet is not enough for a node to assume it has the token. Instead, when any node i receives a token from its upstream neighbor, node i does the following. If i is not the root and the counter value in the token (say, c) *is different* from the counter stored at node i (i.e., c_i), then node i assumes it has received a *valid token* and sets c_i equal to c; if $c = c_i$ and i is not the root, i ignores the received token. If i is the root, however, a different rule is used: if the counter c in the token *is equal* to the root's local counter, then the root assumes it has received a *valid token*, and increments its counter value $\bmod Max$; if $c \neq c_i$, then the root ignores the received token.

Consider legal configuration D of Figure 3. In this state all token packets on links have a counter value 8, and the counter values at all nodes except north are also 8. The counter value at north is $9 \neq 8$. In that case, we say that north has the token. Eventually north will transmit a token packet containing 9. When this packet reaches east, east sets its counter value to 9. This process continues with the token moving clockwise until west receives the token and transmits it to north. North chooses a new value (10) and the cycle continues.

In legal states the ring can be partitioned into two bands. The first band starts with the root and continues up to (but not including) the first counter value (either in a token packet or at a node) whose counter value is different from that of north. The remainder of the ring (including links and nodes) is a second band containing a

counter value different from north. The valid token is at the end of the first band.

The protocol stabilizes to legal states regardless of *initial values of node and packet counters.* Assume c_{max} (the number of distinct counter values stored in nodes and links in the initial state) is less than the counter size *Max*. For example, with a 1000 node ring transmitting at 100 Mbp/s and assuming 10 mile links, $c_{max} = 31,000$. If we use a 32-bit counter, then $Max = 2^{32} > c_{max}$ for even the largest size rings that occur in practice. The stabilization argument is illustrated in Figure 3. We provide a formal argument in section 6. For now we sketch an informal argument that illustrates three essential features of the counter flushing argument (increment liveness, freshness, and flushing).

1. *Increment liveness.* In any execution, north will eventually increment its counter. Suppose not; then north's value will move around the ring until north gets a token with a counter value equal to its own.

2. *Freshness.* In any execution, north will reach a "fresh" counter value not equal to the counter values of any other process (see Figure 3, configuration B). In the initial state there are at most (say) c_{max} distinct counter values. Thus there is some counter value, say, f, not present in the initial state. Since (by increment liveness) north keeps incrementing its counter, north will eventually reach f; in the interim no other process can set their counter value to f since only north "produces" new counter values.

3. *Flushing.* Any state in which north has a fresh counter value f is eventually followed by a state in which all processes have counter value f (see Figure 3, configurations C, D). The value f moves clockwise around the ring "flushing" any other counter values, while north remains at f.

Define a *round trip delay* to be equal to $2N$ time units (i.e., the time it takes a packet to travel around the ring with a unit delay at each node and link). The worst-case stabilization time of this protocol is equal to three round trip delays.

First consider a modification to the protocol in which the root chooses a new counter value *randomly* instead of incrementing the old value. Assume that $Max \gg c_{max}$ (i.e., counter size is much greater than the maximum number of stored packets). With very high probability (i.e., $1 - c_{max}/Max$, roughly 2^{-16} for our ring example), the root picks a "fresh" value after its first opportunity to change its counter. Thus a *randomized counter flushing* ring protocol stabilizes within two round trip delays with high probability.

However, simple *deterministic* incrementing also guarantees a worst case stabilization time of three ring round trip delays. Here is the intuition. Consider an execution with initial state s_I and some state s_F that occurs one round trip delay later. Let the counter value of the root in s_I and s_F be $c(I)$ and $c(F)$, respectively. In one round trip delay, there is enough time for information from the root to "flow" through the entire ring. Thus all node counters in state s_F must have been "produced" by the root since the execution began. More precisely, in s_F, all counter values are in the range $[c(I), \ldots, c(F)]$. If $c(F)$ is fresh, we are done one round trip delay later (see the stabilization argument above). Otherwise, the next root increment will cause a fresh value because $c(F) + 1 \notin [c(I), \ldots, c(F)]$. (This last fact follows because the root will increment at most once for every packet received, and can receive at most $c_{max} < Max$ packets in the interval $[s_I, s_F]$.) But the next increment will happen after at most one round trip delay.[2]

[2] A more careful accounting shows that two round trip delays suffice for stabilization. We prefer to use three round trip delays because of the simplicity of the proofs.

A token packet is encoded as a tuple $(Token, c)$ where c is an integer in the range $0..Max$.
The state of each node i consists of an integer $count_i$ in the range $0..Max$.
The root has an additional flag $token\_expected_0$.
Assume there are n nodes numbered from 0 to $n-1$.
All addition and subtraction of process indices is mod n.
All addition and subtraction of counters is mod Max.

Finished(i) (* boolean function used by action routines below *)
Always true for all nodes other than node 0
Finished(0) is true if and only if $token\_expected_0 = false$

ROOT_START (* Node 0 is considered the root or leader of ring *)
Preconditions:
$Finished(0) = true$
Effects:
$count_0 = count_0 + 1$ (* increment root counter *)
$token\_expected_0 = true$

RECEIVE$_{n-1,0}(Token, c)$ (* node 0 receives token from node $n-1$ *)
Effects:
If $c = count_0$ then (* token counter matches node counter *)
$token\_expected_0 = false$ (* node 0 treats this packet like an ack*)

RECEIVE$_{i-1,i}(Token, c), i \neq 0$ (* node i receives token from clockwise neighbor node i-1*)
Effects:
If $c \neq count_i$ then (* token counter differs from node counter *)
$count_i = c$ (*set value to counter in token packet*)

SEND$_{i,i+1}(Token, c)$, (* node i sends token to clockwise neighbor node i + 1*)
Preconditions:
$c = count_i$ (* counter of token matches node counter *)

For any node, a SEND$_{i,i+1}$ action will occur in 1 unit of time starting from any state.

FIG. 4. *Code for node processes in a token ring. The code is explained in more detail in the main text of the paper.*

The formal code for our stabilizing token passing protocol is in Figure 4. Our protocol is a message passing version of a shared memory protocol in [Dij74a]. One of our contributions is to prove that the stabilization time is equal to three ring delays, using our model of time complexity,[3] which we believe is realistic. But our main contribution is abstracting the mechanism and applying it to other examples, as we show below. A formal proof of correctness and stabilization is deferred to section 6.

Token passing protocols are widely used in local area networks to regulate access to the network. Existing token passing protocols recover from lost tokens using global timers that are refreshed whenever a token is seen. In the IBM token ring [Tan93], the monitor (i.e., root) uses a timer initialized to the longest possible delay for a token to traverse the ring. When this timer expires, the monitor restarts the protocol. Thus

[3]There are no time complexity results in [Dij74a].

the recovery time of the IBM protocol is proportional to the worst-case ring delay. The recovery time of a token passing protocol based on counter flushing (see [CV98] for details) is proportional to the *actual delay around the ring*, which can be an order of magnitude faster than the worst-case delay.

3.1. Further applications on rings. Counter flushing on rings can be applied to two more interesting settings: request-response protocols and data link protocols.

Request-response protocols. Suppose a leader node wishes to periodically send a *Request* packet to a set of network nodes. The responders must each send back a *Response* packet before the sender sends its next request. In order to match responses to requests, the sender numbers [Var93] each request with a counter. Responders only accept *Request* packets with a number different from the last *Request* accepted. After accepting a *Request* the responder sends back a *Response* with the same number as the *Request*. The sender retransmits the current *Request* till it receives each matching *Response* with the same number. After all matching *Response* packets arrive, the sender increments its counter and starts a new phase. The protocol will work correctly if $Max > c_{max}$ and the links are FIFO (or guarantee the "flushing" property in some other way.)

Data link protocols. The token passing protocol in Figure 4 can be adapted to send packets reliably between a sender and receiver by having each token packet carry a piggybacked data packet. By comparison, the elegant stabilizing data link protocol of Afek and Brown [AB93] uses bounded length counters of size greater than 2, but such that the sequence of counter values used is aperiodic. A trivial corollary is that for a pair of nodes connected by a pair of unidirectional links, it suffices to use a counter whose size is larger than 1 plus the maximum number of outstanding packets. Afek and Brown also suggest the use of a random sequence instead of an aperiodic sequence. However, Afek and Brown's result is confined to data link protocols between a pair of nodes and to rings and has not been extended to trees or general networks as we do below. Also, the expected stabilization time of the randomized equivalent of Afek–Brown's protocol is shown to be $O(c_{max})$ round trip delays for large values of *Max*, while our stabilization time is only two round trip delays.

4. Counter flushing on trees. In the last section, we describe counter flushing on a ring using a mutual exclusion protocol that is essentially sequential. By contrast, in this section we describe a broadcast protocol on a tree which exhibits considerable parallelism. In this problem, we have a root node 0 that wishes to broadcast a sequence of values to every node in the network. Correct executions of the protocol can be partitioned into an infinite number of cycles: in cycle M the root must send the packet corresponding to M exactly once to all network nodes. Cycle M begins after cycle $M-1$ ends. In order to detect when the current broadcast cycle has terminated the root needs to obtain feedback from the other nodes. Thus the problem is often called propagation of information with feedback (PIF).

We model the sequence of values that the root wishes to broadcast by having the root have access to a function f that computes the next value to be sent as a function of the previous value sent. In a more general setting, the values could be supplied by some external application.

We assume a root node 0 and a spanning tree rooted at node 0, such that each node i has a parent variable *parent*(i) that points to its parent in the tree. Without stabilization, it is easy to solve this problem using protocols due to Segall and Chang [Seg83, Cha82]. When the root finishes broadcasting a previous value, it chooses a new value using the function f. It then sends a token packet containing the new

value to all its children; other nodes accept new values only from their parents, upon which they send the value to their children. When a leaf of the tree gets a new value, it sends an ack up to its parent. Nodes other than the root send an ack up to their parents when they have received acks from all children. When the root receives an ack from all children, the root starts a new cycle by choosing a new value. Clearly this protocol can deadlock if initialized in a state where the root is expecting acks from its children, but the children do not send any further acks.

To make the protocol stabilizing, we tag each packet sent (and each value stored) with a counter. When sending a new value, the root chooses a new counter value. Node i accepts a new value only when it is tagged with a *different* counter value from the counter stored at node i. Node i accepts an ack only when the counter in the ack is *identical* to node i's counter. Adding counters and checks also allows us to periodically retransmit *Token* packets to avoid deadlock.

The code is given in Figure 5. For simplicity, we do not encode "acks" separately but just have children send $(Token, c, v)$ packets to their parents as acks, where c and v are the counter and value, respectively, at the sending node at the instant the packet was sent.

Figure 6 shows a legal state of the protocol where a new value x is being broadcast to replace the previous value v. The new counter tag for x is 13 while the counter tag for v was 12. The new value x has reached the right child of the root and is in transit to the rightmost leaf node. When this leaf node gets a packet containing $(x, 13)$ it will accept the new value because $12 \neq 13$. It will then send an ack containing the counter 13 to its parent; the parent will accept this as a valid ack.

Suppose the counter size is greater than $Max = nL_{max}$, the maximum number of outstanding counters in the initial state. Then the counter flushing argument guarantees that this protocol will enter a legal state in $4h + 2$ time (h is the tree height), regardless of the initial state. Once it stabilizes, the protocol correctly broadcasts subsequent values generated at the root. We defer a formal correctness proof to section 6.

It may appear that counters can be completely avoided because the PIF protocol works on a tree. However, there are counterexample executions where, if a counter is not used (or its size is less than *Max*), then the system stays in an incorrect state forever.

Another general method for constructing stabilizing protocols is local checking [APV91, Var93]. However, the PIF protocol of Figure 5 is not locally checkable. In a good state of the PIF protocol it is possible to have a parent have counter c and the child have counter $c' \neq c$ (if the value is still propagating from the parent to the child). Thus we can construct a bad global state in which each child of the root has a different counter value but each pair of neighbors appears to be in a good state locally. Independently, Gouda [Gou94] used the concept of observers to unify tree and ring stabilizing systems. His paper, however, uses a different proof from ours.

Further applications on trees. Propagation of information with feedback is an example of a centralized total algorithm [Tel89]. A centralized total algorithm is an algorithm where each process in the network must take some decision before the initiator takes a decision event. Tel [Tel89] shows that many protocols such as PIF, Finn's resynch protocol [Fin79], and distributed infimum[4] are all examples of total algorithms. Tel also shows that PIF can be used to replace any total algorithm. Thus

[4]Distributed infimum can be described as calculating a bound on the minimum of a set of values stored at network nodes and links. The global virtual time calculation used in optimistic distributed simulation is a special case of calculating the distributed infimum.

We assume all counters are integers in the range 0..*Max* and all values are drawn from some domain V.
A token packet is encoded as a tuple $(Token, c, v)$ where c is a counter and v is a value.
The state of each node i consists of:
a counter $count_i$, a boolean flag $token\_expected_i[j]$ for each neighbor j of i and a value field v_i.
We assume that each node i has a function $parent(i)$ that points to i's parent in the tree.
We assume the root is node 0 and all addition of counters are done mod *Max*.

Finished(i) (* boolean function used by action routines below *)
(* set to true when not expecting tokens from any children *)
Return true if i is a leaf, or if i is not a leaf and for all children k of i: $token\_expected_i[k] = false$

ROOT_START (* Root starts a new cycle of broadcasting values *)
Preconditions:
$Finished(0) = true$
Effects:
$v_0 = f(v_r)$ (* compute new value to be broadcast*)
$count_0 = count_0 + 1$ (*choose new counter value mod Max*)
For all children k of root
$token\_expected_0[k] = true$ (* set to true when expecting a correctly numbered token*)

SEND$_{i,j}$($Token, c, v$), (* node i sends token to node j *)
Preconditions:
$c = count_i$ (* counter of token matches node counter *)
$v = v_i$ (* value equal to store value *)
$j \neq parent(i)$ or ($j = parent(i)$ and $Finished(i)$) (* send to children, and to parent if finished *)

RECEIVE$_{j,i}$($Token, c, v$) (* node i receives token from node j *)
Effects:
If $j = parent(i)$ and $c \neq count_i$ then (* new counter from parent *)
$v_i = v$ (* set stored value equal to value in token packet*)
$count_i = c$ (*set local counter to counter in token packet*)
For all children k of i
$token\_expected_i[k] = true$ (* set to true when expecting a token *)
Else if $c = count_i$
$token\_expected_i[j] = false$ (* treat as a valid ack from child j*)

Any action that is continuously enabled for 1 unit of time occurs in 1 unit of time.

FIG. 5. *Code for stabilizing propagation of information with feedback. The code is explained in more detail in the main text of the paper.*

the stabilizing PIF protocol described in Figure 5 offers a stabilizing solution to any problem that requires a total algorithm. An interesting application of the stabilizing PIF protocol is for topology update. For example, in the Autonet network [MAM+90], topology distribution is done over a tree.

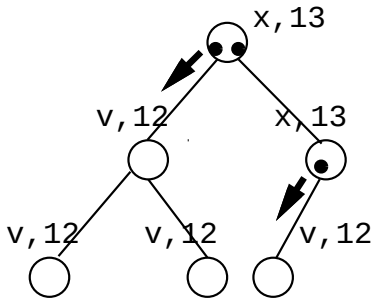


FIG. 6. *Using counter flushing to make broadcast on a tree stabilizing. A new value x is being broadcast to replace the previous value v. A black dot indicates that a node is waiting for an ack on that link.*

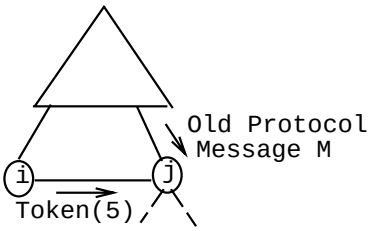


FIG. 7. *The reason for delaying responses to token packets received on cross links (like the one from i to j) is to prevent j from responding to the old (prereset) protocol message after i has locally reset.*

5. Counter flushing in general graphs. We broaden the scope of counter flushing to consider general graphs. However, we continue to assume a root node 0 that is the root of a breadth first search (BFS) spanning tree. Besides links from parents to children we now also have cross links that are not part of the tree. We have seen how to use counter flushing to flush tree links; we now extend the paradigm to flush both tree and cross links.

As before, a node i only accepts a new counter value c from its parent and waits until it gets tokens from its children (numbered with c) before it sends a token up to its parent. However, in addition, i sends a token packet on any cross links it is part of and waits to get a token (numbered with c) on every link before it sends a token to its parent.

The only difficulty is deciding how to reply to token packets received on cross links. Before we see the problems, we introduce an application (global reset) for general counter flushing. We have an underlying protocol P; the root may get requests to reset protocol P. We stipulate that at the point the reset procedure terminates, the state of the underlying protocol P is reset to some successor of a legal initial state of P. To do so, at some point during the reset procedure, (i) each node i must locally reset its protocol P state, (ii) define the *reset interval* of a node to be the interval from the time a node is locally reset until the reset procedure terminates. Then for any pair of neighbors i, j, the sequence of packets received by node j in j's reset interval must be a prefix of the sequence of packets sent by node i during i's reset interval.

In Figure 7, node i has received the counter value (5) corresponding to the current reset and has sent a token packet containing 5 to j. Node j has not received information on the current reset and has an "old" protocol packet in transit from its parent.

Suppose node i's token packet reaches node j first and node j sends back a token immediately (but without changing its counter value or initializing protocol P). Then node j can subsequently receive the "old" protocol packet and send another "old" protocol packet to node i. Thus, we could have a packet sent before node j was reset being received by node i after node i has reset, an error.

A seemingly simple solution is for node j to reset itself locally when it receives the token packet on the cross link from node i. However, if a node accepts counters on cross links to its neighbors, then in the initial configuration we could have a cycle of nodes with different counter values which can result in livelock, where the counters move around in the cycle. This problem can be resolved [KP93] by having each token packet carry a counter *and* a list of visited nodes; a token packet is dropped when it revisits a node in the visited list. However, this solution increases message and time complexity.

The livelock problem disappears if nodes only accept counter values from their parents. To solve the problem referred to in Figure 7, we do two things. First, we tag all protocol P packets with the counter at the sending node; we discard a protocol P packet with a counter that does not match the receiver's counter. This solution eliminates the problem in Figure 7 because the "old" packet will have a different counter value from that of node i. But it introduces another problem. Suppose node i sends a protocol P packet to j after node i resets, but the packet is received before j resets. Then if we simply check the packet tag, the packet will be dropped at j. One might consider buffering the packet at j if the counter tag in the packet is "greater" than the counter at j; however, defining one counter to be greater than another is fraught with complications when using bounded size counters.

Instead we have each node j *delay* responding until the local counter at node j is equal to the counter of the token packet received. Thus in Figure 7 when j receives the token packet from i numbered 5, node j does not send a token numbered 5 back to node i until node j has also received a token packet numbered 5 from its parent. In the meantime, node i will keep retransmitting a token packet numbered 5 to j. Node j will ignore these packets until it, too, has the same counter value of 5. It will then send a token packet back to i with number 5.

We also do not allow protocol P to send packets at node i if node i is waiting for a token packet on one of its links. This implies that (in good executions) any packet sent by i after i has locally reset is sent after j is at the same counter value as i; thus this packet will be accepted by j.

The formal code for this protocol is described in Figure 8. Once again, we defer the proof of correctness and stabilization to section 6. We will show there that the reset protocol stabilizes in three round trip delays.

5.1. Comparison with other reset protocols. We augment the protocol described so far to allow any node to make a reset request as follows. Each node has a reset request bit that is set when the node gets a reset request, or when it has received a *Request* packet from its children. When a node's request bit is on, it periodically sends a *Request* packet to its parent. When the root gets a *Request* packet, the root treats it like a REQUEST_RESET action. On doing a local reset a node clears its request bit; each node i also ignores reset requests and *Request* packets while $Finished(i) = false$.

The resulting reset protocol is similar to a stabilizing reset protocol due to Arora and Gouda [AG94] but has some important differences. First, the protocol [AG94] is based on a shared memory model and thus only requires node counters of size 2. In a

A token packet is encoded as a tuple $(Token, c)$ where c is an integer in the range $0..Max$
The state of each node i consists of:
an integer $count_i$ in the range $0..Max$, and a flag $token\_expected_i[j]$ for each neighbor j of i:
$token\_expected_i[j]$ is always *false* if $j = parent_i$
We assume $parent(i)$ points to i's parent, the root is node 0, and addition of counters is mod Max.

Finished(i) (* boolean function used by action routines below *)
(*set to true when not expecting tokens from any nonparent links *)
Return true if for all neighbors $token\_expected_i[k] = false$

ROOT_START (* root receives request to reset Protocol P *)
Effects:
if *Finished*(0) then (* ignore request if finishing current reset *)
$count_0 = count_0 + 1$ (*choose new counter value mod Max*)
LOCAL_RESET(0) (* locally reset Protocol P *)
For all neighbors k of root, $token\_expected_0[k] = true$ (* expect an ack from all neighbors*)

SEND$_{i,j}$($Token, c$), (* node i sends token to node j *)
Preconditions: (* retransmit periodically regardless of ack bit *)
$c = count_i$ (* counter of token matches node counter *)
$j \neq parent(i)$ or ($j = parent(i)$ and *Finished*(i)) (* send to children, and to parent if finished *)

RECEIVE$_{j,i}$($Token, c$) (* node i receives token from node j *)
Effects:
If $j = parent_i$ and $c \neq count_i$ then (* new counter from parent*)
$count_i = c$ (* set value to counter in token packet*)
LOCAL_RESET(i) (* locally reset Protocol P *)
For all neighbors $k \neq j$ of i (* don't expect ack from parent *)
$token\_expected_i[k] = true$ (* set to true when expecting a token packet *)
Else if $count_i = c$ then
$token\_expected_i[j] = false$ (* treat as a valid ack from neighbor j*)

RESET_FINISHED$_0$ (* root reports finishing of reset *)
Preconditions:
Finished(0)

Protocol P packets are only sent at node i when *Finished*(i) is *true* and are tagged with $count_i$.
A protocol P packet M received at node i is relayed to the application iff the tag of M is equal to $count_i$.
Any action that is continuously enabled for 1 unit of time occurs in 1 unit of time.

FIG. 8. *Simple reset protocol using counter flushing. The code is explained in more detail in the main text of the paper.*

message passing model, where each link can store L_{max} counter values, we believe that larger counter values, as in our protocol, are necessary. A second difference between

our protocol and the one in Arora–Gouda is the use of "delayed acks" and flushing of cross links. This is unnecessary in [AG94], because protocol P is modified so that a node does not read the state of its neighbors unless they have the same counter value. This is possible in a shared memory model but not in a message passing model.

There is also the stabilizing reset protocol of [APV91] which is in turn based on the nonstabilizing reset protocol of [AAG87]. However, this protocol takes $O(n)$ time to stabilize which is slower than our reset protocol or the Arora–Gouda protocol. [AKM$^+$93] suggests making this protocol faster by running it over a spanning tree, but in that case much of the complexity of that protocol is not needed. The fast and lean reset protocol of [IL94] does a reset in constant space. We believe that a 32-bit counter is adequate for most networks, and hence the requirement for logarithmic space in our protocol is not a problem for practical protocols. Our reset protocol is also much simpler.

6. General proofs. In this section, we present proofs of stabilization and correctness for the three protocols (token ring, PIF, and reset) described in Figures 4, 5, and 8, respectively. The three protocols seem different, work on different topologies, and have different objectives. Despite this, we will describe a uniform stabilization proof for all three protocols. In a few places, we distinguish between the ring system (Figure 4) and the tree systems (the PIF and reset protocols, Figures 5 and 8). Note that we added an extra ROOT_START action to the ring system (strictly not needed) and also used the ROOT_START action name in the reset protocol (instead of a more appropriate name) for uniformity of proof.

The proof is structured in four subsections: section 6.1 defines useful terminology; section 6.2 describes legal states; section 6.3 contains a proof that all three protocols stabilize quickly to legal executions; finally, section 6.4 shows that all three protocols exhibit the desired properties in legal executions.

6.1. Definitions. We have already defined the parent of a node for tree systems. For a ring, define the parent of node i to be node $i-1$. Define the *parent path* of a node i to be the sequence of nodes $i_0, i_1, i_2, \ldots, i_l$ such that $i_0 = 0$, $i_l = i$, and the parent of i_m is equal to i_{m-1} for $m = 1, \ldots, l$. Define the links in a parent path $i_0, i_1, \ldots, i_l$ to be the links $(i_0, i_1), \ldots, (i_{l-1}, i_l)$. Notice that the links in the parent path are the links directed "downwards" from the root and do not include any "upward" or "cross" links.

Define the counter at a node i to be $count_i$. We use *root counter* to denote $count_0$, the counter at the root node 0. We say that the counter in packet m is c if m is of the form $(Token, c, *)$. We say that packet m' is behind packet m in link (i, j) in some state s if m and m' are both stored in $s.Q_{i,j}$ and m is closer to the head of $Q_{i,j}$. Recall from Figure 2 that $Q_{i,j}$ models the queue of packets that represent undelivered packets on link (i, j).

To describe the legal states we will define the notion of counters upstream from a node or packet m. Intuitively, these are counters stored on the parent path that leads to m.

Let s be any state of the tree or ring systems. Formally, define the *counters upstream* from node m in state s to be the set of counters in all nodes (including m) and links in the parent path of m in state s. If packet m is stored on link (i, j) in state s, then the *counters upstream* from packet m is the union of the set of counters in packets behind m in link (i, j), together with the set of counters upstream from node i in state s. We will often refer to the counters upstream from m without reference to state s if it is clear from context what state s is.

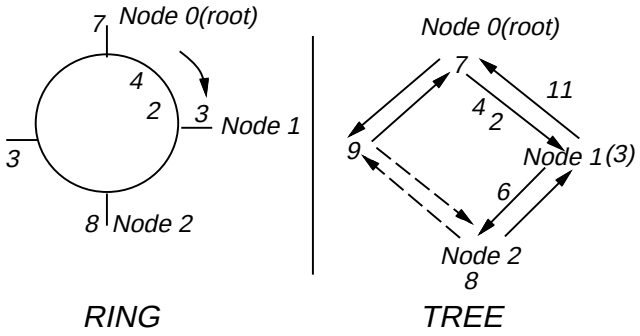


FIG. 9. *Examples of parent paths and upstream counters in a ring and tree. The numbers at nodes represent node counters and the numbers attached to a link represent the counter in a* $(Token, *)$ *packet stored on that link.*

Figure 9 gives examples of these definitions for a ring system (left) and a tree system (right). Notice first that the ring system has a sequence of unidirectional links oriented clockwise in the picture. However, the tree system has a pair of unidirectional links between every pair of neighbors. In the case of the reset protocol, the tree system may include cross links (shown dashed) between neighbors such that neither is the parent of the other. The numbers at nodes and links represent node and packet counters. Thus in both states, link $(0, 1)$ has two stored packets, the first with counter 2 and a second behind the first with counter 4. In both states, node 2 has node counter 8.

The parent path of node 2 is the sequence $0, 1, 2$ in both pictures. The links in the parent path of node 2 are $(0, 1)$ and $(1, 2)$ in both pictures. In the state on the left of Figure 9, the counters upstream from node 2 is the set $\{8, 3, 2, 4, 7\}$. In the state on the right, the counters upstream from node 2 is the set $\{8, 6, 3, 2, 4, 7\}$. Notice that the set does not contain 11 as the packet containing 11 is not in the parent path of node 2. In both states the counters upstream from the packet containing counter 2 is the set $\{4, 7\}$.

6.2. Legal state definitions. We first describe a one-band property that holds in legal states; this property is illustrated in state C of Figure 3 for a ring system. Notice that there is a band of counters starting from the root extending to east and a token packet on the link from east to south, all of which have counter equal to 8. Notice that the remaining counters in the ring are not equal to 7, which is what we would expect in a legal state. Thus, state C satisfies the one-band predicate but not the legal state predicate, both of which are defined below. Figure 6 illustrates a legal state for the PIF protocol. Notice a single band of values equal to 13 starting from the root and extending to the right child of the root. Even if the other node counters were arbitrary, this would suffice to satisfy the one-band predicate.

DEFINITION 6.1. *We say that a state satisfies the* one-band *property if the following five predicates hold in that state:*

- O1: *If there is a packet or node* m *with counter equal to the root counter, then all counters upstream of* m *are equal to the root counter.*
- O2: *If Finished*(0) *is true, then all counters are equal to the root counter.*
- O3: *(tree systems only) If a packet* m *is on link* (i, j) *with counter equal to the root counter and* j *is the parent of* i*, then Finished*(i) *is true and*

$count_i = count_0$.

- O4: *(tree systems only) If* $token\_expected_i[j] = false$ *and* $count_i = count_0$ *and* i *is the parent of* j*, then* $Finished(j) = true$.
- O5: *(tree systems only) If* $token\_expected_i[j] = false$ *and* $count_i = count_0$*, then all counters in link* (j, i) *and* $count_j$ *are equal to the root counter.*

The following lemma states that once the one-band property begins to hold in any execution of any system, it continues to hold. It can be verified easily by checking all possible transitions from such a state.

LEMMA 6.2. *For all three systems, if the one band predicate holds in some state* s_i *of any execution* E*, then the one band predicate holds in all subsequent states* $s_j, j > i$ *of* E.

We now define legal states to to be those in which the one-band property holds and all counters not equal to the root counter are one less than the root counter. For tree systems, we also require that if a node counter is not equal to the root counter, then it is not expecting any acknowledgements. For PIF systems, we also require a value correspondence property. Recall that addition and subtraction of all counters is always implicitly mod *Max*.

DEFINITION 6.3. *We say that a state* s *of any of our three systems is a legal state if*

- L1: s *satisfies the one band property;*
- L2: *any counter that is not equal to the root counter* c *is equal to* $c - 1$*;*
- L3: *(tree systems only) if* $count_i$ *is not equal to* $count_0$*, then* $Finished(i) = true$*;*
- L4: *(PIF systems only) for all* j, k *where* j *and* k *can either be packets or nodes, if the counter associated with* j *is equal to the counter associated with* k*, then the two associated values are the same.*

We will refer to L4 as the *value correspondence* property. The ring and reset systems trivially satisfy value correspondence in all states. The following lemma is easily checked by examining all possible transitions from a legal state.

LEMMA 6.4. *For all three systems, if the legal state predicate holds in some state* s_i *of any execution* E*, then the legal state predicate holds in all subsequent states* $s_j, j > i$ *of* E.

6.3. Stabilization proof. Define a *home* state for all systems to be a state in which all counters in nodes and packets are equal and $Finished(i) = true$ for all nodes i. The following lemmas are easy to check using the definitions.

LEMMA 6.5. *Any home state that satisfies value correspondence is a legal state.*

LEMMA 6.6. *Any state that satisfies the one band property and has* $Finished(0) = true$ *is a home state.*

Formally, define a *fresh* state for all systems to be a state in which all counters in packets and nodes other than the root are *not* equal to the root counter and $Finished(0) = false$. The following lemma is easy to check from the definitions.

LEMMA 6.7. *Any fresh state satisfies the one band property.*

Let the height h of the system be the maximum length of a parent path. Clearly $h = n - 1$ for the ring. Let R, the round trip delay of a system, be $2h + 2$ for the ring and $4h + 2$ for the tree systems. Intuitively, R represents the maximum time for information sent by the root to causally flow to all nodes in the system and then flow back to the root.

The following and subsequent lemmas are stated in terms of time complexity results. They can be translated to liveness results (which do not require the time

complexity assumptions we made in our model, and only require standard fairness assumptions) by replacing "within time X" by "eventually."

LEMMA 6.8. *Within R time of any state s_I, either the root counter will change or the protocol will enter a home state.*

Proof. Let s_F be a state after s_I such that R units of time have elapsed from s_I to s_F and such that $count_0$ has not changed in the interval $[s_I, s_F]$. We will show that s_F is a home state. Let h_i be the length of the parent path of node i in all three systems. It is easy to show by induction that within time $2h_i$ of s_I each node with height h_i will set $count_i = c$ and $count_i$ will remain unchanged till state s_F. Intuitively this is because any packet on a link is delivered within 1 time unit, each node accepts any value sent to it by its parent, and each node retransmits a new counter value to its children in 1 unit of time.

Thus in time $2h$, all nodes will have their counter values equal to c and will remain with this value to the end of the interval. For the ring system, in time $2h+2$, the root will receive a packet with counter equal to c and the system will enter a home state.

For a tree system, the argument is slightly longer. In time $2h + 2$, each node will receive a token packet numbered c on all its "cross" links and thus will set the *token_expected* flag to *false* for such links. Thus by time $2h + 2$, all leaves l will have *Finished*(l) = *true*, all nodes and token packets will have counter value c, and $token\_expected_i[j]$ = *false* for all "cross" links (i, j). Let h'_i be the maximum length of a path from a leaf in the subtree rooted at node i to node i. Then it is easy to see, by a similar induction on the height, that within time $2h + 2 + 2h'_i$ all nodes i will have *Finished*(i) = *true* and this will remain true till s_F. Thus within time $4h + 2$, *Finished*(0) = *true*. Thus s_F must be a home state. □

LEMMA 6.9. *A home state will occur within R time of a fresh state.*

Proof. Consider an execution fragment beginning with a fresh state s_I. Now within R time either the root counter will not increment (Case 1) or it will (Case 2). Consider Case 1. In that case the root counter does not increment within R time and so we must reach a home state by Lemma 6.8. So consider Case 2. Suppose the root counter increments for the first time in some state s_F that occurs within R time after s_I. We know from Lemma 6.7 that s_I satisfies the one band property as a fresh state. We know from Lemma 6.2 that all states after s_I satisfy the one band property. Thus we know that states s_F and s_{F-1} satisfy the one band property. We see from the code of all three systems that we cannot increment the root counter unless *Finished*(0) = *true*. Thus $Finished_0$ = *true* in s_{F-1}. But by Lemma 6.6, state s_{F-1} must be a home state because s_{F-1} is fresh and has *Finished*(0) = *true*. Thus s_{F-1} is a home state that occurs within R time of s_I and we are done. □

We say that the root counter *wraps around* in an execution fragment $s_I, \ldots, s_F$ if we have $s_J.count_0 = s_I.count_0 - 1$ for some J in $[I, F]$.

LEMMA 6.10. *Any execution fragment in which the root counter wraps around must contain a fresh state.*

Proof. In state s_I there are at most c_{max} counters. Since *Max*, the modulus of the counter space, is strictly greater than c_{max}, there must be some counter value f that is not present in any node or packet in the first state s_I. Since the root counter wraps around in the interval $[s_I, s_F]$ and the root counter only changes by incrementing (mod *Max*), there must be some intermediate state in the interval $[s_I, s_F]$ in which the root counter is equal to f. Let s_J be the first such state. It is easy to see that since the value f was not present in state s_I it is not present in any state in the interval $[s_I, s_{J-1}]$. This is because, in all our systems, only the root produces new counters.

Thus in state s_J only the root changes its counter value to f and sets *Finished*(0) to *false* (because the action that takes us to state s_J must be a *RootStart* action). Thus s_J is a fresh state. □

Recall that we defined the parent of a node i for the ring system to be node $i-1$. We define the parent link of a node i to be a link (j, i) such that j is the parent of i. We define a node or packet m to be causally connected (to the root) in an execution fragment E that ends with state s if either

- m is the root;
- m is a node and there is some state that occurs before s in which node m receives a causally connected packet on its parent link;
- m is a packet which was sent by some node i in some some state that occurs before s in which node i was causally connected.

LEMMA 6.11. *Consider any execution fragment $s_I, \ldots, s_F$. Suppose node or packet i is causally connected at the end of this execution fragment. Then the counter associated with i is contained in the sequence $s_I.count_0, s_{I+1}.count_0, \ldots, s_F.count_0$.*

Proof. We use induction on execution fragment length. The lemma is obviously true in the initial state of an execution fragment because only the root is causally connected and the lemma is clearly true for root. So consider any action that extends the last state of the fragment from say s_J to s_{J+1}.

If this action is the receipt of a packet m by node i, and m is not received on a parent link, then the counter of i will not change and so the lemma remains true if it was true in state s_J. If, however, it is received on a parent link, and the packet was causally connected, then after the receipt, node i is causally connected and changes its counter to the counter c associated with m. But since m was causally connected, by inductive hypothesis $c \in s_I.count, s_{I+1}.count, \ldots, s_J.count$. Thus $s_{J+1}.count_i = c \in s_I.count, s_{I+1}.count, \ldots, s_{J+1}.count$. A similar argument holds for the sending of a packet i by a causally connected node. The only other event is ROOT_START after which the lemma clearly holds for the root and trivially holds for all other nodes whose counters remain unchanged. □

LEMMA 6.12. *In any execution fragment, every node and packet will be causally connected within R time of the start of the fragment.*

Proof. Let h_i be the length of the parent path of node i in all three systems. It is easy to show by induction that within time $2h_i$ each node with height h_i will be causally connected. This follows because once a node becomes causally connected, it sends a message to each child which arrives at most 2 time units later, causing the child to be causally connected. Thus all nodes will be causally connected by time $2h$ in, say, state s_J. Let s_F be first state after s_J in which all packets stored in links in s_J are delivered. Also, since all packets in links in s_F must have been sent after s_J, all packets in s_F are also causally connected. The lemma follows because $2h < R - 1$ for all three systems, and because packet delivery takes at most 1 time unit. □

LEMMA 6.13. *If an execution fragment contains a state s which is causally connected, then state s and all subsequent states satisfy value correspondence.*

Proof. We use induction on execution length using the following inductive hypothesis. For all causally connected j, k where j and k can either be packets or nodes, if the counter associated with j is equal to the counter associated with k, then the two associated values are the same. Once all nodes and packets are causally connected the lemma follows from the hypothesis. The basis is true in the initial state as the root is causally connected. For the inductive step, if the action that extends the last state is the sending of a packet k by causally connected node i, if the counter of k is equal to

some other j then the counter of i is equal to k, and thus the value of k is equal to the value of i which is equal to the value of k. A similar argument can be made for the reception of a causally connected message by a node i. □

LEMMA 6.14. *Within $2R$ time of the start s_I of any execution fragment E, we will reach a fresh state or a home state that satisfies value correspondence.*

Proof. Let s_F be first state after s_J in which all packets and nodes are causally connected. From Lemma 6.12, s_F occurs within R time of s_I. By Lemma 6.13, state s_F and all subsequent states in the execution fragment satisfy value correspondence. If the root counter has wrapped around in $[s_I, s_F]$ we are done by Lemma 6.10. So assume the root counter has not wrapped around. Let $c = s_F.count_0$. Thus $c+1$ is not in the sequence $s_I.count, s_{I+1}.count, \ldots, s_F.count$. But by Lemma 6.11 all nodes and packets in states after s_J have counters in the sequence $s_I.count, s_{I+1}.count, \ldots, s_F.count$. Thus we know that as soon as the root counter first increments to $c+1$ we are in a fresh state. But we know that such a state must occur within R time of s_F by Lemma 6.8 or we will reach a home state. Since s_F occurs within R time of s_I the lemma follows. □

LEMMA 6.15. *A home state that satisfies value correspondence occurs within $3R$ time of any state.*

Proof. By Lemma 6.14, within $2R$ time we reach a home state or a fresh state that satisfies value correspondence. By Lemma 6.9, within R time of a fresh state we reach a home state. Thus within $3R$ time we reach a home state that satisfies value correspondence. □

So far we have not defined the legal executions of any of the three systems. By Theorem 6.16, we know that all three systems stabilize to a home state in $3R$ time, and by Lemma 6.5, we know that such a home state is a legal state. Thus it makes sense to define the legal executions of all three systems as the executions that begin with a home state that satisfies value correspondence. An immediate corollary to this definition and Lemma 6.15 is the following.

THEOREM 6.16. *The token ring, PIF, and reset systems all stabilize in $3R$ time.*

6.4. Correctness after stabilization. We see from Theorem 6.16 that all three systems stabilize to legal executions in $3R$ time. We now show that each legal execution results in correct behavior. Notice that by Lemma 6.15 we can partition a legal execution into fragments that start and end with a home state that satisfies value correspondence. We start by understanding the structure of such fragments.

Define a *fresh counter interval* to be an execution fragment[5] E such that

- The first state in E is a home state that satisfies value correspondence.
- The first action in E is a ROOT_START event.
- The second state (i.e., the state following the ROOT_START event) is fresh.
- The last state in E is the first state in E (other than the first state) in which $Finished(0) = true$.

The value of interval E is defined to be the value of $count_0$ in the second (fresh) state in E. For any execution $s_0, a_1, s_1, \ldots$ we can denote an execution fragment by its first and last state indices $[I, F]$ where s_I is the initial state and s_F is the final state.

For a fresh counter interval $[I, F]$ with value c we make the following definitions:

- Let $I(j)$ be the index of the first state in $[I, F]$ such that $count_j = c$.

[5] An execution fragment is a portion of an execution that begins and ends with a state.

- Let $L(j,k)$ be the index of the first state after $I(j)$ which follows the sending of a packet from j to k.
- For tree systems, let $F(j,k)$ be the index of the first state such that $count_j = c$ and $token\_expected_j[k] = false$.
- Let $F(j)$ be the index of the first state such that $count_j = c$ and $Finished(j) = true$.

We now prove some simple facts relating these definitions that are key to correctness.

LEMMA 6.17. *For any fresh counter interval $[I,F]$, every node j, and every neighbor k of j, we have the following.*

- *The states $I(j)$, $L(j,k)$, $F(j,k)$, and $F(j)$ exist.*
- *$L(j,k) < I(k)$ if j is the parent of k (i.e., a node's counter value cannot change until its parent sends it the new counter value).*
- *In the interval $[I(j),F]$, $count_j = c$ (i.e., the value of a node's counter remains unchanged from the time it is initiated till the end of the interval.)*
- *$I(k) < F(j,k) \leq F(j)$ (i.e., a node cannot finish until each of its neighbors is initiated).*
- *$F = F(0)$ is a home state.*

Proof. We use the fact that any home state that satisfies value correspondence is a legal state. Since legal states are stable (Lemma 6.4), every state in the interval $[s_I, s_F]$ is a legal state and satisfies the predicates O1, O2, O3, O4, and O5.

- We know that $Finished(0) = true$ in s_I and s_F. Thus we know (from O2 and O4 applied repeatedly in s_F) that all node counters must be equal to the root counter and $Finished(j) = true$ for all nodes j. We also know that the first action causes the root counter to increment to c. Thus $I(j) = I + 1$ if j is the root. Also for all nodes j other than the root, $count_j$ is not equal to c. But in s_F, $count_j$ is equal to c. Thus there must be some first state $I(j)$ in the interval $[s_I, s_F]$ in which j first changes to c. It is easy to see that $L(j,k)$ must exist because j will eventually send a packet to neighbor k after $I(j)$ in any execution. Also in state $I(j)$, since j changes its counter value, it is easy to see from the code that $token\_expected_i[j]$ is true. But in state s_F, $token\_expected_i[j]$ is false. Thus there must be some intermediate state $F(j,k)$ in which $token\_expected_i[j]$ first becomes false. Similarly, there must be a first state $F(j)$ in which $F(j,k)$ first becomes true for all neighbors k of j.
- In the state preceding $I(j)$, we conclude from O1 that $count_k$ is not equal to c, and there are no counters equal to c in link (j,k). Since $L(j,k)$ is the first state after $I(j)$ in which j sends a packet numbered c on link (j,k), there can be no packets numbered c in the interval $[I(j), L(j,k)]$. Since $count_k$ was not equal to c in $I(j)$ and can only change its counter value by receiving a packet numbered c from its parent j, it follows that $L(j,k) < I(k)$.
- It is easy to see that the root cannot change its counter after s_{I+1} because the root (see code in Figure 8) cannot increment unless $Finished(0) = true$ and s_F is the first state after s_{I+1} in which $Finished(0) = true$. If a node j other than the root changes its counter value after $I(j)$ to some value other than c, it means it received a counter not equal to c on its parent link. By O1, this implies that the root counter is not equal to c, a contradiction.
- The state s that precedes $F(j,k)$ must be (see code in Figure 8) the receipt of a packet with counter equal to $count_j = c$ on link (k,j). Thus by O1,

$count_k = c$ in state s. Thus $I(k) < F(j,k)$. Also $F(j) = \text{Max } F(j,k)$ over all neighbors k of j. So $F(j,k) \leq F(j)$.

- This follows immediately from O2 in state s_F. □

Armed with this lemma, we now show correctness separately for all three systems. Recall that any legal execution can be partitioned into fresh counter intervals. Thus to show correctness, we need only show correctness for a fresh counter interval.

THEOREM 6.18 (token ring correctness). *In any legal execution of the token ring system, at most one node has the token in any state and every node will receive the token infinitely often.*

Proof. We say that a node j has the token starting from any state s in which node j changes its counter value up to the first state after s in which node j sends a packet to $j+1$. Since it is sufficient to show correctness for a fresh counter interval within a legal execution, consider one such interval. It follows from Lemma 6.17 that $I(j) < L(j,j+1) < I(j+1)$ for $j = 0, \ldots, n-1$. Thus, $[I(j), L(j,j+1)]$ is disjoint for all j. Thus, at most one node has the token in any state. Similarly, we know from Lemma 6.17 that $I(j)$ exists for all j and so every node j receives the token during a fresh counter interval. It follows from Lemma 6.15 and the fact that the ROOT_START event is always enabled in a home state that the token system has an infinite number of fresh counter intervals. Thus every node receives the token infinitely often. □

THEOREM 6.19 (PIF correctness) *In any legal execution of the PIF system,*

- *In any state s, v_j is equal to either v_0 or the previous value of v_0.*
- *If the value of some node is not equal to v_0 in state s, there is a later state in which all node values are equal to v_0.*
- *Once a node j's value is equal to v_0, its value cannot change until we reach a state in which all node values are equal to v_0.*

Proof. First, it is easy to see that the ROOT_START event is enabled in a home state and will cause the value of the root counter to change. Thus every legal execution will have an infinite number of home states.

The first part follows from **L2** in the definition of a legal state and the fact that the counter associated with the previous value of v_0 must be $count_0 - 1$. The second part of the theorem follows because we know from Lemma 6.15 that a home state will occur in $3R$ time after state s; in this home state $count_j = count_0$ for all nodes j. Thus by value correspondence (**L4**), $v_j = v_0$ for all j. The third part of the theorem follows from value correspondence and the third statement in Lemma 6.17 which says that a node counter cannot change again until after the next home state. Thus by the code, its value will also remain unchanged in this interval. □

THEOREM 6.20 (reset protocol correctness) *In any legal execution of the reset system,*

- *Once the protocol is in a home state, it remains in a home state until the next reset request, and no node will perform a local reset in this interval.*
- *Consider any reset request that occurs when the reset protocol is in a home state. Then the reset protocol will enter a home state in $O(R)$ time after this reset request and in this home state, the underlying protocol P is in a legal state.*

Proof. The first part follows easily from the code and the definitions of a home state and a fresh counter interval. Notice that when the reset protocol is in a home state, it is impossible for a node j to receive a $(Token, c)$ packet with $c \neq count_j$; thus (from the code) j will never perform a local reset. We now turn to the second part of the theorem.

We know that any reset request that begins in a home state will result in the root picking a fresh counter value, say c, which begins a fresh counter interval. We know from Lemma 6.15 that within $O(R)$ time, this fresh counter interval will end. Thus from Lemma 6.17 if this interval is denoted by $[I, F]$, then there is a state $I(j)$ for each node j at which the node is initiated into the current reset computation. From the code it is easy to see that in this state protocol P is locally reset, and since $count_j$ remains at c this means that there are no further local resets of protocol P at node j.

To show that protocol P is properly reset at the end of the fresh counter interval, we have to show that for any two neighbors j, k the sequence of packets received by k from j during the interval $[I(k), F]$ is a prefix of the sequence of packets sent by j during $[I(j), F]$. Let us call the interval $[I(j), F]$ the reset interval at node j.

So consider any packet m sent by j during the interval $[I(j), F]$. From the protocol code, we know that j does not send any packet during the interval $[I(j), F(j)]$. So we can assume that m is sent after $F(j)$. Thus m will be tagged with c, the value of this fresh counter interval. Now by state F, we know from the properties of link automata that m either will be delivered by state s_F or is stored on link (j, k) in state s_F. If m is delivered, m must have been delivered after $F(j)$ (since it was sent after $F(j)$) and hence by Lemma 6.17 it is delivered in the interval $[I(k), F]$; but in this interval, $count_k = c$ and so m is accepted. On the other hand, if (in state F) m is stored on link (j, k), we know (because the link is FIFO) that all packets sent after m are not delivered. Thus, applying this argument to all packets sent by j to k during $[I(j), F]$, we see that if m is received and accepted, then all packets sent before m in $[I(j), F]$ are received and accepted by k; but if m is not received, then all packets sent after m in $[I(j), F]$ are not received.

All that remains is to show that any protocol P packet m received and accepted by k in $[I(k), F]$ was sent by j in $[I(j), F]$. But if m was accepted it must have tag c. Thus m must have been sent in $[I(j), F]$; this is because, by definition, any protocol P packets sent by j in $[I, I(j) - 1]$ must have a counter value $c' \neq c$. Recall that $I(j)$ is the first state in $[I, F]$ that has $count_j = c$. □

7. Conclusions. Counter flushing is a simple paradigm that has a range of applications and can be used over different topologies. Besides the examples discussed in this paper (token passing, broadcast, and reset), counter flushing can be used to design stabilizing protocols for deadlock detection and snapshot protocols as described in a preliminary version of this paper [Var94]. Our paper exploits a connection between seemingly different protocols such as Dijkstra's token ring protocol [Dij74a], Afek and Brown's data link protocol [AB93], Segall's propagation of information with feedback protocol [Seg83], and Arora and Gouda's global reset protocol [AG94]. At one level, they can all be regarded as repeated versions of a centralized total algorithm [Tel89]; at another level, the data link and reset problems can be regarded as *synchronization* problems whose correctness can be formalized in terms of a mating relation [AE83, Spi88, Var93]. The unified approach allowed us to provide a single unified proof.

Counter flushing has three aspects. First, we establish the presence of a nonexistent counter based on bounding the space of counters; second, we argue a liveness property that guarantees that deterministic incrementing (randomized choosing also works trivially) will lead quickly to a unique counter; third, we show a flushing condition to show that a nonexistent counter flushes out all bad values. While other papers (e.g., [Dol94, AK93]) do use randomization to choose a nonexistent counter, they do not need or use the other two aspects of counter flushing.

Local checking and correction is another general paradigm [APV91, Var93, AGV94]

for designing stabilizing protocols. On a theoretical level, there are problems for which counter flushing is applicable but local checking is not (e.g., token passing on a ring), problems for which local checking is applicable but counter flushing is not (e.g., synchronizers), and problems where they are both applicable (e.g., resets). We believe that while both techniques are practical, counter flushing is simpler to implement. Local checking [APV91, Var93] requires enumeration of predicates and the addition of periodic local snapshots and resets.

We have generalized counter flushing to apply to sliding window protocols [CV96] in order to allow more than one packet in transit between the root node and the receivers. We have also applied counter flushing to design a real token passing protocol in [CV98]. Our modified FDDI protocol [CV98] recovers from multiple tokens in less than 5.7 ms, while the existing FDDI protocol may never recover from multiple tokens; also, the modified FDDI recovers from lost tokens more quickly than FDDI (0–0.36 ms versus 2.5–4.1 ms). This fits in with our overall goal which is to design theoretical techniques that can help design robust practical protocols.

REFERENCES

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni, *Applying static network protocols to dynamic networks*, in Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 358–370.

[AB93] Y. Afek and G. Brown, *Self-stabilization over unreliable communication media*, Distrib. Comput., 7 (1993), pp. 27–34.

[AE83] B. Awerbuch and S. Even, *A Formal Approach to a Communication-Network Protocol; Broadcast as a Case Study*, Technical Report TR-459, Electrical Engineering Department, Technion, Haifa, Israel, 1983.

[AG94] A. Arora and M. Gouda, *Distributed reset*, IEEE Trans. Comput., 43 (1994), pp. 1026–1038.

[AGV94] A. Arora, M. Gouda, and G. Varghese, *Constraint satisfaction as a basis for designing nonmasking fault-tolerance,* in Proceedings of the 14th International Conference on Distributed Computing Systems, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 424–431.

[AK93] S. Aggarwal and S. Kutten, *Time optimal self-stabilizing spanning tree algorithm*, in Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 761, Springer-Verlag, New York, 1993, pp. 400–410.

[AKM+93] B. Awerbuch, S. Kutten, Y. Mansour, B. Patt-Shamir, and G. Varghese, *Time optimal self-stabilizing synchronization*, in Proceedings of the 25th ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 652–661.

[APV91] B. Awerbuch, B. Patt-Shamir, and G. Varghese, *Self-stabilization by local checking and correction*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, San Juan, PR, 1991, pp. 268–277.

[Cha82] E. Chang, *Echo algorithms: Depth parallel operations on general graphs*, IEEE Trans. Software Engrg., 8 (1982), pp. 391–401.

[CV98] A. Costello and G. Varghese, *The FDDI MAC meets self-stabilization*, in Proceedings of the Fourth IEEE Workshop on Self Stabilizing Systems, Austin, TX, 1999, pp. 1–9.

[CV96] A. Costello and G. Varghese, *Self-stabilization by window washing*, in Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, PA, 1996, pp. 35–44.

[Dij74a] E. Dijkstra, *Self stabilization in spite of distributed control*, Comm. ACM, 17 (1974), pp. 643–644.

[DIM91] S. Dolev, A. Israeli, and S. Moran, *Resource bounds for self stabilizing message driven protocols*, in Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1991, pp. 281–293.

[DIM93] S. Dolev, A. Israeli, and S. Moran, *Self-stabilization of dynamic systems assuming only read/write atomicity*, Distrib. Comput., 7 (1993), pp. 3–16.

[Dol94] S. Dolev, *Optimal time self-stabilization uniform dynamic systems*, in Sixth International Conference on Parallel and Distributed Computing and Systems, Washington, DC, 1994, pp. 25–28.

[Fin79] S. Finn, *Resynch procedures and a fail-safe network protocol*, IEEE Trans. Comm., COM-27 (1979), pp. 840–845.

[Gou94] M. Gouda, *Stabilizing observers*, Inform. Process. Lett., 52 (1994), pp. 99–103.

[IL94] G. Itkis and L. Levin, *Fast and lean self-stabilizing asynchronous protocols*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, Santa Fe, NM, 1994, pp. 226–239.

[KP93] S. Katz and K. Perry *Self-stabilizing extensions for message-passing systems*, Distrib. Comput., 7 (1993), pp. 17–26.

[LSP82] L. Lamport, R. Shostak, and M. Pease, *The Byzantine generals problem*, ACM Trans. Programming Languages and Systems, 4 (1982), pp. 382–401.

[LT89] N. Lynch and M. Tuttle, *An introduction to input/output automata*, CWI Quarterly, 2 (1989), pp. 219–246.

[MAM+90] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Sattenthwaite, and C. Thacker, *Autonet: A High-Speed, Self-Configuring Local Area Network Using Point-to-Point Links*, Technical Report 59, Digital Systems Research Center, Palo Alto, CA, 1990.

[Per83] R. Perlman, *Fault tolerant broadcast of routing information*, Comput. Networks, 7 (1983), pp. 395–405.

[Per88] R. Perlman, *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, Laboratory for Computer Science, MIT, Cambridge, MA, 1988.

[Ros81] E. Rosen, *Vulnerabilities of network control protocols: An example*, Comput. Commun. Rev., 1981.

[Sch93] M. Schneider, *Self-stabilization*, ACM Comput. Surveys, 25 (1993), pp. 45–67.

[Seg83] A. Segall, *Distributed network protocols*, IEEE Trans. Inform. Theory, 29 (1983), pp. 23–35.

[Spi88] J. Spinelli, *Reliable Communication*, Ph.D. thesis, Laboratory for Information and Decision Systems, MIT, Cambridge, MA, 1988.

[Tan93] A. Tannenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[Tel89] G. Tel, *The Structure of Distributed Algorithms*, Cambridge University Press, Cambridge, UK, 1989.

[Var93] G. Varghese, *Self-Stabilization by Local Checking and Correction*, Ph.D. thesis, Technical Report MIT/LCS/TR-583, Laboratory for Computer Science, MIT, Cambridge, MA, 1993.

[Var94] G. Varghese, *Self-stabilization by counter flushing*, in Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 244–253.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 511–527

THE POWER OF MIGRATION IN MULTIPROCESSOR SCHEDULING OF REAL-TIME SYSTEMS*

GILAD KOREN†, EMANUEL DAR‡, AND AMIHOOD AMIR‡

Abstract. In this paper we study the performance of off-line multiprocessor real-time schedules that allow task migration compared to those that forbid migration. We consider an *off-line* scheduling problem in which a given collection of tasks, each with a release time, computation time, and *deadline*, are to be run on a multiprocessor system. A preemptive schedule allows the execution of a task to be temporarily suspended and resumed at a later time. A *migrative* schedule allows the task to resume on any processor whereas a *nonmigrative* schedule allows the task to resume only on the processor in which it was initially started. A schedule *value* is the summation of all the values of all the tasks that were completed by their deadlines. In this paper we assume that a task's value is proportional to its computation time.

We present lower and upper bound results. For a system with n processors, we construct a nonmigrative schedule that is guaranteed to achieve at least $1-(1-\frac{1}{2n})^n$ of the *optimal* migrative schedule value.

In addition, we show task sets for which even an optimal nonmigrative schedule achieves at most $n/(2n-1)$ of the optimal migrative value. Asymptotically (as $n \to \infty$) our upper bound approaches $1/2$ and the lower bound approaches $1-\frac{1}{\sqrt{e}} \sim 0.3935$.

Key words. real-time, off-line scheduling, multiprocessor migration, deadline

AMS subject classifications. 68M20, 68N25, 68Q25, 93C83

PII. S0097539797326241

1. Introduction.

Real-time systems can be most generally described as systems that need to complete certain tasks within a specified time limit. Traditionally, these are systems that handle critical processes such as power stations, nuclear reactors, or space vehicles. In many such applications, failure of the task to complete in time may have disastrous results. Such systems are called *hard real-time systems*.

A less stringent type of real-time system is one where every task still has to be completed by a given deadline—it is then said to *succeed*; otherwise it *fails*. However, such a failure merely means that a certain stimulus has not been responded to; it does not have catastrophic connotations. An example may be a case where the stimuli arrive at a quick rate. Failure to respond to a number of them is still tolerable, as long as a certain ratio succeeds. In this paper we are concerned with the latter type of real-time systems, the *soft real-time systems*. Even within the realm of soft real-time systems, several models appear in the literature. We concern ourselves with the *firm real-time systems*. In this model each task has a *value* which is obtained by the system only if the task completes by its deadline.

A *scheduling algorithm* for a real-time system is presented with tasks, each of which has a *release time* from which it can start working, a *deadline* by which it must

*Received by the editors August 25, 1997; accepted for publication (in revised form) May 5, 1999; published electronically June 3, 2000.

http://www.siam.org/journals/sicomp/30-2/32624.html

†School of Mathematics and Computer Science, Netanya Academic College, 42365 Netanya, Israel, and Department of Mathematics and Computer Science, Bar-Ilan University, 52-900 Ramat Gan, Israel (koren@cs.biu.ac.il).

‡Department of Mathematics and Computer Science, Bar-Ilan University, 52-900 Ramat Gan, Israel (dar@cs.biu.ac.il, amir@cs.biu.ac.il). The research of the third author was partially supported by NSF grant CCR-96-10170, BSF grant 96-00509, and a 1999 Bar-Ilan University Internal Research grant.

complete working if it is to be successful, and a *computation time*, the time it actually needs in order to run in its entirety. Each task has a *value* obtained if completed successfully. The special case where the value of a task is proportional to its running time is called the *uniform value density* case [3, 10].

The aim of the system is to maximize the sum of the values obtained from all successfully completed tasks. It is the duty of the scheduler to schedule the tasks that maximize this total value.

A number of issues are raised at this point. A scheduler that knows in advance all the tasks and their parameters is called an *off-line* or *clairvoyant* scheduler. If it is presented with a task only upon its release, then the scheduler is called an *on-line* scheduler. The model is said to allow *preemption* if the scheduler may interrupt a working task. Various different types of preemption exist. A preempted task may proceed later from the point it was preempted, be forced to start from the beginning, or not be allowed to run again. In this paper we are concerned with real-time systems that allow preemption. A preempted task may later resume its execution where it left off.

A significant amount of work has appeared about the various possible models of real-time systems [5, 4]. In a single processor environment, it has been shown that in *underloaded* systems, i.e., where there exists a (possibly clairvoyant) schedule that allows all tasks to successfully complete, there exist on-line schedulers with 100% guarantee [6]. In the *overloaded case* such an optimal schedule is impossible [11]. Scheduling algorithms such as D^{over} give a competitive guarantee for overloaded firm real-time systems [10].

In this paper we are concerned with *multiprocessor* real-time systems with preemption. Dertouzos and Mok [7] showed that even in an underloaded system, no on-line algorithm can guarantee 100% success. A task is said to *migrate* if it continues running on a different processor from the one it was preempted in. Koren and Shasha [9] give a multiprocessor on-line competitive algorithm (MOCA) with a competitive guarantee for a multiprocessor real-time system with no *migration*. MOCA does not use migration but its guarantee holds even when migration is allowed. There is a number of differences between the case of MOCA and our problem. (1) MOCA is an on-line algorithm, while we study the off-line case. (2) MOCA does not offer any guarantees in the *uniform value density case*—where the value of a task is proportional to its running time. This is precisely the case we handle.

Bar-Noy et al. [1] study the the case of uniform value density, but in their model tasks have no slack time. (They must be scheduled upon release, if at all.) They give an upper bound of 0.66 on the best possible competitive factor. Bar-Noy, Mansour, and Schieber [2] devised on-line scheduling algorithms for this case with competitive factors of $\frac{1}{2} - \frac{1}{4n}$ for an even number of processors n and $\frac{1}{2}$ for an odd number of processors.

To our knowledge, *there is no on-line scheduler* for a real-time multiprocessor system *with migration* that has any nontrivial *competitiveness guarantees* for the uniform value density case of tasks with slack time.

We would like to investigate the power of migration. There are both practical and theoretical reasons for such an investigation. From a practical perspective, systems without migration are less centralized and less complicated. However, migration gives more power in processor utilization, in versatility, and in fault handling. It may also be easier to prove bounds on systems with migration.

This paper is a first step in the direction of understanding the power of migration

in real-time systems. Specifically, we would like to answer the following question: *In the uniform value density case, does a clairvoyant scheduler without migration have the same power as a clairvoyant scheduler with migration?* If we can pinpoint the *performance ratio* (as defined below) between off-line schedulers with migration and without, we will have gone a step forward in obtaining on-line schedulers as well. We will be able to limit ourselves to either the case of migration or nonmigration. Any competitive on-line scheduling algorithm that will be obtained will guarantee a competitive on-line scheduling algorithm for both cases.

DEFINITION 1.1. *A set of tasks S, in a system with n processors, has a* performance ratio *of $\omega_n(S) = \omega_n$ if*

$$\omega_n = \frac{\{value\ obtained\ by\ an\ optimal\ nonmigrative\ schedule\ for\ S\}}{\{value\ obtained\ by\ an\ optimal\ migrative\ schedule\ for\ S\}}.$$

An off-line nonmigrative scheduling algorithm $\mathcal{A}$, in a system with n processors, has a performance ratio *of $\omega_n(\mathcal{A}) = \omega_n$ if for all S*

$$\omega_n \leq \frac{\{value\ obtained\ when\ \mathcal{A}\ schedules\ S\}}{\{value\ obtained\ by\ an\ optimal\ migrative\ schedule\ for\ S\}}.$$

1.1. Main results. The contribution of this paper is two-fold. In the "grand view" of computer science we add our voice to the impressive body of work dealing with theoretical complexity analysis of operating systems. The operating systems area is at the heart of computer science and understanding the power and limitations of its various concepts is of fundamental importance to our field. It is not surprising that much work has historically been done in this area, particularly in job scheduling (see, e.g., [12, 8]).

However, the area of real-time systems, albeit important in practice, has been relatively neglected in the theory literature. Real-time systems grapple with different issues; thus, the scheduling literature does not generally provide answers to such problems. This paper is a modest addition to a growing number of works that deal with theoretical issues in real-time systems.

The more specific contribution of this paper is in providing both an upper and a lower bound on the performance ratio of off-line schedulers without migration versus with migration in a multiprocessor real-time system with n processors.

- We show an example where an off-line scheduler without migration cannot schedule more than $\frac{n}{2n-1}$ the value of the optimal schedule with migration.
- We describe a scheduler that without using migration obtains a value of at least $1 - \left(1 - \frac{1}{2n}\right)^n$ of the optimal value using migration.

Asymptotically (as $n \to \infty$) our upper bound approaches $1/2$ and the lower bound approaches $1 - \frac{1}{\sqrt{e}} \sim 0.3935$.

This paper is constructed as follows: We start with some notations and definitions in section 2. In section 4 we show the upper bound on the performance ratio. Section 5 is an overview of the scheduler's main ideas and proof of the lower bound on the performance ratio. Section 6 presents the algorithm that, inputting the system tasks' parameters, constructs a nonmigrative schedule with the necessary guarantees for section 5. We conclude with some open problems.

Note on terminology. We have defined an *off-line* scheduling problem. Therefore all tasks and their parameters are known from the start although the tasks are not ready to run until some specified time in the future which we denote here as *release time*. In describing our schedulers, the reader will notice occasional use of on-line-like

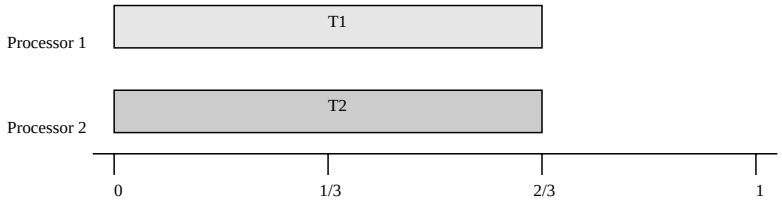


FIG. 1. *A nonmigrative schedule.*

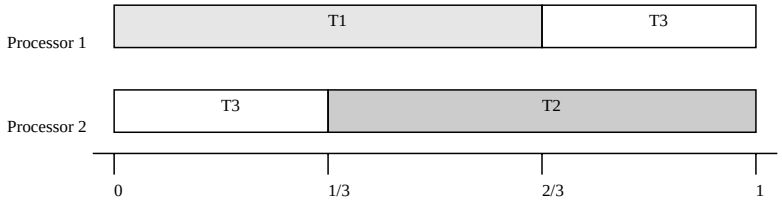


FIG. 2. *A migrative schedule.*

language: "*the task just released*," "*the new task*," etc. The reader should understand that these are just forms of expressions that we consider natural to describing a possible future situation considered by the scheduler. In reality the entire computation is done *off-line*.

2. Preliminaries: Notation and an example. We view time as the set of real nonnegative numbers R^+; however, the results are equally valid for Q^+ or N^+.

DEFINITION 2.1. *Let T be a task. We denote its parameters by $T\left(r\left(T\right),\ c\left(T\right),\ d\left(T\right)\right)$, where $r\left(T\right)$ is the* release time*, $c\left(T\right)$ is the* computation time*, and $d\left(T\right)$ is the* deadline.

A legal schedule *of a task set S on n processors is a schedule where each task T is scheduled for execution only between $r(T)$ and $d(T)$ for a total length that does not exceed $c(T)$. In addition, a task may execute on at most one processor at any given moment. Note that a schedule is legal even if some tasks fail. In a* legal nonmigrative schedule *there is the additional constraint that a task may not run on more than one processor.*

3. Example: Performance ratio. Consider a system with two processors ($n = 2$) and the following task set S of size three:

$$S = \left\{T_1\left(0,\ \frac{2}{3},\ 1\right),\ T_2\left(0,\ \frac{2}{3},\ 1\right),\ T_2\left(0,\ \frac{2}{3},\ 1\right)\right\}.$$

An optimal nonmigrative schedule will schedule $\{T_1,\ T_2\}$ to obtain a value of $c(T_1) + c(T_2) = \frac{2}{3} + \frac{2}{3} = 1\frac{1}{3}$ (see Figure 1). An optimal migrative schedule will schedule all three tasks obtaining a value of $c(T_1) + c(T_2) + c(T_3) = \frac{2}{3} + \frac{2}{3} + \frac{2}{3} = 2$ (see Figure 2). Hence, $\omega_2\left(S\right) = \frac{1\frac{1}{3}}{2} = \frac{2}{3}$.

4. The upper bound. We begin by demonstrating an upper bound on the performance ratio. The example below shows that for every number of processors n there are task sets that can be completely scheduled when migration is allowed; however, some tasks must fail when migration is not allowed.

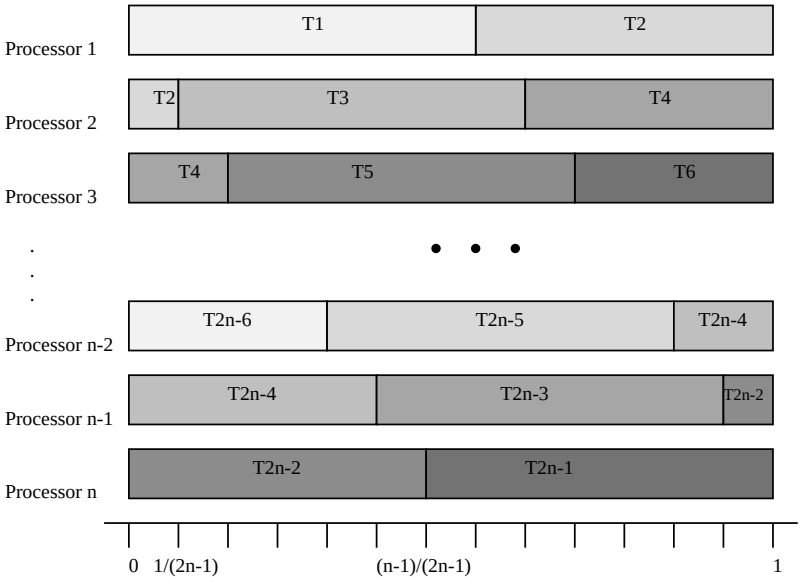


FIG. 3. *Upper bound example.*

THEOREM 4.1. *For any number of processors n, $n \geq 2$, there exists a task set, S^n, such that $\omega_n(S^n) = \frac{n}{2n-1}$.*

Proof. Let S^n be a set of $2n-1$ identical tasks: $T_i(0, \frac{n}{2n-1}, 1)$ (all with release time 0, deadline at 1, and computation time of $\frac{n}{2n-1}$).

An optimal off-line migrative schedule can successfully complete all those tasks (see Figure 3) and therefore obtains a value of

$$\sum_{i=1}^{2n-1} T_i = (2n-1)\frac{n}{2n-1} = n.$$

An optimal nonmigrative schedule completes at most one task on each processor for a total of n tasks obtaining a value of

$$n\frac{n}{2n-1} = \frac{n^2}{2n-1}.$$

The performance ratio is therefore

$$\omega_n(S^n) = \frac{\frac{n^2}{2n-1}}{n} = \frac{n}{2n-1}. \qquad \square$$

COROLLARY 4.2. *There is no off-line nonmigrative scheduling algorithm for n processors with performance ratio above*

$$\frac{\frac{n^2}{2n-1}}{n} = \frac{n}{2n-1}.$$

5. The lower bound. To prove the lower bound we will present the NO-MIGRATION algorithm—a scheduling algorithm that does not use migration.

Denote $1-\left(1-\frac{1}{2n}\right)^n$ by $\rho(n)$. The scheduler NO-MIGRATION receives as its input a task set and constructs a schedule that achieves at least $\rho(n)$ the value of the optimal migrative schedule. We will assume the existence of such a migrative schedule in order to prove that NO-MIGRATION indeed satisfies the ratio $\rho(n)$. However, a construction of such an optimal scheduler is not needed and it is not used by NO-MIGRATION.

Before describing Algorithm NO-MIGRATION we need one more definition.

DEFINITION 5.1. *Given a schedule σ on n processors, define $Proj\,(\sigma) \subseteq R^+$, the* projection *of σ, as follows: $t \in Proj\,(\sigma)$ iff there exists a task T that, by schedule σ, executes at time t on some processor.*

The idea of NO-MIGRATION is the following. Start with the set of tasks S and construct a subset R_1 that is schedulable on a single processor. The sum of the computation times of the tasks in R_1 is at least half the projection of an optimal migrative schedule of S. Delete the tasks of R_1 from S and repeat the process, constructing a set R_2 with the same properties (relative to the reduced S). Repeat this n times, to produce a nonmigrative schedule on n processors.

For clarity of exposition and proof, we assume the existence of two building blocks, EDF* and EDF-LST, that construct the sets $R_i,\ i = 1, \ldots, n$, with the desired properties. The details of these algorithms are described in the next section.

5.1. The NO-MIGRATION algorithm.

Algorithm Outline

INPUT: A set of tasks S.

1. $TASK\_SET \leftarrow S$
2. for $i = 1$ to n do
   (a) construct a subset $R_i \subseteq TASK\_SET$ (using the EDF-LST algorithm and the construction in section 7.2).
   /* We will see in section 6 that R_i satisfies
   - R_i is schedulable on one processor; and
   - The sum of R_i tasks' computation times is at least half of P_i, the projection of an optimal migrative schedule of $TASK\_SET$.

   */
   (b) $TASK\_SET \leftarrow TASK\_SET \setminus R_i$.
3. End For

End Algorithm

For each $i = 1, \ldots, n$, the tasks in R_i can run on a single processor. Thus the tasks in $\cup_{i=1}^{n} R_i$ constitute a nonmigrative schedule on n processors. For each $i = 1, \ldots, n$, the computation time of all the tasks in R_i is at least half the projection of an optimal schedule on the tasks in $TASK\_SET$ during the ith iteration. Theorem 5.3 proves that the total computation time of the tasks in $\cup_{i=1}^{n} R_i$ is at least $\rho(n)$ of the total computation time in an optimal migrative schedule.

Notation. Denote by V the *value* of a migrative schedule of task set S on n processors. Denote by $P_i,\ i = 1, \ldots, n$, the *projection* of an optimal migrative schedule on n processors of $TASK\_SET$ in iteration i of the NO-MIGRATION algorithm.

LEMMA 5.2. *For all $1 \le i \le n$,*

$$\sum_{j=1}^{i} R_i \ge \frac{1}{2n} \sum_{j=1}^{i} \left(\frac{2n-1}{2n}\right)^{j-1} V.$$

Proof. We prove this claim by induction.

For $i = 1$, $R_1 \geq \frac{1}{2n}\left(\frac{2n-1}{2n}\right)^0 V = \frac{1}{2n}V$. This is true since, by definition of the projection of a schedule, the length of the first projection, P_1, is not smaller than the busy time of any single processor. Hence $nP_1 \geq V$ and R_1's value is at least half of P_1's length. Therefore,

$$R_1 \geq \frac{P_1}{2} \geq \frac{1}{2n}V.$$

Assume the correctness of the claim for i; we have to prove it for $i+1$.

In step $i+1$, the total length of tasks that still remain to be scheduled is $V - \sum_{j=1}^{i} R_j$. The length of the $(i+1)$st projection, P_{i+1}, is greater than the busy time of any single processor. Hence, $nP_{i+1} \geq (V - \sum_{j=1}^{i} R_j)$. The value of R_{i+1} is at least half of P_{i+1}'s length. Therefore,

$$R_{i+1} \geq \frac{P_{i+1}}{2} \geq \frac{1}{2n}\left(V - \sum_{j=1}^{i} R_j\right) \tag{5.1}$$

and

$$\begin{aligned}
\sum_{j=1}^{i+1} R_j &= \sum_{j=1}^{i} R_j + R_{i+1} \\
&\geq \sum_{j=1}^{i} R_j + \frac{V - \sum_{j=1}^{i} R_j}{2n} && \text{(by inequality 5.1 above)} \\
&= \frac{2n\left(\sum_{j=1}^{i} R_j\right) + V - \sum_{j=1}^{i} R_j}{2n} \\
&= \frac{V + (2n-1)\sum_{j=1}^{i} R_j}{2n} \\
&= \frac{V}{2n} + \frac{(2n-1)}{2n}\sum_{j=1}^{i} \frac{1}{2n}\left(\frac{2n-1}{2n}\right)^{j-1} V && \text{(by the induction hypothesis)} \\
&= \frac{V}{2n} + \sum_{j=2}^{i+1} \frac{1}{2n}\left(\frac{2n-1}{2n}\right)^{j-1} V \\
&= \sum_{j=1}^{i+1} \frac{1}{2n}\left(\frac{2n-1}{2n}\right)^{j-1} V,
\end{aligned}$$

which concludes the induction. □

THEOREM 5.3. *Given a set of tasks S there exists a nonmigrative schedule for n processors obtaining a value of at least*

$$\rho(n)V,$$

where V denotes the value of an optimal nonmigrative schedule. Furthermore, such a schedule is produced by NO-MIGRATION.

Proof. The schedule created by NO-MIGRATION assigns the tasks of R_i to processor i. Since each of the R_i's is schedulable on one processor we obtain a legal schedule

TABLE 5.1
Lower and upper bounds for different processor numbers.

n	2	3	4	7	10	40	70	100	400	700	1000
Lower	.4375	.4213	.4138	.4047	.4013	.3954	.3946	.3942	.3937	.3936	.3935
Upper	.6667	.6000	.5714	.5385	.5263	.5063	.5036	.5042	.5006	.5004	.5003
Ratio between bounds	.656	.702	.724	.752	.762	.781	.784	.782	.786	.787	.787

on n processors that use no migration. Its total value is $\sum_{i=1}^{n} R_i$, which according to Lemma 5.2 satisfies

$$\begin{aligned}\sum_{i=1}^{n} R_i &\geq \sum_{i=1}^{n} \frac{1}{2n}\left(\frac{2n-1}{2n}\right)^{j-1} V \\ &= \frac{1}{2n}\left(\frac{\left(1-\frac{1}{2n}\right)^n - 1}{\left(1-\frac{1}{2n}\right)-1}\right) V \\ &= \left(1-\left(1-\frac{1}{2n}\right)^n\right) V. \qquad \square\end{aligned}$$

The following lemma presents the asymptotic behavior (as the number of processors n grows) of the above guarantee.

LEMMA 5.4. *$\rho(n)$ is a monotonic decreasing function and*

$$\text{for all n} \quad \rho(n) \geq \lim_{n\to\infty} \rho(n) = 1 - \frac{1}{\sqrt{e}}.$$

Proof. $\rho(n)$ is a monotonic decreasing function; therefore its infimum is reached at the limit.

$$\lim \rho(n) = \lim 1 - \left(1-\frac{1}{2n}\right)^n = 1 - \lim\left(1+\frac{-\frac{1}{2}}{n}\right)^n = 1 - e^{-\frac{1}{2}} = 1 - \frac{1}{\sqrt{e}}.$$

(We have made use of the equality $\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$.) $\square$

We conclude that for each n the lower bound is greater than $1 - \frac{1}{\sqrt{e}} \sim 0.3935$.

Recall that the upper bound we constructed is $\frac{n}{2n-1}$ which converges to 0.5 as $n \to \infty$. In fact, both upper and lower bound sequences converge quite rapidly, as can be observed in Table 5.1 and the graph in Figure 4.

THEOREM 5.5. *For any number of processors $n \geq 1$. Given a set of tasks S there exists a nonmigrative schedule for n processors obtaining a value of at least*

$$\left(1-\frac{1}{\sqrt{e}}\right) V \sim 0.3935\ V,$$

where V denotes the value of an optimal nonmigrative schedule for n processors. Furthermore, such a schedule is produced by NO-MIGRATION.

Proof. We use Theorem 5.3 and Lemma 5.4 above. $\square$

6. Building uniprocessor schedulable subsets (the R_is). In this section we show how to build the R_i subsets, i.e., subsets of the scheduled tasks whose value is at least half of the projection of any n processor schedule.

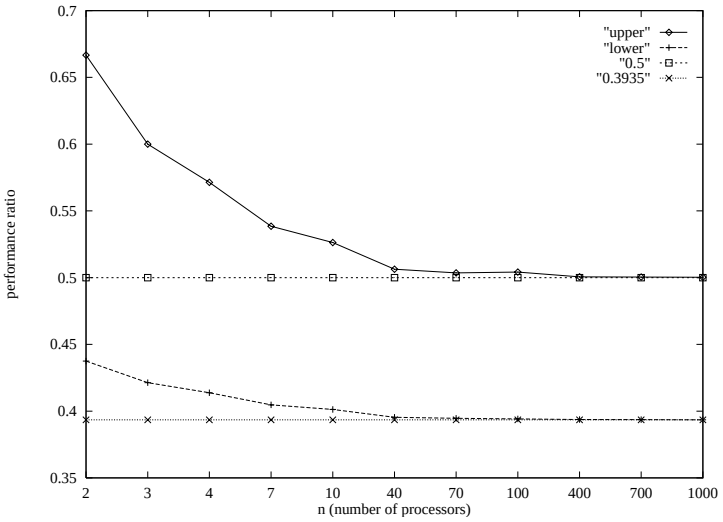


FIG. 4. *Graph of convergence speed.*

THEOREM 6.1. *Given $Sched_n(S)$, a schedule of S on n processors, there exists a $R' \subseteq S$ schedulable (on a* single *processor) with a total value of at least*

$$\frac{1}{2} Proj\left(Sched_n(S)\right).$$

The theorem will be proven in three steps as follows:

(1) First, we introduce a preemptive uniprocessor scheduling algorithm—EDF*. Note that this is essentially an on-line algorithm, but we use it here as part of an off-line scheduler.

(2) EDF-LST is a uniprocessor scheduling algorithm that uses EDF* as a subroutine. It schedules S on one processor. (Of course some tasks may not successfully complete.)

Using the schedule created above by EDF-LST we partition the projection of S into disjoint intervals. For each interval we have a (possibly overloaded) task subset scheduled in this interval.

(3) For each such subset of tasks, we construct a subsubset with the property that the sum of its tasks' values is at least half the length of the corresponding interval and it is schedulable on a uniprocessor system. The collection of all these subsubsets is schedulable on a single processor and achieves a value not less than half of the projection of $Sched_n(S)$.

6.1. The EDF* algorithm. We introduce EDF*—a uniprocessor preemptive scheduling algorithm. EDF* is used as a subroutine in EDF-LST.

The EDF* algorithm works as follows: EDF* maintains a set of **accepted** tasks and schedules these tasks according to the *earliest deadline first* (EDF) strategy [6]. Initially the set of accepted tasks is empty. Upon their release, "new" tasks may be added to the accepted set, provided that this set remains underloaded (that is, schedulable by EDF [6]).

Algorithm Outline
INPUT: A set of tasks S.

1. At the release time of some task T, if T has the minimum deadline among all currently accepted tasks we check whether the acceptance of T would not cause any of the currently accepted tasks to miss its deadline. (A similar strategy is employed by D^{over}[10].) If there is no such danger, then the new task T is accepted, the currently executing task is preempted, and T starts to execute. Else T is not accepted, and the current task continues executing.
2. Schedule the accepted tasks according to EDF.
3. If (at task completion) **accepted** becomes empty, then the ready task with an earliest deadline becomes "accepted" and scheduled for execution.

End Algorithm.

7. Example: Run through EDF*. We will illustrate the scheduling of EDF* by the set of tasks presented in Table 7.1. Initially, **accepted** is empty. At time 0, the first two tasks are released. Since T_1 has earlier deadline, **accepted** becomes $\{T_1\}$ and T_1 is scheduled. At time 1, T_3 is released, its deadline is earlier then T_1's deadline, and both tasks can be scheduled to completion. Hence T_3 preempts T_1 and it is added to **accepted**. At time 2, T_4 is released, it has a deadline which is smaller than all the tasks in **accepted** but cannot be scheduled without preventing any of the other tasks in **accepted** from completion. Hence, T_3 continues until its completion at time 3, then T_2 resumes and completes at time 5, leaving **accepted** empty. At time 5, T_2 is added to the empty **accepted** set, it is scheduled until it completes at time 7. Note that T_4 was not scheduled at all.

7.1. The EDF-LST algorithm. In cases of underloaded task sets, EDF* would have been enough for our needs. In fact, an underloaded case would mean that *all* tasks can be scheduled on *one* processor. Consequently, we can certainly expect that a task set schedulable on n processors will be unschedulable (overloaded) when scheduled on one processor. For this reason, we introduce a scheduling algorithm called EDF-LST that is an "extension" of EDF* for overloaded cases.

7.1.1. LST: Latest start time. We define *latest start time (LST)* as a critical point in a task's existence: If a task reaches its LST it means that it has no more slack time. Hence, it must be scheduled immediately and executed continuously until its completion on its deadline; otherwise it will miss its deadline. This is given formally in the following.

DEFINITION 7.1. *At any given moment t for a task T,*

$$LST\,(t,T) = deadline\,(T) - remaining\_computation\_time\,(T)\,.$$

LST-exception: *When (and if) a task that is not currently executing reaches an LST point it raises the LST-exception.*

The EDF-LST scheduling algorithm schedules according to the EDF* algorithm as long as no LST-exception is raised. At some point an LST-exception may be raised, either by previously preempted tasks, some task that EDF* did not "accept," or at the release time of tasks with no slack time. As soon as an LST-exception is raised EDF-LST might diverge from EDF* by scheduling tasks at their LST points. These tasks are called *LST tasks* while the tasks scheduled according to EDF* are called *EDF tasks*.

When there are no more LST tasks to schedule, EDF-LST returns to schedule according to EDF*. This cycle continues until there are no more tasks to schedule. The

TABLE 7.1
Set of tasks for section 7.

	r	c	d
T_1	0	3	6
T_2	0	2	8
T_3	1	2	5
T_4	2	2	4

precise conditions for LST tasks or EDF tasks will be seen in the next subsection.

This schedule creates alternating sets of EDF tasks and LST tasks covering some contiguous time interval bordered by *idle* periods. A time period is said to be *idle* if at all its points all tasks either completed, missed their deadlines, or were not released yet.

We will select from the tasks in a time interval bordered by idle periods, an underloaded (i.e., schedulable) subset of tasks whose cumulative value is at least half the length of that time interval. Combining this for all intervals, we build the desired schedule.

7.1.2. EDF-LST. The pseudo code of the algorithm is built from one main function and two exception handlers. (The subroutine that implements EDF* is not shown here.)

At each moment, the algorithm can be in one of two phases: edf_phase or lst_phase. In the first, tasks are scheduled according to EDF*. In the latter, scheduling decisions are made by the LST-exception handler as a result of raising such an exception.

The algorithm has cycles. Each cycle starts with an edf_phase followed by an optional lst_phase. While in the edf_phase the algorithm maintains a collection of "accepted" tasks (exactly in the same manner as it is done by EDF*). If a task T raises an LST-exception it will be scheduled only if its deadline is far enough[1] in the future. Otherwise, T is aborted. Scheduling T means switching to the lst_phase. Task T will schedule to completion unless another LST-exception will be raised by another task with an even "further" deadline[2] and so on.

This chain of exceptions/abortions ends when some task is able to complete its execution. This marks the end of a cycle. The algorithm now switches back to the edf_phase. (The "accepted" set is the ready task with earliest deadline.)

Let us briefly explore the intuition for choosing the above criteria for the "far enough" deadline. The main idea is identifying in each cycle a collection of "executing units" that covers an interval. The set of all the accepted tasks is the first unit. Each task scheduled by its LST-exception is a subsequent unit of its own.

Two adjacent units may interfere with each other (in the sense that scheduling one may prevent the scheduling of the other) but removing any one unit will enable its (one or) two adjacent units to complete. This mean that eliminating the collection of

[1]The meaning of "far enough" here is as follows. We consider the time interval in which the currently *accepted tasks* will execute to completion (if not interrupted). We look at all the tasks that will raise an LST-exception during that interval and whose deadlines are beyond the end of that interval. T's deadline must not be smaller than all these deadlines.

[2]The meaning of "further" here is as follows. We consider the time interval in which the currently *scheduled task* will execute to completion (if not interrupted). We look at all the tasks that will raise an LST-exception during that interval and whose deadlines are beyond the end of that interval. T's deadline must not be smaller than all these deadlines.

all odd numbered units enables the collection of all even numbered units to complete and vice-versa. These two collections cover the entire interval; hence the bigger of the two covers at least half of the interval. This is the condition we were seeking.

EDF-LST Algorithm.

INPUT: A set of tasks S.

In the following algorithm, **accepted** denotes the set of accepted tasks as defined in the definition of EDF*.

- **main body** /* main body of the algorithm */
  1. if S is empty, end of execution. /* strange, no tasks were presented */
  2. /* otherwise, there are tasks to execute */
     /* in the beginning, starting in edf_phase */
     $phase \leftarrow edf\_phase$
  3. /* main loop of the algorithm */
     loop (forever) {
     - as long as **phase** equals **edf_phase**, schedule according to EDF*.
       /* tasks scheduled in this phase are called EDF tasks */
     - otherwise, **phase** equals **lst_phase** in which case scheduling decisions are made in the LST_exception handler (see below).

     }
- **End of main body**
- **LST_Exception handler**

  /* a task T raises an LST-exception, say at time t */
  - if ($phase = edf\_phase$) then
    /* scheduling is currently done according to EDF* */
    1. Let $endof$ (**accepted**) be the time that tasks of **accepted** are expected to complete if scheduled without any interruption.
       Consider the set of all tasks that will raise an LST-exception in the time interval $[t, endof$ (**accepted**)] provided the current tasks of **accepted** would continue without interruption, and whose deadline is later than $endof$ (**accepted**).
       From this set of tasks, choose the task with the latest deadline. Denote its deadline by d_{max}.
    2. if $d(T) \geq d_{max}$ then
       /* $d(T)$ is indeed far enough */
       /* switch from edf_phase to lst_phase */
       /* end of scheduling according to EDF* start to schedule according to LST-exceptions */
       $phase \leftarrow lst\_phase$
       • $i \leftarrow 1$ /* index of LST_i */
       • label T as $\mathbf{LST_1}$
       • /* Let $raised$ ($\mathbf{LST_1}$) $= t$
       (the time that task $\mathbf{LST_1}$ raised its LST-exception) */
       • /* Let $endof$ ($\mathbf{LST_1}$) $= d(LST_1)$ (the time that LST_1 is expected to complete if scheduled without any interruption) */
       • start to execute $\mathbf{LST_1}$
       else
       /* $d(T)$ is not far enough */
       • abort T (continue in **edf_phase**, scheduling according to EDF*)
  - else

/* some T raised an **LST**-exception, while some LST_i is executing */
1. Consider the set of all tasks that will raise an LST-exception in the time interval
   $[raised(\mathbf{LST_i}), endof(\mathbf{LST_i})]$, provided that LST_i would continue without interruption, (T will be aborted), and whose deadline is later than endof($\mathbf{LST_i}$).
   From this set of tasks, choose the task with the latest deadline. Denote its deadline by d_{max}.
2. if $d(T) \geq d_{max}$ then
   /* $d(T)$ is far enough */
   /* switch from the current LST task to T */
   • $i \leftarrow i + 1$
   • label T as $\mathbf{LST_{i+1}}$
   • /* Let $raised(\mathbf{LST_{i+1}}) = t$
   (the time that task $\mathbf{LST_{i+1}}$ raised its LST-exception) */
   • /* Let $endof(\mathbf{LST_{i+1}}) = d(LST_{i+1})$ (the time that LST_{i+1} is expected to complete if scheduled without any interruption) */
   • start to execute $\mathbf{LST_{i+1}}$
   else
   /* $d(T)$ is not far enough */
   • abort T (continue with $\mathbf{LST_i}$)

– End if

End LST_Exception handler

• **Task_Completion handler**
/* The currently executing task T completes, say at time t */
if ($phase = edf\_phase$) then
/* T was scheduled by EDF* */
task completion is handled according to EDF* (schedule the next accepted task or if empty the next ready task with earliest deadline)
else
/* T is LST_j for some j */
/* switch phases, return to EDF* scheduling */
$phase \leftarrow edf\_phase$
start scheduling according to EDF*
(**accepted** is set to the next ready task with earliest deadline)

End Task Completion handler

End Algorithm.

Note that the collection of tasks scheduled by EDF-LST may be an overloaded task set, that is, some of tasks do not execute to completion. This can happen if tasks that where scheduled by EDF* are preempted and never resumed, or tasks that raise LST-exception are not selected to execute or, even if selected, may be aborted when the next LST task is scheduled.

See Figure 5 for a schematic example of an EDF-LST schedule.

In the next section we show how to select an underloaded subset of tasks with a "big enough" value, i.e., the sum of its tasks' values is no less than the projection of an optimal migrative schedule.

7.2. Selecting an underloaded subset from the EDF-LST schedule. This section completes the last necessary step to prove our lower bound. We study the schedule created by EDF-LST. First we suitably partition the tasks in this schedule.

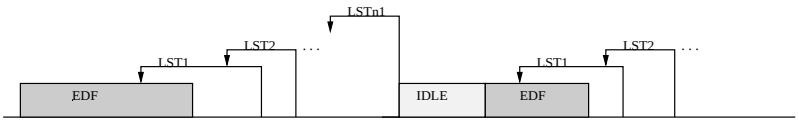


FIG. 5. *A schematic of an* EDF-LST *schedule.*

Then, we use this partition to construct an underloaded subset with a "big enough" value.

7.2.1. Partitioning EDF-LST's schedule. The schedule of EDF-LST naturally splits the tasks of S into two collections. The first is composed of tasks that *appear* in the schedule. These are tasks that were scheduled for execution for at least one time interval and eventually either completed or aborted. The second includes the rest of the tasks—those that were never scheduled for execution. These are tasks that were not scheduled by EDF* and when their LST was raised they were aborted. In the following analysis we partition the first collection into subsets; the second collection is ignored.

The result of the EDF-LST algorithm is a schedule of tasks that have the following structure:

$$EDF^1,\ LST^1, EDF^2,\ LST^2, \ldots, EDF^i,\ LST^i, \ldots,$$

where EDF^i is a set of tasks scheduled according to EDF* and LST^i is a set of tasks scheduled following an LST-exception. The individual tasks of EDF^i and LST^i are denoted by $EDF^i_1, EDF^i_2, EDF^i_3, \ldots,$ and $LST^i_1, LST^i_2, LST^i_3, \ldots,$ respectively.

For every i define the following subsets of S:

- α_i : Tasks that **last appear**[3] as one of LST^i tasks but with *odd* indexes, i.e., $LST^i_{\mathbf{1}}, LST^i_{\mathbf{3}}, LST^i_{\mathbf{5}}, \ldots$.
- β_i : Tasks that **last appear** as part of EDF^i and *did not complete* and tasks that **last appear** as one of LST^i tasks but with *even* indexes, i.e., $LST^i_{\mathbf{2}}, LST^i_{\mathbf{4}}, LST^i_{\mathbf{6}}, \ldots$.

In addition define

- γ : Tasks that **last appear** as any of EDF^is and *completed* successfully.

7.2.2. Building an underloaded task set.

LEMMA 7.2. *Each of the sets $\gamma, \alpha_i, \beta_i$ (for all i) is an underloaded task set (i.e., can be scheduled successfully on one processor).*

Proof. γ is obviously underloaded since all its tasks were successfully completed by EDF-LST.

We will show that for all i, α_i and β_i are underloaded. If α_i is empty, so is β_i (because empty α_i implies that LST^i is empty; if this is the case then all tasks of EDF^i were scheduled to completion and hence β_i is empty) and we are done. Otherwise, suppose we eliminate the tasks of α_i from the schedule created by EDF-LST. We will show how all the tasks of β_i can now successfully complete by using only the time intervals thus vacated.

• *Eliminating LST^i_1 will enable all the tasks of EDF^i to complete.*

This can be seen by the following argument:

[3] "*Last appear*" means the last time it appeared, i.e., the last time it was executed. Note that a task may be preempted and rescheduled many times. However, since each task may "last appear" at most once we obtain a partitioning into **disjoint** task.

(1) If there is no LST_2^i, then LST_1^i completes running by its deadline, according to the EDF-LST schedule. EDF-LST schedules LST_1^i only if its deadline occurs after $endof(EDF^i)$.
(2) If LST_2^i exists, then the LST-exception of LST_2^i occurred after $endof(EDF^i)$. (Otherwise, LST_1^i would not have been scheduled.)

Conclude that in both cases the time freed by LST_1^i is enough for all EDF^i tasks.

• *Eliminating LST_3^i will enable LST_2^i to complete.*

The reasoning is similar. If there is no LST_4^i, then LST_3^i completed by EDF-LST. EDF-LST schedules LST_3^i only if its deadline is later than $d(LST_2^i)$.

If there exists LST_4^i, then the LST-exception of LST_4^i occurred after $d(LST_2^i)$. (Otherwise, LST_3^i would not have been scheduled.)

Conclude that in both cases the time freed by LST_3^i is enough for LST_2^i.

• *In general, eliminating α_i enables all the tasks of β_i to complete.*

This can be easily verified by induction in a similar manner to the above two cases.

Conclusion: β_i is underloaded.

A similar argument, though somewhat simpler since the tasks of EDF^i do not matter here, shows that eliminating β_i enables all the tasks of α_i to complete.

Conclusion: α_i is underloaded. □

7.3. Notation.

- Let σ be any schedule (migrative or nonmigrative) of S on n processors. Denote the projection of σ by $PROJ$.
- Denote the union of the nonidle periods of EDF-LST (for S) by $BUSY$.

For all i define

- $\max_i$: the set with higher total value among α_i and β_i (if equal value, choose arbitrarily);
- $\min_i$: the set with lower total value among α_i and β_i.

For ease of notation we will henceforth denote the *values* of $\gamma, \alpha_i, \beta_i$, and $\max_i$ also by $\gamma, \alpha_i, \beta_i$, and $\max_i$, respectively. Similarly we will use $PROJ$ and $BUSY$ to also denote their total lengths.

LEMMA 7.3. $\gamma \bigcup \{\cup_i \max_i\}$ *is an underloaded task set.*

Proof. From the proof of Lemma 7.2 above, we see that expanding the tasks of $\max_i$ to the time vacated by eliminating $\min_i$ (and only this time) suffices for the completion of all the tasks of $\max_i$. Hence, eliminating $\cup_i \min_i$ enables $\cup_i \max_i$ to complete without disturbing any of the γ tasks. □

LEMMA 7.4. $\gamma + 2\sum_i \max_i \geq BUSY$.

Proof. Only tasks from γ, α_i, and β_i appear in the schedule of EDF-LST. Hence,

$$\gamma \bigcup \cup_i \{\alpha_i \cup \beta_i\} \geq BUSY.$$

Since $2max_i \geq (\alpha_i + \beta_i)$ we get the desired result. □

LEMMA 7.5. $BUSY + \gamma \geq PROJ$.

(Recall that BUSY refers to a uniprocessor schedule created by EDF-LST *while PROJ refers to an n processors schedule.)*

Proof. It is obvious that $BUSY + (PROJ \setminus BUSY) \geq PROJ$. Consider a point t in $PROJ \setminus BUSY$ (i.e., an idle point according to EDF-LST). Let T be a task that executes at time $t \leq d(T)$ (according to σ).

If T was aborted by EDF-LST, then EDF-LST has no idle time between $r(T)$ and $d(T)$; otherwise T would occupy the idle periods, making them busy.

Hence, T completed by EDF-LST. Moreover, it completed before t (otherwise t won't be an idle point) meaning it completed as an EDF task,[4] that is, $T \in \gamma$.

We conclude that all the tasks T_i which executed (even partially) at idle periods of EDF-LST are tasks of γ, i.e., $PROJ - BUSY \leq \gamma$, which proves the lemma. ∎

COROLLARY 7.6. $\gamma + \sum_i \max_i \geq \frac{PROJ}{2}$.

Proof. Lemma 7.4 shows that

$$\gamma + 2\sum_i \max_i + \gamma \geq BUSY + \gamma.$$

Using Lemma 7.5 we get

$$2\left(\gamma + \sum_i \max_i\right) \geq PROJ. \qquad \blacksquare$$

This concludes the proof of Theorem 6.1 and assures us that there is indeed a nonmigrative schedule on n processors whose value is no less than $1 - (1 - \frac{1}{2n})^n$ of the optimal migrative schedule.

7.4. A run-through of the No-Migration algorithm. Let us run through No-Migration executed on the set of tasks presented in the performance ratio example (section 3). This set of tasks has three identical tasks, T_1, T_2, and T_3. They all have the same parameters: start time 0, deadline 1, and computation time $\frac{2}{3}$.

In the first iteration we utilize EDF-LST which starts by utilizing EDF*. Since all tasks have the same start time, EDF* chooses any one to start with, say T_1. Therefore, $EDF_1^1 = T_1$. At time $t = \frac{1}{3}$, there is an LST-exception for both T_2 and T_3. Suppose the exception of T_2 is treated first. Since its deadline is greater than the time that EDF_1^1 is expected to complete, we preempt T_1 and schedule T_2. Hence, $LST_1^1 = T_2$. Immediately, EDF-LST treats the exception generated by T_3 leading to its abortion. At time 2/3, T_1 raises an LST-exception which leads to its abortion.

In this example, α_1 is $\{T_2\}$, β_1 is $\{T_1\}$, and γ is empty (so far). Note that T_3 is missing from all these sets. In this case α and β are of equal size so R_1 can be either of them; suppose the latter is chosen.

Now to the second iteration in No-Migration. T_1 is removed from the initial task collection and we proceed as above with the remaining tasks (T_2 and T_3). Suppose T_2 is scheduled initially. At time 1/3 it will be preempted by an exception of T_3 which will complete. Hence α_2 is $\{T_2\}$ and β_2 is $\{T_3\}$ (γ is empty). Since these are equal R_2 could be any of them. Assume the latter is chosen. This concludes the second and last iteration of No-Migration.

Thus, the schedule generated by No-Migration, T_1, gets scheduled on the first processor and T_3 gets scheduled on the second processor. T_2 is not scheduled at all.

In this example the No-Migration algorithm, in fact, chose an optimal nonmigrative schedule.

8. Conclusion and open problems. We gave an upper and a lower bound on the performance ratio between nonmigrative and migrative clairvoyant schedulers in real-time systems. While the bounds are close there is still a gap between them. We believe that the upper bound is the real bound.

An important open problem is finding a competitive on-line scheduler for n processors in a real-time system with the uniform density value measure. The results

[4] Recall that an LST task completes exactly on its deadline.

in this paper guarantee that whether such a competitive on-line scheduler is migrative or nonmigrative, they nevertheless imply the existence of *both* a migrative and a nonmigrative competitive scheduler.

REFERENCES

[1] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber, *Bandwidth allocation with preemption*, in Proceedings of the 27th ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 616–625.

[2] A. Bar-Noy, Y. Mansour, and B. Schieber, *private communication,* IBM T. J. Watson Research Center, Yorktown Heights, NY, 1992.

[3] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang, *On the competitiveness of on-line task real-time task scheduling*, J. Real-Time Systems, 4 (1992), pp. 124–144; conference version appeared in Proceedings of the 12th Real-Time Systems Symposium, San Antonio, TX, 1991, pp. 106–115.

[4] A. Burns, *Scheduling hard real-time systems: A review*, Software Engrg. J., 6 (1991), pp. 116–128.

[5] S.-C. Cheng and J. A. Stankovic, *Scheduling algorithms for hard real-time systems: A brief survey*, in Hard Real-Time Systems, J. Stankovic and K. Ramamritham, eds., IEEE Computer Society Press, Los Alamitos, CA, 1988, pp. 150–173.

[6] M. L. Dertouzos, *Control robotics: The procedural control of physical processes*, in Proceedings of the IFIF Congress, Stockholm, Sweden, 1974, North-Holland, Amsterdam, 1974, pp. 807–813.

[7] M. L. Dertouzos and A. K.-L. Mok, *Multiprocessor on-line scheduling of hard-real-time tasks*, IEEE Trans. Software Engrg., 15 (1989), pp. 1497–1506.

[8] J. E. G. Coffman and M. R. Garey, *Preemptive vs. nonpreemptive two-processor scheduling*, J. Assoc. Comput. Mach., 40 (1993), pp. 991–1018.

[9] G. Koren and D. Shasha, *MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling*, Theoret. Comput. Sci., 128 (1994), pp. 75–97.

[10] G. Koren and D. Shasha, *D-over: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems*, SIAM J. Comput., 24 (1995), pp. 318–339.

[11] C. D. Locke, *Best-Effort Decision Making for Real-Time Scheduling*, Ph.D. thesis, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1986.

[12] J. Ullman, *Complexity of sequencing problems,* in Computer and Job-Shop Scheduling Theory, E. G. Coffman Jr., ed., Wiley, New York, 1976.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 528–560

APPROXIMATING MINIMUM-SIZE k-CONNECTED SPANNING SUBGRAPHS VIA MATCHING*

JOSEPH CHERIYAN† AND RAMAKRISHNA THURIMELLA‡

Abstract. An efficient heuristic is presented for the problem of finding a minimum-size k-connected spanning subgraph of an (undirected or directed) *simple* graph $G = (V, E)$. There are four versions of the problem, and the approximation guarantees are as follows:

- minimum-size k-*node* connected spanning subgraph of an *undirected* graph $\quad 1 + [1/k]$,
- minimum-size k-*node* connected spanning subgraph of a *directed* graph $\quad 1 + [1/k]$,
- minimum-size k-*edge* connected spanning subgraph of an *undirected* graph $\quad 1 + [2/(k+1)]$, and
- minimum-size k-*edge* connected spanning subgraph of a *directed* graph $\quad 1 + [4/\sqrt{k}]$.

The heuristic is based on a subroutine for the degree-constrained subgraph (b-matching) problem. It is simple and deterministic and runs in time $O(k|E|^2)$.

The following result on simple undirected graphs is used in the analysis: The number of edges required for augmenting a graph of minimum degree k to be k-edge connected is at most $k\,|V|/(k+1)$.

For undirected graphs and $k = 2$, a (deterministic) parallel **NC** version of the heuristic finds a 2-node connected (or 2-edge connected) spanning subgraph whose size is within a factor of $(1.5 + \epsilon)$ of minimum, where $\epsilon > 0$ is a constant.

Key words. graphs, directed graphs, graph connectivity, edge connectivity, node connectivity, matchings, degree constrained subgraphs, NP-complete problems, approximation algorithms, network design

AMS subject classifications. 05C40, 05C70, 05C85, 68R10, 68W25, 90B18, 90C27, 90C35, 90C59

PII. S009753979833920X

1. Introduction.

Given an undirected or directed *simple* graph $G = (V, E)$, an efficient approximation algorithm[1] is presented for the problem of finding a k-connected spanning subgraph $G' = (V, E')$ that has the minimum number of edges ($k \geq 1$ is an integer). Let n and m denote $|V|$ and $|E|$, respectively. There are four versions of the problem, depending on whether G is a graph (i.e., an undirected graph) or a digraph (i.e., a directed graph) and on whether the spanning subgraph G' is required to be k-node connected or k-edge connected. All four versions of the problem are NP-hard: the two problems on graphs are NP-hard for $k \geq 2$, and the two problems on digraphs are NP-hard for $k \geq 1$, [GJ 79].

Previous work. Results of Mader [Ma 71, Ma 72] (also see [Bo 78]) imply that every *minimal*[2] k-edge connected graph has at most kn edges, and every minimal

*Received by the editors May 29, 1998; accepted for publication (in revised form) December 23, 1999; published electronically June 3, 2000. An extended abstract of this paper appeared in *Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science*, 1996, pp. 292–301.
http://www.siam.org/journals/sicomp/30-2/33920.html

†Department of Combinatorics and Optimization, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (jcheriyan@math.uwaterloo.ca, http://www.math.uwaterloo.ca/~jcheriyan). The research of this author was supported in part by NSERC grant OGP0138432 (NSERC code OGPIN 007).

‡Department of Mathematics and Computer Science, University of Denver, 2360 S. Gaylord St., Denver, CO 80208 (ramki@cs.du.edu, http://www.cs.du.edu/~ramki). The research of this author was supported in part by NSF Research Initiation Award grant CCR-9210604.

[1] An α-*approximation algorithm* for a combinatorial optimization problem runs in polynomial time and delivers a solution whose value is always within the factor α of the optimum value. The quantity α is called the *approximation guarantee* of the algorithm.

[2] A graph H is called *minimal* with respect to a property $\mathcal{P}$ if H possesses $\mathcal{P}$, but for every edge e in H, $H \backslash e$ does *not* possess $\mathcal{P}$.

k-node connected graph has at most kn edges. Clearly, a k-connected (i.e., k-node connected or k-edge connected) graph has at least $kn/2$ edges, since each node has degree $\geq k$. Similarly, every k-connected digraph has at least kn arcs (directed edges) since each node has outdegree $\geq k$, and results of Edmonds [Ed 72] and Mader [Ma 85] imply that every minimal k-connected digraph has at most $2kn$ arcs. These facts immediately imply a 2-approximation algorithm for all four versions of the problem, since there is an easy polynomial-time algorithm to find a minimal k-edge connected (or k-node connected) spanning subgraph of a given graph or digraph. For graphs, recent algorithmic work gives another easy and efficient method for finding a k-connected spanning subgraph whose size (i.e., number of edges) is at most kn. A k-edge connected spanning subgraph (V, E') is obtained by taking $E' = F_1 \cup F_2 \cup \cdots \cup F_k$, where F_i $(1 \leq i \leq k)$ is the edge set of a maximal (but otherwise arbitrary) spanning forest of $(V, E \backslash (F_1 \cup \cdots \cup F_{i-1}))$ (see [Th 89, NI 92]), and a k-node connected spanning subgraph (V, E') is obtained similarly, but now each F_i is a maximal scan-first-search spanning forest; see [NI 92, FIN 93, CKT 93].

In the approximate solution of NP-hard combinatorial optimization problems, it often turns out that finding a solution within a factor of 2 of optimum is almost trivial, but achieving (asymptotically) better approximation guarantees needs a deeper understanding of the problem. For example, consider the metric TSP, i.e., the traveling salesman problem, with edge weights satisfying the triangle inequality. Finding a solution whose value is within a factor of 2 of optimum is trivial. The Christofides heuristic [Ch 76] broke the 2-approximation barrier by employing a powerful idea: matching.

Given a graph, consider the problem of finding a minimum-size 2-edge connected spanning subgraph (2-ECSS), or a minimum-size 2-node connected spanning subgraph (2-NCSS). Several recent papers have focused on these two problems. Khuller and Vishkin [KV 94] achieved the first significant advance by obtaining approximation guarantees of 1.5 and 1.66 for the minimum-size 2-ECSS problem and the minimum-size 2-NCSS problem. Garg, Santosh, and Singla [GSS 93] improved the approximation guarantee of the latter problem to 1.5. These algorithms are based on depth-first search (DFS), and they do not imply efficient parallel algorithms for the PRAM model. Subsequently, Chong and Lam [CL 95, CL 96] gave (deterministic) **NC** algorithms on the PRAM model with approximation guarantees of $(1.5 + \epsilon)$ and $(1.66 + \epsilon)$ for the minimum-size 2-ECSS problem and the minimum-size 2-NCSS problem.

For graphs and the general minimum-size k-ECSS problem, first Karger [Ka 94] used randomized rounding to improve the approximation guarantee (for k large with respect to $\log n$) to $1 + \sqrt{[O(\log n)/k]}$; Karger's algorithm is not deterministic but Las Vegas. Then Khuller and Raghavachari [KR 96] improved the approximation guarantee (for all k) from 2 to (roughly) 1.85. They left open the problem of improving on the approximation guarantee of 2 for the minimum-size k-NCSS problem.

For digraphs and the problem of finding a minimum-size 1-connected (i.e., strongly connected) spanning subgraph, Khuller, Raghavachari, and Young [KRY 96, KRY 95] gave a 1.61-approximation algorithm. For digraphs and $k \geq 2$, there appears to have been no previous work on achieving approximation guarantees better than 2.

An illustrative example. Here is an example illustrating the difficulty in improving on the 2-approximation guarantee for the minimum-size k-connected spanning subgraph problem. Let the given graph G have n nodes, where n is even. Suppose that the edge set of G, $E(G)$, is the union of the edge set of the complete bipartite graph $K_{k,(n-k)}$ and the edge set E_{opt} of an n-node, k-regular, k-edge connected (or k-

TABLE 1
A summary of previous and new approximation guarantees for minimum-size k-edge connected spanning subgraphs (k-ECSS), and minimum-size k-node connected spanning subgraphs (k-NCSS).

	Previous results	
	Undirected Graphs	Digraphs
k-ECSS	$2-[1/k]$ for $k \geq 2$ [K 96] 1.85 for $k \geq 2$ [KR 96] $1+\sqrt{O(\log n)/k}$ [Ka 94]	1.61 for $k = 1$ [KRY 96] 2 for $k \geq 2$
k-NCSS	1.5 for $k = 2$ [GSS 93] 2 for $k \geq 3$	1.61 for $k = 1$ [KRY 96] 2 for $k \geq 2$

	Results in this paper	
	Undirected Graphs	Digraphs
k-ECSS	$1+[2/(k+1)]$ **improves for** $k \geq 3$	$1+[4/\sqrt{k}]$ **improves for** $k \geq 17$
k-NCSS	$1+[1/k]$ **improves for** $k \geq 3$	$1+[1/k]$ **improves for** $k \geq 2$

node connected) graph. For example, for $k = 2$, $E(G)$ is the union of $E(K_{2,(n-2)})$ and the edge set of a Hamiltonian cycle. A naive heuristic may return $E(K_{k,(n-k)})$ which has size $k(n-k)$, roughly two times $|E_{opt}|$. A heuristic that significantly improves on the 2-approximation guarantee must somehow return many edges of E_{opt}.

Results in this paper.

Heuristics and approximation guarantees. This paper first presents a simple heuristic for finding an approximately minimum-size k-NCSS of a given graph or digraph. An approximation guarantee of $1+[1/k]$ is proved. A variant of the heuristic finds a small-size k-ECSS of a given graph or digraph. For graphs and the minimum-size k-ECSS problem, the approximation guarantee is $1+[2/(k+1)]$. For digraphs and the minimum-size k-ECSS problem, the approximation guarantee is $1+[4/\sqrt{k}]$. Let $G = (V, E)$ be the given graph. The heuristic has two steps. The first step finds a minimum-size subgraph (V, M) of minimum-degree k (or $k-1$) via a subroutine for the degree-constrained subgraph (b-matching) problem. The second step adds an (inclusionwise) minimal edge set $F \subseteq E \backslash M$ such that the resulting graph $(V, M \cup F)$ is either k-node connected or k-edge connected, as required. Heuristics of this type have been considered by other researchers, but we were not aware of this when the preliminary version of this paper (*Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1996, pp. 292–301) appeared. Subsequently, Khuller (private communication, October 1996) and Watanabe (private communication, October 1996) informed us that they had examined or implemented heuristics of this type. One of the contributions of this paper is to refine the general heuristic to the four minimum-size k-CSS problems discussed above, and to give nearly tight analyses of the four approximation guarantees. The running time of the heuristic is $O(k|E|^2)$, and for

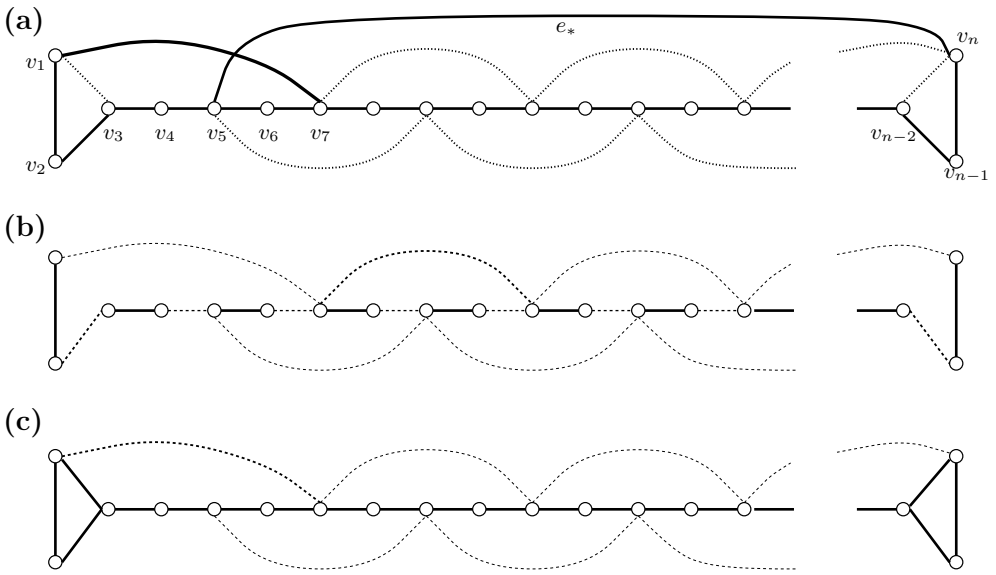


FIG. 1. *Illustrating the 2-NCSS heuristic on a 2-node connected graph $G = (V, E)$; $n = |V|$ is even, and $k = 2$. Adapted from Garg, Santosh, and Singla* [GSS 93, Figure 7]. (a) *A minimum-size 2-node connected spanning subgraph has $n + 1$ edges and is indicated by thick lines (the path $v_1, v_2, \ldots, v_n$ and edges v_1v_7 and $e_* = v_5v_n$).* (b) *The first step of the heuristic in section 3.1 finds a minimum-size $M \subseteq E$ such that every node is incident to $\geq (k-1) = 1$ edges of M. The thick lines indicate M; it is a perfect matching. The second step of the heuristic finds an (inclusionwise) minimal edge set $F \subseteq E$ such that $(V, M \cup F)$ is 2-node connected. F is indicated by dashed lines—the "key edge" e_* is not chosen in F. $|M \cup F| = 1.5n - 5$.* (c) *Another variant of the heuristic first finds a minimum-size $M \subseteq E$ such that every node is incident to $\geq k = 2$ edges of M. The thick lines indicate M (M is the path $v_1, v_2, \ldots, v_n$ and edges v_1v_3, $v_{n-2}v_n$). The second step of the heuristic finds the edge set $F \subseteq E$ indicated by dashed lines—the "key edge" e_* is not chosen in F. $(V, M \cup F)$ is 2-node connected, and for every edge vw in F, $(V, M \cup F) \backslash vw$ is* not *2-node connected. $|M \cup F| = 1.5n - 3$.*

graphs the running time improves to $O(k^3|V|^2 + |E|^{1.5}(\log |V|)^2)$. The analyses on graphs/digraphs of the minimum-size k-NCSS heuristic are based on theorems of Mader [Ma 72, Ma 85]. In the context of augmenting the node connectivity of graphs and digraphs, the first application of Mader's theorems is due to Jordán [Jo 95, Jo 93]. Two key lemmas in our analyses, namely, Lemmas 3.3 and 3.18, are inspired by similar results of Jordán, namely, [Jo 95, Lemma 3.3] and the following paragraph in [Jo 95] and Lemma 2.6 and Corollary 2.7 in [Jo 93]. In the context of approximation algorithms for minimum-size k-connected spanning subgraph problems, Chong and Lam [CL 95] appear to be the first to use matching.

For graphs, the heuristic finds a 2-node connected or 2-edge connected spanning subgraph whose size is within a factor of 1.5 of the minimum size. A parallel (deterministic) version gives a $(1.5+\epsilon)$-approximation **NC** algorithm. Similarly, a sequential *linear-time* version gives an approximation guarantee of $(1.5 + \epsilon)$.

Table 1 summarizes the approximation guarantees obtained in this paper for the four versions of the problem and compares these with the previous best approximation guarantees. Figure 1 illustrates the working of the heuristic on an example.

Here is a summary of developments since September 1996. Fernandes [Fe 97, Theorem 5.1] showed that the minimum-size 2-ECSS problem on graphs is MAX SNP-hard; also see Czumaj and Lingas [CL 99, Theorem 5.3]. Independently of this paper,

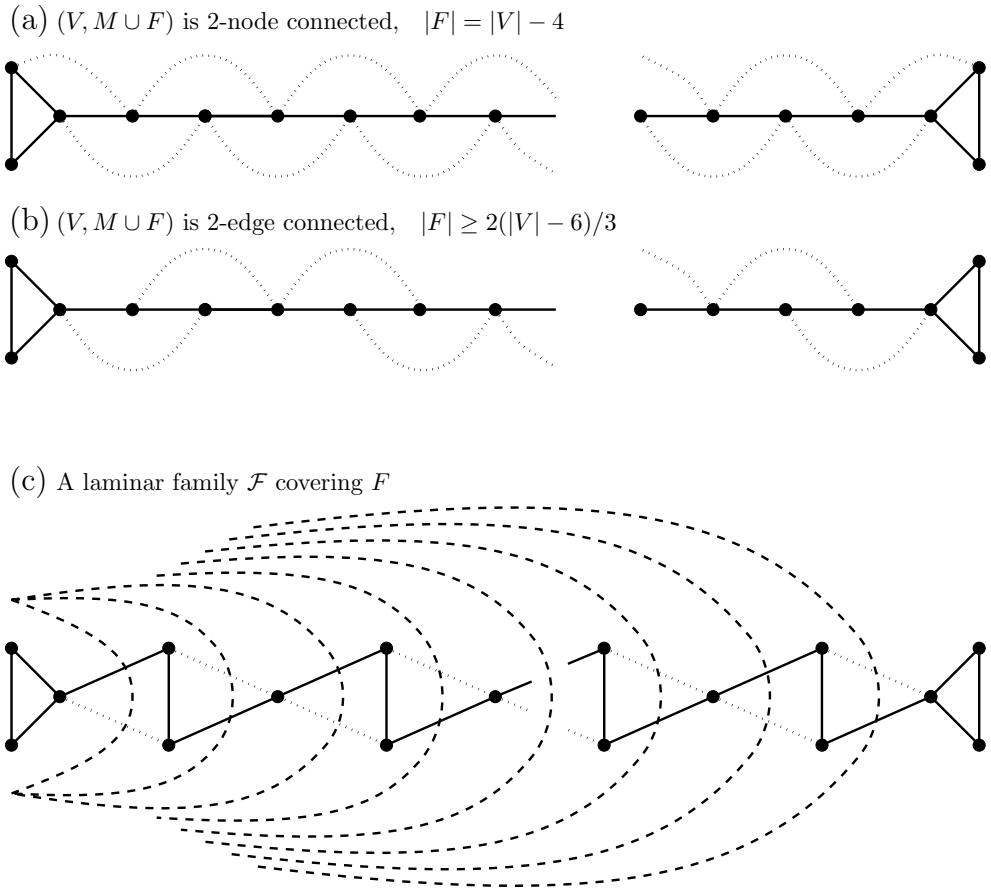


FIG. 2. *An illustration of Lemma* 3.3 *(a corollary of Mader's theorem, Theorem* 3.2*) and of Theorem* 4.3. *An* n*-node graph of minimum degree* $k = 2$*,* (V, M)*, is indicated by solid lines.* (a) *The dotted lines indicate an (inclusionwise) minimal edge set* F *such that* $(V, M \cup F)$ *is* 2*-node connected.* F *has size* $n - 4$*, for* $n \geq 4$*. By Lemma* 3.3*, the maximum size of* F *over all possible* M *is* $\leq n - 1$. (b) *The dotted lines indicate an (inclusionwise) minimal edge set* F *such that* $(V, M \cup F)$ *is* 2*-edge connected.* F *has size* $\geq 2(n - 6)/3$*, for* $n \geq 6$*. By Theorem* 4.3*, the maximum size of* F *over all possible* M *is* $\leq 2(n - 1)/3$. (c) *The dashed lines indicate a laminar family of tight node sets* $\mathcal{F}$ *covering the* F*-edges of the* 2*-edge connected graph in* (b)*. The proof of Theorem* 4.3 *is based on examining* M*,* F*, and* $\mathcal{F}$*.*

and using different methods, Chong and Lam [CL 96b] have also obtained a parallel (deterministic) $(1.5 + \epsilon)$-approximation **NC** algorithm for the minimum-size 2-NCSS problem on graphs. The approximation guarantee for the minimum-size 2-ECSS problem has been improved from 1.5 to 17/12 by [CSS 98] and then to 4/3 by [VV 99].

Contributions to approximation algorithms for "uniform" network design. As discussed above, the subarea of network design with uniform edge costs and uniform connectivity requirements has attracted a fair amount of recent interest in theoretical computer science, e.g., the references cite 10 papers from this subarea. This paper takes up four central questions from this subarea and settles them in the sense that reasonably good approximation guarantees are derived based on a simple heuristic. To achieve the approximation guarantees, the paper has to rely on some deep areas of graph theory and combinatorial optimization.

Combinatorial contributions. The paper has two combinatorial results that may be of independent interest. The first is Theorem 3.5 which gives a new lower bound on the size of a k-edge connected spanning subgraph. The proof relies on the Gallai–Edmonds decomposition theorem of matching theory. Theorem 3.5 is related to a result of Gupta: a bipartite graph of minimum degree k has k edge-disjoint edge covers. The second combinatorial result of independent interest is Theorem 4.3. This theorem gives an asymptotically tight upper bound of $k|V|/(k+1)$ on the size of an (inclusionwise) minimal edge set F such that $(V, M \cup F)$ is a k-edge connected (simple) graph, where (V, M) is a graph of minimum degree $\geq k$. The (long) proof makes use of a laminar family of tight node sets that covers F. Theorem 4.3 is related to a theorem of Mader on "critical cycles" in a k-node connected graph; see Theorem 3.2. Apparently, Mader's theorem has *no* analogue for k-edge connected graphs; for $k = 2$, this can be seen from the example in Figure 5 in section 4.1; the example generalizes to all $k \geq 2$. However, there is one implication of Mader's theorem that is an analogue of Theorem 4.3: If (V, M) is as above, and F is an (inclusionwise) minimal edge set such that $(V, M \cup F)$ is a k-node connected graph, then $|F| \leq |V| - 1$ (see Lemma 3.3). Both the bounds $(k|V|/(k+1)$ in Theorem 4.3, and $|V| - 1$ in Lemma 3.3) are tight up to an additive term of $(k+1)$, for all $k \geq 2$. Figure 2 has relevant examples for $k = 2$, and these examples generalize for all $k \geq 2$. Although Theorem 4.3 and Lemma 3.3 are analogous, the two results seem to be focusing on two essentially different combinatorial structures, and neither result implies the other one.

Organization of the paper. The rest of the paper is organized as follows. Section 2 has definitions and notation. Section 3 presents the heuristic for approximating a minimum-size k-node connected spanning subgraph of a graph or a digraph and separately analyzes the approximation guarantees on graphs and digraphs. Section 4 describes and analyzes the heuristic for approximating a minimum-size k-edge connected spanning subgraph of a graph or a digraph. Section 5 has conclusions, including a discussion of the relationship to graph theory.

2. Definitions and notation. For a subset S' of a set S, $S \backslash S'$ denotes the set $\{x \in S : x \notin S'\}$.

This paper considers finite *simple* graphs and digraphs, i.e., the graphs/digraphs have no loops nor multiedges. (But, Propositions 3.9 and 3.10 do allow multiedges.) Let $G = (V, E)$ be a graph or a digraph. $V(G)$ and $E(G)$ stand for the node set and the edge set of G. By the *size* of G we mean $|E(G)|$. First, suppose that G is a graph. An edge incident to nodes v and w is denoted by vw. For a subset M of E and a node v, we use $\deg_M(v)$ to denote the number of edges of M incident to v; $\deg(v)$ denotes $\deg_E(v)$.

A node is said to be *covered* by an edge set M if the node is incident to at least one edge of M; otherwise, the node is *uncovered* by M. An *edge cover* is a set of edges that covers all the nodes. A *matching* of a graph $G = (V, E)$ is an edge set $M \subseteq E$ such that $\deg_M(v) \leq 1$, $\forall v \in V$; furthermore, if every node $v \in V$ has $\deg_M(v) = 1$, then M is called a *perfect matching*. A graph G is called *factor-critical* if for every node $v \in V$, there is a perfect matching in $G \backslash v$; see [LP 86].

An $x \leftrightarrow y$ path refers to a path whose end nodes are x and y. We call two paths *openly disjoint* if every node common to both paths is an end node of both paths. Hence, two (distinct) openly disjoint paths have no edges in common and possibly have no nodes in common. A set of $k \geq 2$ paths is called openly disjoint if the paths are pairwise openly disjoint. For a node set $S \subseteq V(G)$, $\delta_G(S)$ denotes the set of all edges in $E(G)$ that have one end node in S and the other end node in $V(G) \backslash S$ (when

there is no danger of confusion, the notation is abbreviated to $\delta(S)$); $\delta(S)$ is called a *cut*, and by a *k-cut* we mean a cut that has exactly k edges.

A graph $G = (V, E)$ is said to be k-edge connected if $|V| \geq k + 1$ and the deletion of any set of $< k$ edges leaves a connected graph. A graph $G = (V, E)$ is said to be *k-node connected* if $|V| \geq k + 1$ and the deletion of any set of $< k$ nodes leaves a connected graph.

Let $G = (V, E)$ be a digraph. An arc (directed edge) with start node v and end node w is denoted (v, w). For $M \subseteq E$ and a node v, $\deg_{M,out}(v)$ ($\deg_{M,in}(v)$) denotes the number of arcs of M with start node v (end node v). For a node set $S \subseteq V$, $\delta_{out}(S)$ ($\delta_{in}(S)$) denotes the set of arcs with start nodes in S and end nodes in $V\backslash S$ (end nodes in S and start nodes in $V\backslash S$). The digraph is called *strongly connected* (1-connected) if for every (ordered) pair of nodes v,w there exists a directed path from v to w. The digraph is called *k-edge connected* if $|V| \geq k + 1$ and the deletion of any set of $< k$ arcs leaves a strongly-connected digraph. The digraph is called *k-node connected* if $|V| \geq k + 1$ and the deletion of any set of $< k$ nodes leaves a strongly-connected digraph.

An edge vw (arc (v, w)) of a k-node connected graph G (digraph G) is called *critical* with respect to k-node connectivity if $G\backslash vw$ ($G\backslash(v, w)$) is *not* k-node connected. Similarly, we have the notion of critical edges (arcs) with respect to k-edge connectivity.

Let $G = (V, E)$ be a graph, and let $b : V \to Z_+$ assign a nonnegative integer b_v to each node $v \in V$. The perfect b-matching (or perfect degree-constrained subgraph) problem is to find an edge set $M \subseteq E$ such that each node v has $\deg_M(v) = b_v$. The maximum b-matching (or maximum degree-constrained subgraph) problem is to find a maximum-cardinality $M \subseteq E$ such that each node v has $\deg_M(v) \leq b_v$. The b-matching problem can be solved in time $O(|E|^{1.5}(\log |V|)^{1.5}\sqrt{\alpha(|E|, |E|)})$; see [GaTa 91, section 11]. (For our version of the problem, note that each edge has unit cost and unit capacity, and each node v may be assumed to have $0 \leq b_v \leq \deg(v)$.) Also see [Ge 95, section 7.3] and [Ga 85].

3. A $(1 + \frac{1}{k})$-approximation algorithm for minimum-size k-node connected spanning subgraphs. This section presents the heuristic for finding an approximately minimum-size k-node connected spanning subgraph (abbreviated k-NCSS) and proves an approximation guarantee of $1+[1/k]$. First, we focus on graphs, and then we turn to digraphs. The analysis of the heuristic for graphs hinges on a deep theorem of Mader [Ma 72, Theorem 1]. Given a graph $G = (V, E)$, a straightforward application of Mader's theorem shows that the number of edges in the k-NCSS returned by the heuristic is at most

$$(|V| - 1) + \min\{|M| : M \subseteq E \text{ and } \deg_M(v) \geq (k - 1) \quad \forall v \in V\};$$

see Lemma 3.3 below. An approximation guarantee of $1+[2/k]$ on the heuristic follows, since the number of edges in a k-node connected graph is at least $k|V|/2$, by the "degree lower bound"; see Proposition 3.4. Often, the key to proving improved approximation guarantees for (minimizing) heuristics is a nontrivial lower bound on the value of every solution. We improve the approximation guarantee from $1+[2/k]$ to $1+[1/k]$ by exploiting a new lower bound on the size of a k-edge connected spanning subgraph; see Theorem 3.5 which states the following.

> The number of edges in a k-edge connected spanning subgraph of a graph $G = (V, E)$ is at least $\lfloor |V|/2 \rfloor + \min\{|M| : M \subseteq E \text{ and } \deg_M(v) \geq (k - 1) \quad \forall v \in V\}$.

The analysis of the heuristic for digraphs is similar and hinges on another theorem of Mader [Ma 85, Theorem 1], which may be regarded as the generalization of [Ma 72, Theorem 1] to digraphs. An approximation guarantee of $1 + [1/k]$ is proved on the digraph heuristic by employing a simpler version of Theorem 3.5, namely Proposition 3.8, to give a lower bound on the number of edges in a solution.

Assume that the given graph or digraph $G = (V, E)$ is k-node connected; otherwise, the heuristic will detect this and report failure.

3.1. Undirected graphs. Let $E^* \subseteq E$ denote a minimum-cardinality edge set such that the spanning subgraph (V, E^*) is k-*edge* connected. Note that every k-node connected spanning subgraph (V, E') (such as the optimal solution) is necessarily k-edge connected and so has $|E'| \geq |E^*|$.

The heuristic has two steps. The first finds a *minimum-size* spanning subgraph (V, M), $M \subseteq E$, whose minimum degree is $(k-1)$, i.e., each node is incident to $\geq (k-1)$ edges of M. Clearly, $|M| \leq |E^*|$, because (V, E^*) has minimum degree k, i.e., every node is incident to $\geq k$ edges of E^*. To find M efficiently, we use the algorithm for the maximum degree-constrained subgraph (b-matching) problem. Our problem is

$$\min\{|M| : \deg_M(v) \geq (k-1) \quad \forall v \in V, \text{ and } M \subseteq E\}.$$

To see that this is a b-matching problem, consider the equivalent problem of finding the complement $\overline{M}$ of M with respect to E, where $\overline{M} = E \backslash M$:

$$\max\{|\overline{M}| : \deg_{\overline{M}}(v) \leq \deg(v) + 1 - k \quad \forall v \in V, \text{ and } \overline{M} \subseteq E\}.$$

The b-matching problem can be solved in time $O(|E|^{1.5}(\log |V|)^2)$ (see [GaTa 91]), hence this running time suffices to find M.

The second step is equally simple. We find an (inclusionwise) minimal edge set $F \subseteq E \backslash M$ such that $M \cup F$ gives a k-node connected spanning subgraph, i.e., $(V, M \cup F)$ is k-node connected and for each edge $vw \in F$, $(V, M \cup F) \backslash vw$ is *not* k-node connected. Recall that an edge vw of a k-node connected graph H is *critical* (with respect to k-node connectivity) if $H \backslash vw$ is *not* k-node connected. The next result characterizes critical edges.

PROPOSITION 3.1. *An edge vw of a k-node connected graph H is not critical iff there are at least $k+1$ openly disjoint $v \leftrightarrow w$ paths in H (including the path vw).*

To find F efficiently, we start with $F = \emptyset$ and take the current subgraph to be $G = (V, E)$ (which is k-node connected). We examine the edges of $E \backslash M$ in an arbitrary order, say, $e_1, e_2, \ldots, e_\ell$ ($\ell = |E \backslash M|$). For each edge $e_i = v_i w_i$, we attempt to find $(k+1)$ openly disjoint $v_i \leftrightarrow w_i$ paths in the current subgraph. If we succeed, then we remove the edge e_i from the current subgraph (since e_i is not critical); otherwise, we retain e_i in the current subgraph and add e_i to F (since e_i is critical). At termination, the current subgraph with edge set $M \cup F$ is k-node connected, and every edge $vw \in F$ is critical. The running time for the second step is $O(k|E|^2)$.

The proof of the next lemma hinges on a theorem of Mader [Ma 72, Theorem 1]. For an English translation of the proof of Mader's theorem see Lemma I.4.4 and Theorem I.4.5 in [Bo 78].

THEOREM 3.2 (Mader [Ma 72, Theorem 1]). *In a k-node connected graph, a cycle consisting of critical edges must be incident to at least one node of degree k.*

LEMMA 3.3. $|F| \leq |V| - 1$.

Proof. Consider the k-node connected subgraph returned by the heuristic $G' = (V, E')$, where $E' = M \cup F$. Suppose that F contains a cycle C. Note that every

edge in the cycle is critical, since every edge in F is critical. Moreover, every node v incident to the cycle C has degree $\geq (k+1)$ in G', because v is incident to two edges of C, as well as to at least $(k-1)$ edges of $M = E'\backslash F$. But this contradicts Mader's theorem. We conclude that F is acyclic and so has $\leq |V|-1$ edges. The proof is done. □

PROPOSITION 3.4. *Let $G = (V, E)$ be a graph of node connectivity $\geq k$. The heuristic above finds a k-node connected spanning subgraph (V, E') such that $|E'| \leq (1 + [2/k])|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution. The running time is $O(k^3|V|^2 + |E|^{1.5}(\log |V|)^2)$.*

Proof. The approximation guarantee follows because $|E_{opt}| \geq (k|V|/2)$, so

$$\frac{|M|+|F|}{|E_{opt}|} = \frac{|M|}{|E_{opt}|} + \frac{|F|}{|E_{opt}|} \leq 1 + \frac{|V|}{(k|V|/2)} = 1 + [2/k].$$

We have already seen that M can be found in time $O(|E|^{1.5}(\log |V|)^2)$, and F can be found in time $O(k|E|^2)$. The running time of the second step can be improved to $O(k^3|V|^2)$ as follows: we run a linear-time preprocessing step to compute a sparse certificate $\widetilde{E}$ of G for k-node connectivity, i.e., $\widetilde{E} \subseteq E$, $|\widetilde{E}| \leq k|V|$, and for all nodes v, w, $(V, \widetilde{E})$ has k openly disjoint $v \leftrightarrow w$ paths iff G has k openly disjoint $v \leftrightarrow w$ paths; see [NI 92, FIN 93, CKT 93]. We compute M as before, by running the first step on G. To find the set $F \subseteq E\backslash M$, we run the second step on $\widetilde{E} \cup M$ rather than on E, and for each edge $v_i w_i \in \widetilde{E}\backslash M$, we attempt to find $(k+1)$ openly disjoint $v_i \leftrightarrow w_i$ paths in the current subgraph of $(V, \widetilde{E} \cup M)$. The second step runs in time $O(k|\widetilde{E} \cup M|^2) = O(k^3|V|^2)$, since $|\widetilde{E} \cup M| = O(k|V|)$. □

To improve the approximation guarantee to $1 + [1/k]$, we present an improved lower bound on $|E^*|$, where E^* denotes a minimum-cardinality edge set such that $G^* = (V, E^*)$ is k-edge connected. Suppose that E^* contains a perfect matching P_0 (so $|P_0| = n/2$). Then $|E^*| \geq (n/2) + \min\{|M^*| : M^* \subseteq E,\ \deg_{M^*}(v) \geq (k-1)\ \forall v \in V\}$. To see this, focus on the edge set $M' = E^*\backslash P_0$. Clearly, every node $v \in V$ is incident to at least $(k-1)$ edges of M', because $\deg_{E^*}(v) \geq k$ and $\deg_{P_0}(v) = 1$. Since M^* is a minimum-size edge set with $\deg_{M^*}(v) \geq (k-1)\ \forall v \in V$, we have $|M^*| \leq |M'| = |E^*| - (n/2)$. The next theorem generalizes this lower bound to the case when E^* has no perfect matching. The proof is given in the next subsection (section 3.2), after developing some preliminaries.

THEOREM 3.5. *Let $G^* = (V, E^*)$ be a graph of edge connectivity $\geq k \geq 1$, and let n denote $|V|$. Let $M^* \subseteq E^*$ be a minimum-size edge set such that every node $v \in V$ is incident to $\geq (k-1)$ edges of M^*. Then $|E^*| \geq |M^*| + \lfloor n/2 \rfloor$.*

THEOREM 3.6. *Let $G = (V, E)$ be a graph of node connectivity $\geq k$. The heuristic described above finds a k-node connected spanning subgraph (V, E') such that $|E'| \leq (1 + [1/k])|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution. The running time is $O(k^3|V|^2 + |E|^{1.5}(\log |V|)^2)$.*

Proof. The approximation guarantee of $1 + [1/k]$ follows easily from Theorem 3.5, using an argument similar to Proposition 3.4. We have $E' = M \cup F$, where $|F| \leq (n-1)$. Moreover, since M is a minimum-size edge set with $\deg_M(v) \geq (k-1)$, $\forall v \in V$, Theorem 3.5 implies that $|M| \leq |E_{opt}| - \lfloor n/2 \rfloor \leq |E_{opt}| - (n-1)/2$. Hence,

$$\frac{|M|+|F|}{|E_{opt}|} \leq \frac{|E_{opt}| - (n-1)/2 + (n-1)}{|E_{opt}|} \leq 1 + \frac{n/2}{|E_{opt}|} \leq 1 + [1/k],$$

where the last inequality uses the "degree lower bound," $|E_{opt}| \geq kn/2$.

The running time analysis is the same as that in Proposition 3.4. □

3.2. A lower bound for the size of a k-connected spanning subgraph and Gupta's theorem on bipartite graphs. This subsection gives a proof of Theorem 3.5. This theorem is used in the previous subsection to prove an approximation guarantee of $1 + [1/k]$ for a minimum-size k-NCSS. Theorem 3.5 gives the following new lower bound on the size of a k-ECSS:

> Let $G^* = (V, E^*)$ be a k-edge connected graph ($k \geq 1$), and let n denote $|V|$. Let $M^* \subseteq E^*$ be a minimum-size edge set such that every node $v \in V$ is incident to $\geq (k-1)$ edges of M^*. Then $|E^*| \geq |M^*| + \lfloor n/2 \rfloor$.

First, a theorem of Gupta on bipartite graphs is recalled. For the special case of bipartite graphs, (a stronger form of) the lower bound in Theorem 3.5 follows easily from Gupta's theorem; see Proposition 3.8. This proposition is used in section 3.4 to prove an approximation guarantee of $1+[1/k]$ for a minimum-size k-NCSS of a digraph. Gupta's theorem does not apply to nonbipartite graphs. The proof of Theorem 3.5 (for arbitrary graphs) relies on the Gallai–Edmonds decomposition theorem of matching theory. When the Gallai–Edmonds decomposition of the graph is "nontrivial," one can define a bipartite graph B that partially represents the decomposition. The proof of Theorem 3.5 is completed by examining B. One way is to prove a variant of Gupta's theorem (see Proposition 3.9), and then apply it to B. This is described below. Readers interested in a detailed study of the proofs in this subsection may find it useful to review two results in matching theory, namely, the Gallai–Edmonds decomposition theorem [LP 86, Theorem 3.2.1] and the Hungarian method for bipartite matching [LP 86, Lemma 1.2.2].

THEOREM 3.7 (see Gupta [Gu 67]). *Let $B = (X \cup Y, E)$ be a bipartite graph with minimum degree k. Then there exists a partition of the edge set of B, namely E, into k sets $E_1, E_2, \ldots, E_k$ such that each node $v \in X \cup Y$ is incident to at least one edge from each set E_i, $1 \leq i \leq k$.*

For an elegant proof, see the solutions to Problems 10–12 in [L 93, Chapter 7]. Also, see [BM 76, Problem 6.1.6]. The next result strengthens Theorem 3.5 for bipartite graphs. The proof is via Gupta's theorem. Another brief proof follows from Proposition 3.10.

PROPOSITION 3.8. *Let $B^* = (X \cup Y, E^*)$ be a bipartite graph with minimum degree $\geq k$. Let $M^* \subseteq E^*$ be a minimum-size edge set such that every node $v \in X \cup Y$ is incident to $\geq k-1$ edges of M^*. Then $|E^*| \geq |M^*| + (|X \cup Y|/2)$.*

Proof. Apply Gupta's theorem to E^*, and let $E_1, E_2, \ldots, E_k$ be the partition of E^*. Focus on the set, say E_k, that has the maximum cardinality. Clearly, $|E_k| \geq |E^*|/k \geq |X \cup Y|/2$. Now, consider $M' = E^* \backslash E_k$, and observe that each node $v \in X \cup Y$ is incident to $\geq (k-1)$ edges of M', because Gupta's result shows that v is incident to some edge from each of the remaining $(k-1)$ sets $E_1, E_2, \ldots, E_{k-1}$. The proof is done since $|E^*| - (|X \cup Y|/2) \geq |M'|$ and $|M'| \geq |M^*|$. □

Proposition 3.8 does *not* generalize to nonbipartite graphs B^*, even if we strengthen the condition "B^* has minimum degree $\geq k$" to "B^* is k-edge connected." For example, let $k = 2$, and let $B^* = K_3$, the complete graph on three nodes. Then M^* is a minimum edge cover of K_3 and has size 2. But then $|E^*| = |M^*|+1 < |M^*|+(|V|/2)$. The generalization of Proposition 3.8 fails because B^* is a 2-edge connected, 2-regular graph such that for every edge cover M^*, the edge-complement of M^* in B^*, $(V, E^* - M^*)$, has an isolated node, so it does not have an edge cover. For every *even* integer $k \geq 2$, there is an infinite family of nonbipartite graphs such that the generalization of Proposition 3.8 fails. Take B^* to be a k-edge connected, k-regular

graph with an *odd* number of nodes n. Then M^* has size at least $(1+(k-1)n)/2$, so $(V, E^* - M^*)$ has an isolated node, and hence has size $< n/2$. It can be seen that the examples in this paragraph are factor-critical graphs.

The next proposition may be regarded as a variant of Gupta's theorem. Note that the bipartite graph B in the next proposition may have minimum degree 1, and B may have multiple copies of an edge.

PROPOSITION 3.9. *Let $B = (X \cup Y, E)$ be a bipartite (loopless) multigraph with node bipartition $X \cup Y$. Let each node $y \in Y$ have* $\deg(y) \geq k$*,and let B have a matching of size $|X|$. Then B has an edge cover J such that each node $y \in Y$ is incident to exactly one edge of J, and each node $x \in X$ is incident to either exactly one edge of J or at least $(k-1)$ edges of $E \backslash J$.*

Proof. See Figure 3(b) for an illustration. Let J_0 be a matching of size $|X|$. The edge cover J is constructed iteratively, starting with $J' = J_0$ and $J'' = \emptyset$. Throughout, J' is a matching of the current B, and at the end of the construction, $J' \cup J''$ is an edge cover of the original B that satisfies the proposition.

If $J' \cup J''$ is an edge cover, i.e., if J' is a perfect matching, then the proof is completed by taking $J = J' \cup J''$. Clearly, the degree requirements in the proposition hold. Otherwise, if $J' \cup J''$ is not an edge cover, the size of $J' \cup J''$ is increased by one such that one more Y-node is covered and the degree requirements in the proposition are maintained. Let $v \in Y$ be a node that is *not* covered by $J' \cup J''$. Let T be the node set of the maximal J'-alternating tree that contains v. That is, a node w is in T iff there exists a J'-alternating path between v and w. (For a matching J', recall that a J'-alternating path means a path whose edges are alternately in J' and not in J'.)

Claim 1. There is a node $x \in T \cap X$ with $\deg(x) \geq k+1$.

To prove this claim, note that (i) $|T \cap Y| = |T \cap X| + 1$ (since each node $y \in T \cap Y$ except v is incident to an edge of J'), and (ii) for every node $y \in T \cap Y$, every incident edge wy has the other end node w in $T \cap X$ (otherwise, w can be added to T, and so T is not maximal). By assumption, each node $y \in T \cap Y$ has $\deg(y) \geq k$, hence, (i), (ii), and the pigeonhole principle guarantee that there is a node $x \in T \cap X$ with $\deg(x) > k$. This proves the claim.

Let xz be the J'-edge incident to x, i.e., x is matched to z by J'. This edge is (permanently) added to the edge cover J by taking $J'' = J'' \cup \{xz\}$. The node z is deleted from B. Since $x \in T$, there exists a J'-alternating path between v and x (by definition of T). Let this path be P'. The matching J' is updated by switching alternate edges along P', i.e., J' is replaced by the symmetric difference of J' and $E(P')$. Note that the current B (with node z deleted) has a matching of size $|X|$, namely J', and has $\deg(y) \geq k$, for all nodes $y \in V(B) \backslash X$. Therefore, the hypothesis of the proposition continues to hold.

The above step is repeated until $J' \cup J''$ covers all nodes of B. Finally, J is taken to be $J' \cup J''$. The construction guarantees that J satisfies the degree requirements in the proposition. □

Recall the Gallai–Edmonds decomposition theorem of matching theory [LP 86, Theorem 3.2.1]. For every graph H, there is a partition of $V(H)$ into a set of (matching) noncritical nodes $D(H)$ and a set of (matching) critical nodes $V \backslash D(H)$ (i.e., $D(H)$ consists of all nodes that are left uncovered by some maximum matching of H). The partition is "trivial" either if H has a perfect matching, or if H is factor-critical: in the first case, $D(H) = \emptyset$, and in the second case, $D(H) = V(H)$. Let $A(H)$ be the set of critical nodes of H that are adjacent to one or more noncritical nodes of H. Possibly, $A(H)$ is the empty set. When there is no danger of confu-

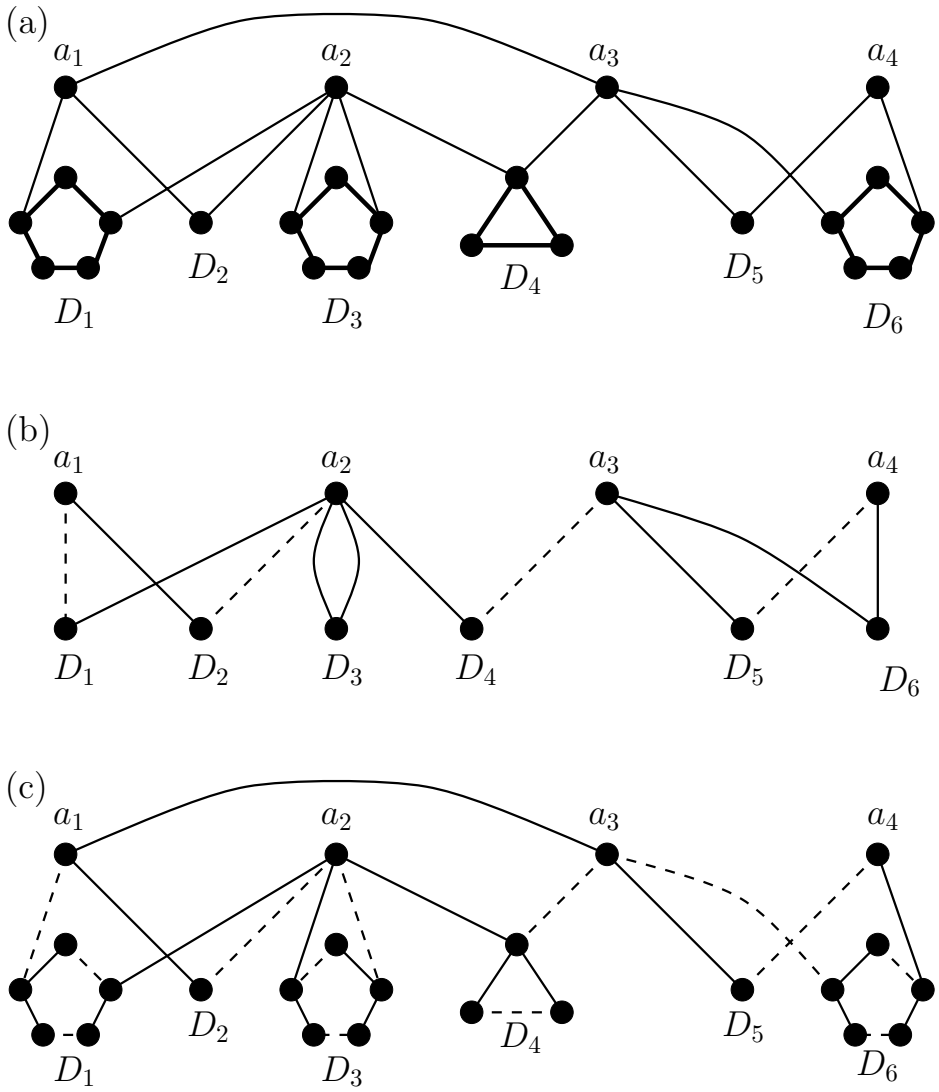


FIG. 3. *An illustration of the proofs of Theorem* 3.5 *and Propositions* 3.9 and 3.10. (a) $G = (V, E)$ *is a* 2*-edge connected graph* ($k = 2$)*, and the Gallai–Edmonds decomposition is given by* $A = A(G) = \{a_1, a_2, a_3, a_4\}$ *and* $D = D(G) = V(D_1) \cup V(D_2) \cup V(D_3) \cup V(D_4) \cup V(D_5) \cup V(D_6)$. *The odd (factor-critical) components of* $G \backslash A$ *are* $D_1, \ldots, D_6$. (b) *The bipartite multigraph* B *in the proofs of Propositions* 3.9 and 3.10. *In Proposition* 3.10, B *is obtained from* G *by deleting the nodes in* $V \backslash (A \cup D)$ *and the edges in* $E(A)$*, and shrinking* $D_1, \ldots, D_6$ *into single nodes. In* B*, note that* $\deg(D_1), \ldots, \deg(D_6) \geq k = 2$*, and there is a matching* J' *of size* $|A| = 4$. J' *is indicated by dashed lines,* $J' = \{a_1D_1, a_2D_2, a_3D_4, a_4D_5\}$. *In the construction of Proposition* 3.9*, the first iteration chooses, say,* $v = D_3$. *Then* $T = \{D_3, a_2, D_2, a_1, D_1\}$ *and* $x = a_2 \in T \cap A$ *has degree* $\geq k + 1 = 3$. *The edge* a_2D_2 *is added to* J''*, the node* D_2 *is deleted, and in* J'*,* a_2D_2 *is replaced by* a_2D_3. *Finally,* $J' = \{a_1D_1, a_2D_3, a_3D_6, a_4D_5\}$, $J'' = \{a_2D_2, a_3D_4\}$, *and* $J = J' \cup J''$ *is the required edge cover.* (c) *In* G, J *maps to an edge set* $\widetilde{J}$. $\widetilde{J}$ *is extended to the required edge cover* P *of* G *by adding a perfect matching on the nodes of* G *not incident to* $\widetilde{J}$. P *is indicated by dashed lines.*

sion, we use A and D instead of $A(H)$ and $D(H)$. Let $\mathrm{def}(H)$ denote the deficiency of H, i.e., the number of nodes that are not covered by a maximum matching of H. (So, $\mathrm{def}(H) = |V(H)| - 2|P_0|$, where P_0 is a maximum matching of H.) The

Gallai–Edmonds decomposition theorem shows that in the graph $H\backslash A$, the noncritical nodes D form $q = |A(H)| + \mathrm{def}(H)$ odd components $D_1, D_2, \ldots, D_q$, i.e., each D_i $(i = 1, \ldots, q)$ is a connected component of $H\backslash A$ with $V(D_i) \subseteq D(H)$ and $|V(D_i)|$ odd. Moreover, every one of these odd components D_i is factor-critical.

The next result is a generalization of Proposition 3.9.

PROPOSITION 3.10. *Let G be a graph, and let $D = D(G)$ and $A = A(G)$ be the node sets in the Gallai–Edmonds decomposition. Let $q = |A(G)| + \mathrm{def}(G)$, and let $D_1, D_2, \ldots, D_q$ be the odd components of $G\backslash A$. If every D_i gives a cut containing at least k edges, i.e., if $\delta(V(D_i))$ has size $\geq k$ for $i = 1, \ldots, q$, then G has an edge cover P such that each node in $V(G)\backslash A$ is incident to exactly one edge of P, and each node in A is incident to either exactly one edge of P or at least $(k-1)$ edges of $E(G)\backslash P$.*

Proof. See Figure 3 for an illustration. The proof follows easily by applying Proposition 3.9 to a bipartite graph associated with the Gallai–Edmonds decomposition.

If $\mathrm{def}(G) = 0$, then the proof is done: take P to be a perfect matching of G. Otherwise, $\mathrm{def}(G) > 0$, and so $D \neq \emptyset$. Suppose that $A = \emptyset$. Then every component D_i of G is factor-critical, but this violates the condition on $|\delta(V(D_i))|$. Hence, A is nonempty. Clearly, every edge in $\delta(V(D_i))$ $(i = 1, \ldots, q)$ has one end node in A and the other in D_i. Let $G[A \cup D]$ be the subgraph of G induced by $A \cup D$. Let $B = (X \cup Y, E')$, $X = A$, be the bipartite (loopless) multigraph obtained from $G[A \cup D]$ by deleting all edges with both end nodes in A and by shrinking the components $D_1, D_2, \ldots, D_q$ of $G[A \cup D]\backslash A$ to single nodes. The shrunken nodes are also called $D_1, D_2, \ldots, D_q$, and so $Y = \{D_1, D_2, \ldots, D_q\}$. B has $\geq k$ edges incident to each of the shrunken nodes $D_1, D_2, \ldots, D_q$, since in G each of the cuts $\delta(V(D_i))$ $(i = 1, \ldots, q)$ has $\geq k$ edges. Moreover, B has a matching of size $|X| = |A|$, by the Gallai–Edmonds decomposition theorem. Therefore, B satisfies the conditions in Proposition 3.9. By the proposition, B has an edge cover J satisfying the degree requirements in the proposition; note that each node $D_i \in Y$ is incident to exactly one edge of J. Let $\widetilde{J}$ denote a set of edges of G that corresponds to J, i.e., for each edge $a_h D_i \in J$ with $a_h \in X = A$, $D_i \in Y$, there is an edge $a_h w_i \in \widetilde{J}$ such that (in G) w_i is a node in D_i and w_i is adjacent to a_h. Let $V(\widetilde{J})$ be the set of nodes of G incident to edges in $\widetilde{J}$, i.e., $V(\widetilde{J}) = A \cup \{w_i \in V(D_i) : i = 1, \ldots, q\}$. By the Gallai–Edmonds decomposition theorem, $G\backslash V(\widetilde{J})$ has a perfect matching $\widetilde{P}$. To see this, note that each component of $G\backslash V(\widetilde{J})$ is either an even component of $G\backslash A$ or is obtained by deleting one node from an odd (factor-critical) component of $G\backslash A$; in either case, the component has a perfect matching.

Take $P = \widetilde{J} \cup \widetilde{P}$. Clearly, P is an edge cover of G such that each node $v \in V\backslash A$ is incident to exactly one edge of P. Moreover, by Proposition 3.9, every node in A is incident to either exactly one edge of P or to $\geq (k-1)$ edges of $E\backslash P$. □

Proof of Theorem 3.5. See Figure 3 for an illustration. We construct an appropriate edge set P^* such that $|P^*| \geq \lfloor n/2 \rfloor$ and every node $v \in V$ is incident to $\geq (k-1)$ edges of $E^*\backslash P^*$. In the statement of Theorem 3.5, note that M^* is a minimum-size edge set such that (V, M^*) has minimum degree $(k-1)$. Hence, $|E^*\backslash P^*| \geq |M^*|$. The theorem follows immediately from the existence of the edge set P^*, because $|E^*| = |E^*\backslash P^*| + |P^*| \geq |E^*\backslash P^*| + \lfloor n/2 \rfloor \geq |M^*| + \lfloor n/2 \rfloor$.

If the size of a maximum matching of G^* is $\geq (n-1)/2$, i.e., if G^* has a matching that leaves at most one node uncovered, then we take P^* to be a maximum matching. (This handles the case when G^* is a factor-critical graph.)

To handle the case when $\mathrm{def}(G^*) \geq 2$, we apply Proposition 3.10 to G^*, noting that G^* satisfies the conditions in the proposition. (Since G^* is k-edge connected, $\deg(v) \geq k\ \forall v \in V$, and every node set $S \subseteq V$, $\emptyset \neq S \neq V$, has $|\delta(S)| \geq k$.) We take P^* to be the edge cover P guaranteed by the proposition. Since P^* is an edge cover of G^*, $|P^*| \geq n/2$. Moreover, $(V, E^* \backslash P^*)$ has minimum degree $\geq k-1$ by the proposition and the fact that G^* has minimum degree $\geq k$. The theorem follows. □

We mention a corollary of Theorem 3.5, though this is not relevant to the main theme of the paper.

COROLLARY 3.11 (Petersen's theorem). *A 3-regular graph without cut edges has a perfect matching.*

Proof. Let $G^* = (V, E^*)$ be the graph, and let $n = |V|$. Clearly, n is even, and $|E^*| = 3n/2$. The key point is that every node set S of odd cardinality (i.e., $S \subset V$ and $|S|$ odd) has $|\delta(S)| \geq 3$ since $|\delta(S)|$ is *odd* (since $3|S| - 2|E(S)|$ is odd) and is ≥ 2. Suppose that G^* has no perfect matching. Then $\mathrm{def}(G^*) > 0$, and so in the Gallai–Edmonds decomposition we have $D(G^*) \neq \emptyset$; moreover, G^* is not factor-critical (n is even) so $A(G^*) \neq \emptyset$. Applying Proposition 3.10 with $k = 3$ shows that G^* has an edge cover P such that every node is incident to $\geq (k-1) = 2$ edges of $M = E^* \backslash P$. Clearly, $|P| \geq n/2$, since P is an edge cover, and $|M| = |E^* \backslash P| \geq n$, since (V, M) has minimum degree 2. Since $|E^*| = |P| + |M| = 3n/2$, we have $|P| = n/2$ and $|M| = n$. Therefore, P is a perfect matching of G^*. □

3.3. Minimum-size 2-connected spanning subgraphs of undirected graphs: A parallel $(1.5 + \epsilon)$-approximation algorithm. This subsection focuses on the design of an efficient parallel algorithm and a linear-time sequential algorithm for the problem of finding a minimum-size 2-node connected (2-edge connected) spanning subgraph of a graph. Let $\epsilon > 0$ be a constant, independent of $|V(G)|$. A deterministic parallel version of the main heuristic runs in **NC** and achieves an approximation guarantee of $(1.5 + \epsilon)$, whereas a randomized **NC** version achieves an approximation guarantee of 1.5. A sequential *linear-time* version of the main heuristic achieves an approximation guarantee of $(1.5 + \epsilon)$. The proof of the 1.5 approximation guarantee in this subsection again hinges on Mader's theorem (Theorem 3.2), but instead of employing the lower bound in Theorem 3.5, we employ a nice lower bound result due to Chong and Lam (Proposition 3.13).

The heuristic for a minimum-size 2-NCSS described below can be used to find a 1.5-approximation of a minimum-size 2-ECSS. For this, we run a preprocessing step on the given graph $G = (V, E)$, which is assumed to be 2-edge connected, to partition the edge set into blocks (maximal 2-node connected subgraphs). Then separately for each block, we run our heuristic for a minimum-size 2-NCSS. For a block, the optimal 2-ECSS may *not* be 2-node connected; nevertheless, the lower bound used by the 2-NCSS heuristic applies to 2-ECSS too, so the edge set found by our algorithm will have size within 1.5 times the minimum size of a 2-ECSS.

Consider the problem of approximating a minimum-size 2-NCSS. Assume that the given graph $G = (V, E)$ is 2-node connected. The heuristic consists of two steps. The first finds a *minimum edge cover* $M \subseteq E$ of G, i.e., a minimum-cardinality edge set such that every node is incident to at least one edge of M. One way of finding M is to start with a *maximum matching* $\widetilde{M}$ of G, and then to add one edge incident to each node that is not matched by $\widetilde{M}$. Recall that $\mathrm{def}(G)$ denotes the number of nodes not matched by a maximum matching of G, i.e., $\mathrm{def}(G) = |V| - 2|\widetilde{M}|$. Then we have $|M| = |\widetilde{M}| + \mathrm{def}(G)$. (It is easily seen that no edge cover of G has smaller cardinality

than $|\widetilde{M}| + \mathrm{def}(G)$.) The second step of the heuristic finds an (inclusionwise) minimal edge set $F \subseteq E \backslash M$ such that $M \cup F$ gives a 2-NCSS. In other words, $(V, M \cup F)$ is 2-node connected, but for each edge $vw \in F$, $(V, M \cup F) \backslash vw$ is *not* 2-node connected. Let E' denote $M \cup F$, and let $E_{opt} \subseteq E$ denote a minimum-cardinality edge set such that (V, E_{opt}) is 2-*edge* connected.

LEMMA 3.12. $|E'| = |M| + |F| \leq 1.5|V| + \mathrm{def}(G) - 1$.

Proof. By Mader's theorem (Theorem 3.2), F is acyclic, so $|F| \leq |V| - 1$. A minimum edge cover M of G has size $|M| = |\widetilde{M}| + \mathrm{def}(G)$, where $\widetilde{M}$ is a maximum matching of G. Obviously, $|\widetilde{M}| \leq |V|/2$. The result follows. □

The next result, due to Chong and Lam, gives a lower bound on the size of a 2-ECSS. Proposition 3.14 generalizes Chong and Lam's lower bound to k-edge connected spanning subgraphs, $k \geq 1$.

PROPOSITION 3.13 (see Chong and Lam [CL 95, Lemma 3]). *Let $G = (V, E)$ be a graph of edge connectivity ≥ 2, and let $|E_{opt}|$ denote the minimum size of a 2-edge connected spanning subgraph. Then $|E_{opt}| \geq \max(|V| + \mathrm{def}(G) - 1, |V|)$.*

PROPOSITION 3.14. *Let $G = (V, E)$ be a graph of edge connectivity $\geq k \geq 1$, and let $|E_{opt}|$ denote the minimum size of a k-edge connected spanning subgraph. If G is not factor-critical, then $|E_{opt}| \geq \frac{k}{2}(|V| + \mathrm{def}(G))$. In general, $|E_{opt}| \geq \frac{k}{2}\max(|V| + \mathrm{def}(G) - 1, |V|)$.*

Proof. Suppose that G is *not* factor-critical and $\mathrm{def}(G)$ is ≥ 1. Then, by the Gallai–Edmonds decomposition theorem of matching theory [LP 86, Theorem 3.2.1], there is a *nonempty* node set A such that $G \backslash A$ has $|A| + \mathrm{def}(G)$ odd components ($G \backslash A$ may have some even components too). Focus on a (odd or even) component D_i of $G \backslash A$. The number of edges of E_{opt} such that either one or both end nodes are in D_i is at least $(|V(D_i)| + 1)k/2$, because every node $v \in V(D_i)$ is incident to $\geq k$ edges of E_{opt}, and moreover, $\delta(V(D_i))$ has at least k edges of E_{opt}. Summing over all components D_i of $G \backslash A$ proves the proposition. □

THEOREM 3.15. *Let $G = (V, E)$ be a graph of node (edge) connectivity ≥ 2. Let $\epsilon > 0$ be a constant. The heuristic described above finds a 2-node connected (2-edge connected) spanning subgraph (V, E') such that $|E'| \leq 1.5|E_{opt}|$, where $|E_{opt}|$ denotes the minimum size of a 2-ECSS.*

A randomized parallel version of the heuristic runs in **RNC** *and achieves an approximation guarantee of* 1.5. *A deterministic parallel version of the heuristic runs in* **NC** *and achieves an approximation guarantee of* $(1.5 + \epsilon)$.

The sequential running time is $O(\sqrt{|V|}|E|)$. A sequential linear-time version of the heuristic achieves an approximation guarantee of $(1.5 + \epsilon)$.

Proof. The approximation guarantee follows from Lemma 3.12 and Proposition 3.13, since

$$\frac{|E'|}{|E_{opt}|} \leq \frac{1.5|V| + \mathrm{def}(G) - 1}{\max(|V| + \mathrm{def}(G) - 1, |V|)} \leq 1 + \frac{0.5|V|}{|V|} \leq 1.5.$$

Consider the deterministic parallel version of the heuristic. Let $\widetilde{M}$ denote a maximum matching of G. For step 1, we find an approximately maximum matching in **NC** using the algorithm of [FGHP 93]: for a constant ϵ, $0 < \epsilon < 0.5$, the algorithm finds a matching M' with $|M'| \geq (1 - 2\epsilon)|\widetilde{M}|$ in parallel time $O(\epsilon^{-4}(\log |V|)^3)$ using $O(\epsilon^{-1}|V|^{2+(2/\epsilon)})$ processors. We obtain an (inclusionwise) minimal edge cover M of size $\leq (1 + 2\epsilon)|\widetilde{M}| + \mathrm{def}(G)$ by adding to M' one edge incident to every node that is not matched by M'. For step 2, we use a variant of the **NC** algorithm of [HKe+ 95, KeR 95], see Algorithm 2 and Lemma 2 in Kelsen and Ramachandran

[KeR 95]. Let G' be a 2-node connected spanning subgraph of G such that $E(G')$ contains the minimal edge cover M. Call an edge vw of G' *essential* if either vw is in M or $G'\backslash vw$ is not 2-node connected (i.e., an edge of G' is nonessential if it is not in M and it is not critical with respect to the 2-node connectivity of G'). Algorithm 2 of [KeR 95] starts by taking the current subgraph G' to be G and repeatedly finds a spanning tree T of G' that has the minimum number of nonessential edges, *minimally* augments T to obtain a 2-node connected spanning subgraph G'' of G', and then replaces the current subgraph G' by G''. Finding the spanning tree T is easy: we compute a minimum spanning tree of G' where the cost of each edge in M is taken to be (-1), the cost of each remaining essential edge of G' is 0, and the cost of each nonessential edge of G' is 1. The parallel complexity of the whole algorithm is in **NC**; see [HKe+ 95, KeR 95]. Now, the approximation guarantee is $(1.5+\epsilon)$.

For the sequential linear-time version of the heuristic, note that a matching M' with $|M'| \geq (1-2\epsilon)|\widetilde{M}|$ can be found in time $O((|V|+|E|)/\epsilon)$. Moreover, in linear time, we can find a minimal 2-node connected spanning subgraph whose edge set contains the minimal edge cover $M \subseteq E$ obtained by adding edges to M'; see [HKe+ 95]. □

3.4. Directed graphs. The main heuristic extends to digraphs. The key tool in the analysis of the approximation guarantee is another theorem of Mader [Ma 85, Theorem 1]. Given a digraph $G=(V,E)$ that is assumed to have node connectivity at least k, the first step of the heuristic finds an arc set $M \subseteq E$ of minimum cardinality such that for every node v, there are $\geq (k-1)$ arcs of M going out of v and $\geq (k-1)$ arcs of M coming into v. Clearly, $|M| \leq |E_{opt}|$, where $E_{opt} \subseteq E$ denotes a minimum-cardinality arc set such that (V, E_{opt}) is k-node connected. The second step of the heuristic is as in section 3.1: we find an (inclusionwise) minimal arc set $F \subseteq E\backslash M$ such that $M \cup F$ is the arc set of a k-node connected spanning subgraph. The key point is that $|F| \leq 2|V|-1$, by Mader's digraph theorem (Theorem 3.16).

Consider the first step in more detail. To find the arc set M, we transform the digraph problem to a b-matching problem on the bipartite graph $B(G)$ associated with G. For each node $v \in V(G)$, there is a pair of nodes v_-, v_+ in the bipartite graph $B(G)$, and for each arc (v,w) of G there is one edge v_+w_- in the bipartite graph. Our problem of finding a minimum-cardinality $M \subseteq E$ with $\deg_{M,in}(v) \geq (k-1)$, $\deg_{M,out}(v) \geq (k-1)$ $\forall v \in V$ corresponds to the problem of finding a minimum-cardinality edge set M' of the bipartite graph such that each node of the bipartite graph is incident to $\geq (k-1)$ edges of M'. As in section 3.1, this is a b-matching problem.

An *alternating cycle* of a digraph is a nonempty, even-length sequence of distinct arcs $C = e_1, e_2, \ldots, e_{2\ell-1}, e_{2\ell}$, $\ell \geq 1$, such that (using indices modulo 2ℓ) for each $i = 0, 1, \ldots$, the arcs e_{2i} and e_{2i+1} have the same start node, and the arcs e_{2i+1} and e_{2i+2} have the same end node. In other words, the set of undirected edges corresponding to an alternating cycle C is a union of cycles, and moreover, alternate occurrences of nodes have two C-arcs coming out or two C-arcs going in. See Figure 4 for an illustration. For an alternating cycle C, a C-out node is a node having two outgoing arcs of C, and a C-in node is a node having two incoming arcs of C. Recall that an arc e of a k-node connected digraph H is called critical if $H\backslash e$ is not k-node connected. Here is Mader's theorem on the critical arcs of a k-node connected digraph; see Figure 4 for an illustration.

THEOREM 3.16 (see Mader [Ma 85, Theorem 1]). *In a k-node connected digraph, if there is an alternating cycle C each of whose arcs is critical, then there is either a C-out node of outdegree k or a C-in node of indegree k.*

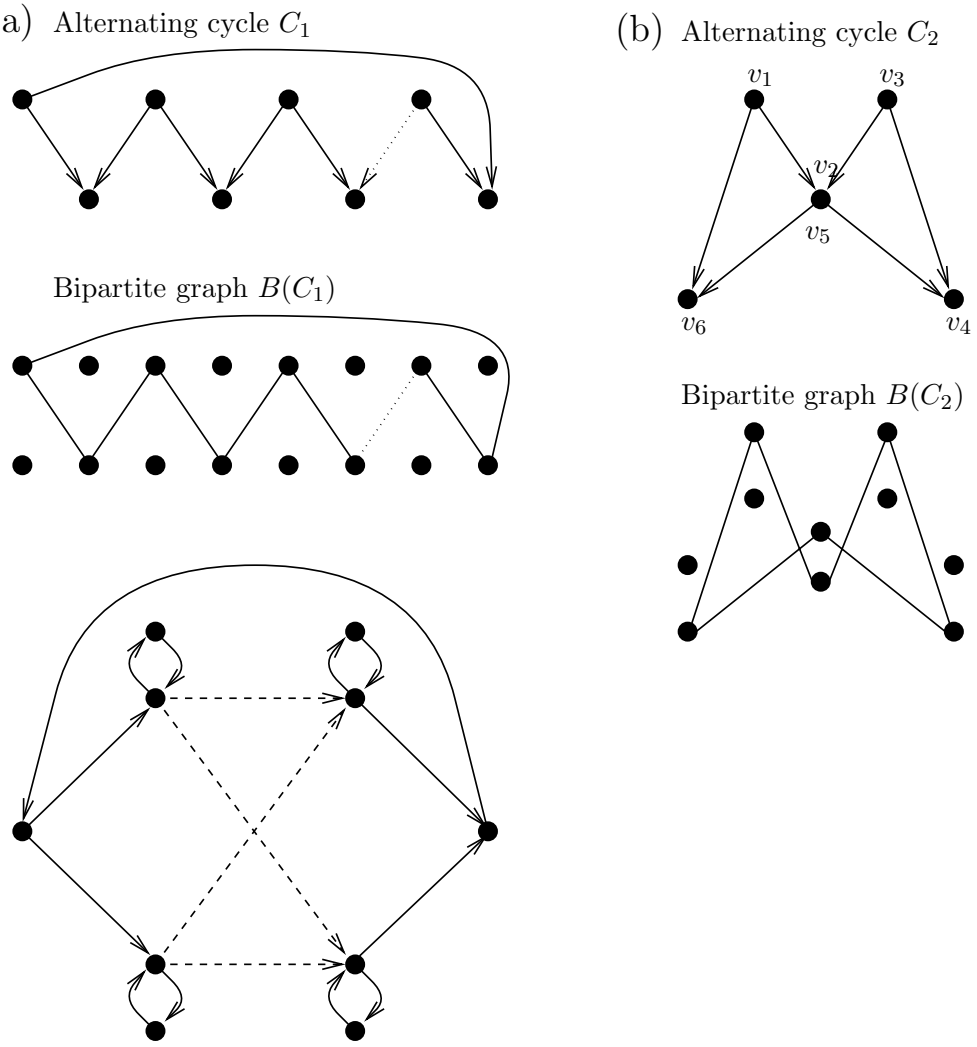


FIG. 4. *An illustration of an alternating cycle in a digraph and of Mader's theorem on critical alternating cycles in a k-node connected digraph; see Theorem* 3.16. (a) *An alternating cycle* C_1 *and its bipartite graph* $B(C_1)$. (b) *Another alternating cycle* $C_2 = (v_1, v_2), (v_3, v_2), (v_3, v_4), (v_5, v_4), (v_5, v_6), (v_1, v_6)$ *and its bipartite graph* $B(C_2)$. *For an alternating cycle, the undirected version may not be a cycle, but the bipartite graph has at least one cycle.* (c) *An alternating cycle* C *of a* 1*-connected (strongly connected) digraph is indicated by dashed lines. Every* C*-out node has outdegree* $> k = 1$*, and every* C*-in node has indegree* $> k = 1$*. None of the arcs in the alternating cycle is critical for* 1*-connectivity. This example is modified from an example of Mader* [Ma 85].

FACT 3.17 (see Mader [Ma 85, Lemma 2]). *Let* H *be a digraph, and let* $B(H)$ *be the associated bipartite graph. There is a cycle in* $B(H)$ *iff there is an alternating cycle in* H.

Remarks. Mader [Ma 85] states the theorem for minimal k-node connected digraphs, but in fact, his proof needs only the fact that every arc in the alternating cycle

is critical. Now, consider a digraph H_0 that is obtained from an arbitrary strongly connected digraph by subdividing every arc at least once (i.e., an arc is replaced by ≥ 1 new nodes and a directed path of ≥ 2 arcs). Note that H_0 contains no alternating cycle. Mader [Ma 85, p. 104] shows that there exists a minimal k-node connected digraph G such that H_0 is contained in the subgraph of G induced by arcs whose start nodes have outdegrees $> k$ and whose end nodes have indegrees $> k$.

LEMMA 3.18. *Let $F \subseteq E \backslash M$ be the set of critical arcs found by the second step of the heuristic. Then $|F| \leq 2|V| - 1$.*

Proof. Let $G' = (V, E')$, where $E' = M \cup F$. We claim that F contains no alternating cycle. By way of contradiction, suppose that $C \subseteq F$ is an alternating cycle. Observe that every C-out node v has $\geq (k+1)$ outgoing arcs of E', since there are $\geq (k-1)$ arcs of M outgoing from v, and there are two arcs of C outgoing from v. Similarly, every C-in node has $\geq (k+1)$ incoming arcs of E'. This contradicts Mader's digraph theorem. Hence, F contains no alternating cycle. Then $|F| \leq 2|V| - 1$, because the bipartite graph associated with (V, F) is acyclic. □

The previous lemma immediately gives an approximation guarantee of $1 + [2/k]$ for a minimum-size k-NCSS of a digraph, because the "degree lower bound" implies that a digraph k-NCSS has $\geq k|V|$ arcs. The approximation guarantee can be improved to $1 + [1/k]$ via the lower bound on the size of a digraph k-NCSS implied by Proposition 3.8.

PROPOSITION 3.19. *Let $G = (V, E)$ be a digraph of node connectivity $\geq k$. The heuristic above finds a k-node connected spanning subgraph (V, E') such that $|E'| \leq (1 + [2/k])|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution.*

THEOREM 3.20. *Let $G = (V, E)$ be a digraph of node connectivity $\geq k$. The heuristic described above finds a k-node connected spanning subgraph (V, E') such that $|E'| \leq (1 + [1/k])|E_{opt}|$, where $E_{opt} \subseteq E$ denotes a minimum-cardinality arc set such that (V, E_{opt}) is k-node connected. The running time is $O(k|E|^2)$.*

Proof. The proof of the approximation guarantee is similar to the proof for undirected graphs in Theorem 3.6. Let $G_{opt} = (V, E_{opt})$ be a k-node connected spanning subgraph of minimum size. Apply Proposition 3.8 to the bipartite graph $B(G_{opt})$ of G_{opt} to deduce that $|M^*| \leq |E(B(G_{opt}))| - |V(B(G_{opt}))|/2$, where $M^* \subseteq E(B(G_{opt}))$ is a minimum-size edge set such that every node of $B(G_{opt})$ is incident to $\geq k - 1$ edges of M^*. Since the arc set $M \subseteq E(G)$ found by the heuristic has $|M| \leq |M^*|$ (since M comes from a supergraph of E_{opt}), it follows that $|M| \leq |E(B(G_{opt}))| - |V(B(G_{opt}))|/2 = |E_{opt}| - |V(G)|$. Consequently, since $|E'| = |M| + |F|$ and $|F| \leq 2|V(G)| - 1$,

$$\frac{|E'|}{|E_{opt}|} \leq \frac{|E_{opt}| - |V(G)| + (2|V(G)| - 1)}{|E_{opt}|} \leq 1 + \frac{1}{k},$$

where the last inequality uses the "degree lower bound," $|E_{opt}| \geq k|V(G)|$. The running time analysis is similar to that for the heuristic for graphs; see section 3.1. □

4. Approximating minimum-size k-edge connected spanning subgraphs. The heuristic can be modified to find an approximately minimum-size k-edge connected spanning subgraph (abbreviated k-ECSS) of a graph or a digraph. First, we focus on graphs, and we prove a $(1 + [2/(k+1)])$-approximation guarantee for finding a minimum-size k-ECSS. The analysis hinges on Theorem 4.3 which may be regarded as an analogue of Mader's theorem [Ma 72, Theorem 1] for k-edge connected graphs. Then we turn to digraphs, and prove an approximation guarantee of $1 + [4/\sqrt{k}]$ for the k-ECSS heuristic.

In this section, an edge e (arc e) of a k-edge connected graph (digraph) H is called *critical* if $H\backslash e$ is *not* k-edge connected. Assume that the given graph or digraph $G = (V, E)$ is k-edge connected; otherwise, the heuristic will detect this and report failure.

4.1. Undirected graphs. In this subsection, $G = (V, E)$ is a graph. The first step of the heuristic finds an edge set $M \subseteq E$ of minimum cardinality such that every node in V is incident to $\geq k$ edges of M. Clearly, $|M| \leq |E_{opt}|$, where $E_{opt} \subseteq E$ denotes a minimum-cardinality edge set such that (V, E_{opt}) is k-edge connected. The second step of the heuristic finds an (inclusionwise) minimal edge set $F \subseteq E\backslash M$ such that $M \cup F$ is the edge set of a k-edge connected spanning subgraph. In detail, the second step starts with $F = \emptyset$ and $E' = E$. Note that $G' = (V, E')$ is k-edge connected at the start. We examine the edges of $E\backslash M$ in an arbitrary order $e_1, e_2, \ldots$. For each edge $e_i = v_i w_i$ (where $1 \leq i \leq |E\backslash M|$), we determine whether or not $v_i w_i$ is critical for the current graph by finding the maximum number of edge-disjoint $v_i \leftrightarrow w_i$ paths in G'.

PROPOSITION 4.1. *An edge $v_i w_i$ of a k-edge connected graph is not critical iff there exist at least $k+1$ edge-disjoint $v_i \leftrightarrow w_i$ paths (including the path $v_i w_i$).*

If $v_i w_i$ is noncritical, then we delete it from E' and G'; otherwise, we retain it in E' and G', and also we add it to F. At termination of the heuristic, $G' = (V, E')$, $E' = M \cup F$, G' is k-edge connected, and every edge $vw \in F$ is critical, i.e., $G'\backslash vw$ is *not* k-edge connected. Theorem 4.3 below shows that $|F| \leq k|V|/(k+1)$ for $k \geq 1$. Since $|E_{opt}| \geq k|V|/2$, the minimum-size k-ECSS heuristic achieves an approximation guarantee of $1 + [2/(k+1)]$ for $k \geq 1$.

The next lemma turns out to be quite useful. A straightforward counting argument gives the proof; see Mader [Ma 71, Lemma 1] or Cai [Ca 93, Claim 3].

LEMMA 4.2. *Let $G = (V, M)$ be a simple graph of minimum degree $k \geq 1$.*

(i) *Then for every node set $S \subseteq V$ with $1 \leq |S| \leq k$, the number of edges with exactly one end node in S, $|\delta(S)|$, is at least k.*

(ii) *If a node set $S \subseteq V$ with $1 \leq |S| \leq k$ contains at least one node of degree $\geq (k+1)$, then $|\delta(S)|$ is at least $k+1$.*

The goal of Theorem 4.3 below is to give an upper bound on the number of critical edges in the edge-complement of a spanning subgraph of minimum degree k in an arbitrary k-edge connected graph H. Clearly, every critical edge $e \in E(H)$ is in some k-cut $\delta(A_e)$, $A_e \subseteq V(H)$. By a *tight* node set S of a k-edge connected graph H we mean a set $S \subset V(H)$ with $|\delta_H(S)| = k$, i.e., a node set S such that $\delta_H(S)$ is a k-cut. As usual, a family of sets $\{S_i\}$ is called *laminar* if for any two sets in the family, either the two sets are disjoint, or one set is contained in the other. For an arbitrary subset F' of the critical edges of H, it is well known that there exists a laminar family $\mathcal{F}$ of tight node sets *covering* F', i.e., there exists $\mathcal{F} = \{A_1, A_2, \ldots, A_\ell\}$, where $A_i \subseteq V(H)$ and $\delta(A_i)$ is a k-cut, for $1 \leq i \leq \ell$, such that each edge $e \in F'$ is in some $\delta(A_i)$, $1 \leq i \leq \ell$. (For details, see [Fr 93, section 5] or [Ca 93, Lemma 3].) It is convenient to define a tree T corresponding to $\mathcal{F} \cup \{V(H)\}$: there is a T-node corresponding to each set $A_i \in \mathcal{F}$ and to $V(H)$, and there is a T-edge $A_i A_j$ (or $V(H)A_j$) iff $A_j \subset A_i$ and no other node set in $\mathcal{F}$ contains A_j and is contained in A_i. Note that the T-node corresponding to the node set A_i of the laminar family $\mathcal{F}$ is denoted by A_i, and the T-node corresponding to the node set $V(H)$ is denoted by $V(H)$. Each T-edge corresponds to a k-cut of H. Suppose that the tree T is rooted at the T-node $V(H)$. We associate another node set $\phi_i \subseteq V(H)$ with each node set

A_i of $\mathcal{F}$:

$$\phi_i = A_i \backslash \bigcup \{A \in \mathcal{F} : A \subset A_i, A \neq A_i\}.$$

In other words, a T-node $A_i \in \mathcal{F}$ that is a leaf node of T has $\phi_i = A_i$; otherwise, ϕ_i consists of those H-nodes of A_i that are not in the node sets $A', A'', \ldots$, where $A', A'', \ldots \in \mathcal{F}$ correspond to the children of A_i in the tree T. For distinct T-nodes A_i and A_j, note that ϕ_i and ϕ_j are disjoint. See Figure 5 for an illustration of $\mathcal{F} = \{A_i\}$, the family of node sets $\{\phi_i\}$, and the tree T for a particular graph.

The proof of Theorem 4.3 is long and nontrivial. Readers interested in a detailed study of the proof may be helped by an examination of the examples in Figure 2(c) and Figure 5, the illustration of the proof in Figure 6(a)–(d), and a study of the proof of Theorem 4.8, which is an analogous but weaker result for k-edge connected digraphs.

THEOREM 4.3. *Let $H = (V, E)$ be a k-edge connected, n-node graph ($k \geq 1$). Let $M \subseteq E$ be an edge set such that the spanning subgraph (V, M) has minimum degree $\geq k$. Let F be the set consisting of edges of $E \backslash M$ that are in some k-cut of H. Let $\mathcal{F} = \{A_1, \ldots, A_\ell\}$ be a laminar family of tight node sets that covers F, i.e., for each $e \in F$, there is an $A_i \in \mathcal{F}$ such that $e \in \delta(A_i)$. Then*

$$|F| \leq \frac{k}{k+1} \left| \bigcup_{i=1}^{\ell} A_i \right| \leq \frac{k}{k+1}(n-1). \tag{4.1}$$

Some key preliminaries are discussed before delving into the proof. The upper bound on $|F|$ is asymptotically tight. Consider the k-edge connected graph G obtained as follows: take $\ell + 1$ copies of the $(k+1)$-clique, $C_0, C_1, \ldots, C_\ell$, and for each $i = 1, \ldots, \ell$, choose an arbitrary node v_i in C_i and add k (nonparallel) edges between v_i and C_0. Take $M = \bigcup_{i=0}^{\ell} E(C_i)$ and $F = E(G) \backslash M$. Observe that $|F| = k(n - (k+1))/(k+1)$.

FACT 4.4. *For a laminar family of tight node sets $\mathcal{F} = \{A_1, \ldots, A_\ell\}$, $\cup_{i=1}^{\ell} \delta(A_i) = \cup_{i=1}^{\ell} \delta(\phi_i)$.*

Proof. For each $i = 1, \ldots, \ell$, an edge in $\delta(\phi_i)$ is either in $\delta(A_i)$ or in $\delta(A'), \delta(A''), \ldots$, where $A', A'', \ldots \in \mathcal{F}$ correspond to the children of A_i in the tree T. Hence, the set on the left side contains the set on the right side.

To see that the set on the left side is contained in the set on the right side, note that for every edge e in the left side set, there is an (inclusionwise) minimal tight node set $A_{i(e)}$ such that $e \in \delta(A_{i(e)})$, and the associated node set $\phi_{i(e)}$ has $e \in \delta(\phi_{i(e)})$. ∎

FACT 4.5. *Let H, M, F and $\mathcal{F} = \{A_1, \ldots, A_\ell\}$ be as in Theorem 4.3. The inequality in the theorem*

$$|F| \leq \frac{k}{k+1} \left| \bigcup_{i=1}^{\ell} A_i \right|$$

is implied by the inequality

$$\left| \bigcup_{i=1}^{\ell} \delta(A_i) \right| \leq \frac{k}{k+1} \sum_{i=1}^{\ell} |\phi_i| + \frac{1}{2} \sum_{i=1}^{\ell} |M \cap \delta(\phi_i)|.$$

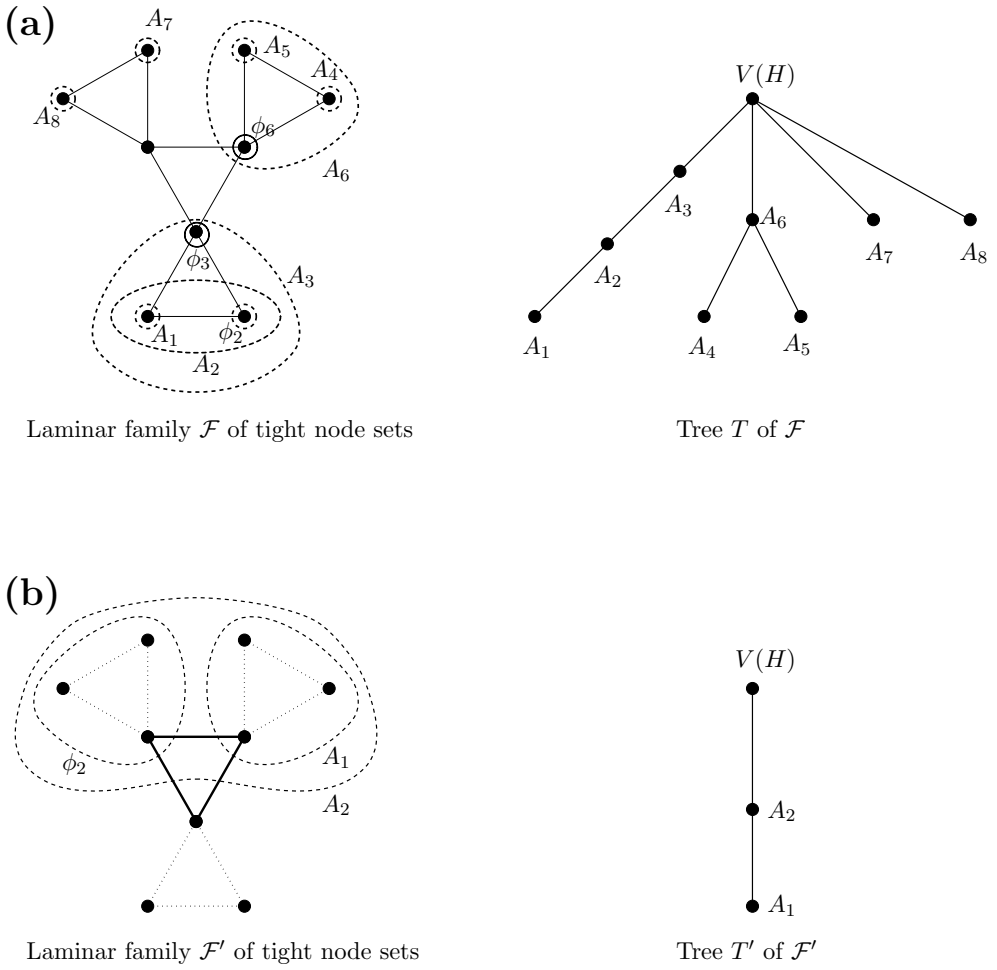


FIG. 5. *Two laminar families of tight node sets for a 2-edge connected graph H ($k = 2$).* (a) *The laminar family $\mathcal{F}$ covers all critical edges of H. $\mathcal{F}$ consists of the node sets $A_1, \ldots, A_8$, where each A_i is tight since $|\delta(A_i)| = 2 = k$. For a node set A_i, ϕ_i is the node set $A_i \setminus \cup \{A_j \in \mathcal{F} : A_j \subset A_i,\ A_j \neq A_i\}$. Note that $\phi_i = A_i$ for the inclusionwise minimal A_i, i.e., for $i = 1, 4, 5, 7, 8$. Also, the tree T corresponding to $\mathcal{F} \cup \{V(H)\}$ is illustrated.* (b) *The laminar family $\mathcal{F}'$ covers all critical edges of $E(H) \setminus M$, where $M \subset E(H)$ is such that every node is incident to at least $k = 2$ edges of M. M is indicated by dotted lines. All edges of $E(H) \setminus M$ are critical. $\mathcal{F}'$ consists of the tight node sets A_1, A_2. Also, the node sets ϕ_1, ϕ_2 are indicated ($\phi_1 = A_1$), and the tree T' representing $\mathcal{F}' \cup \{V(H)\}$ is illustrated.*

Proof. Let $M_c \subseteq M$ denote the set of M-edges that are covered by the laminar family $\mathcal{F}$, i.e.,

$$M_c = \bigcup_{i=1}^{\ell} [M \cap \delta(A_i)] = M \cap \left[\bigcup_{i=1}^{\ell} \delta(A_i)\right] = M \cap \left[\bigcup_{i=1}^{\ell} \delta(\phi_i)\right] = \bigcup_{i=1}^{\ell} [M \cap \delta(\phi_i)]\,.$$

Consider an arbitrary edge $e = vw$ that is in M_c. If $e \in \delta(\phi_i)$ $(i = 1, \ldots, \ell)$, then either $v \in \phi_i, w \notin \phi_i$ or $w \in \phi_i, v \notin \phi_i$. Since the node sets ϕ_i $(i = 1, \ldots, \ell)$ are mutually disjoint, there are at most two tight node sets $A_i \in \mathcal{F}$ such that $e \in \delta(\phi_i)$.

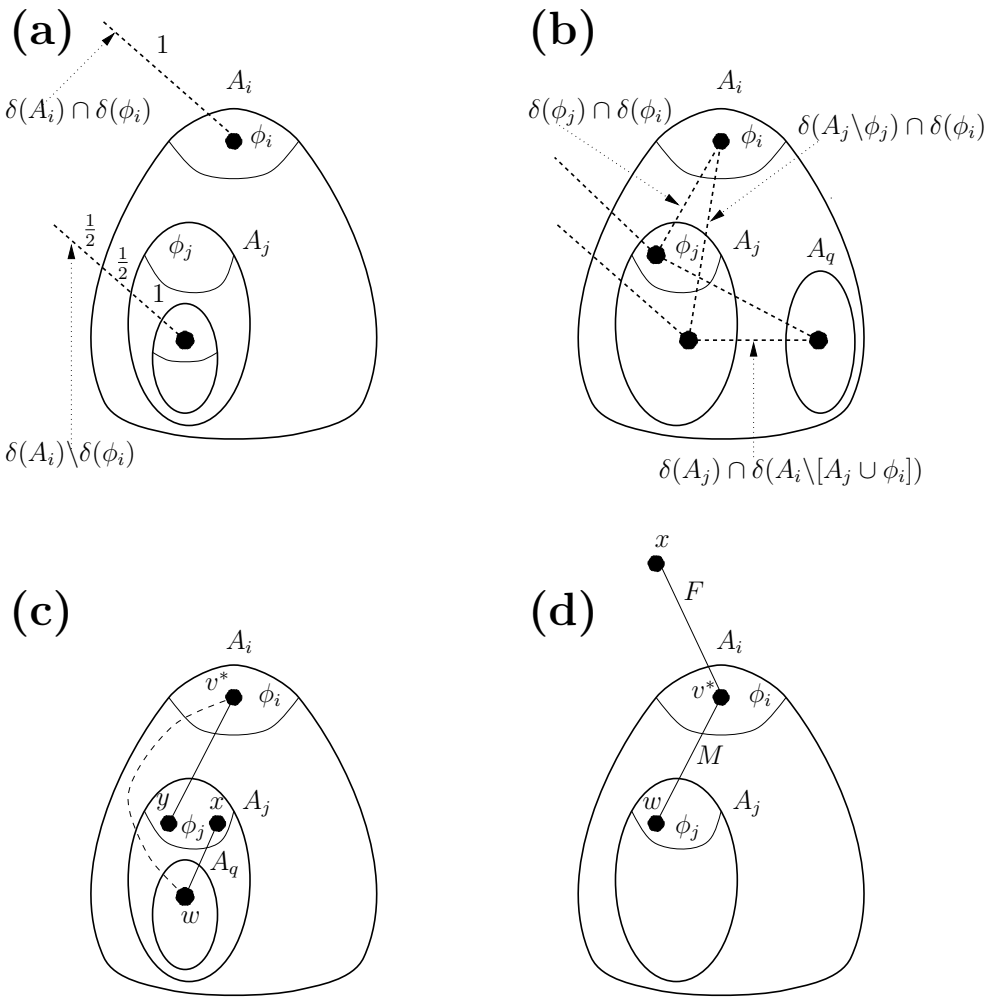


FIG. 6. *An illustration of the proof of Theorem* 4.3. (a) *Every edge in* $\delta(A_i) \cap \delta(\phi_i)$ *contributes* ≥ 1 *to the left-hand side of inequality* (σ), *and every edge in* $\delta(A_i) \backslash \delta(\phi_i)$ *contributes* $\geq \frac{1}{2}$. (b) *The tight node set* A_i *is shown, together with two tight node sets* A_j, A_q *contained in* A_i. *The node sets* ϕ_i *and* ϕ_j *are also shown. The three kinds of edges arising in the proof are illustrated.* (c) *In Claim* 3, $\phi_i = \{v^*\}$ *and* $|\phi_j| \geq (k+1)$. *An edge* wv^* *with* $w \in A_j \backslash \phi_j$ *is replaced by a pair of new edges* wx *and* yv^*, *where* $x \in \phi_j$, $y \in \phi_j$. (d) *In Claim* 3, $\phi_i = \{v^*\}$. *If an edge* v^*x *with* $x \notin A_i$ *is in* F *(so* $v^*x \notin M$*), then there is an edge* v^*w *in* M *with* $w \in \phi_j$, *where* $\phi_j \subseteq A_j \subset A_i$ *and* $|\phi_j| \geq (k+1)$. *Edges* v^*x *and* v^*w *are swapped between* M *and* F.

(For example, if there are tight node sets $A_g, A_h \in \mathcal{F}$, $g \neq h$, with $v \in \phi_g, w \in \phi_h$, then $e \in \delta(\phi_g), e \in \delta(\phi_h)$, and $e \notin \delta(\phi_i)$ for $i = 1, \ldots, \ell$, $i \neq g$, $i \neq h$.) Then

$$|M_c| = \left| \bigcup_{i=1}^{\ell} [M \cap \delta(\phi_i)] \right| \geq \frac{1}{2} \sum_{i=1}^{\ell} |M \cap \delta(\phi_i)|, \tag{4.2}$$

since we are counting the cardinality of a union of sets such that each element occurs in at most two of these sets.

Now note that $\cup_{i=1}^{\ell} \delta(A_i) = F \cup M_c$, hence

$$\left| \bigcup_{i=1}^{\ell} \delta(A_i) \right| = |F| + |M_c|. \tag{4.3}$$

Also, $\cup_{i=1}^{\ell} A_i = \cup_{i=1}^{\ell} \phi_i$, hence

$$\frac{k}{k+1} \left| \bigcup_{i=1}^{\ell} A_i \right| = \frac{k}{k+1} \left| \bigcup_{i=1}^{\ell} \phi_i \right| = \frac{k}{k+1} \sum_{i=1}^{\ell} |\phi_i|. \tag{4.4}$$

Substituting inequalities (4.2), (4.3), and (4.4) into the second inequality in the fact gives

$$|F| + |M_c| \leq \frac{k}{k+1} \left| \bigcup_{i=1}^{\ell} A_i \right| + |M_c|,$$

which is the inequality in Theorem 4.3. □

Most of the complications in the proof of Theorem 4.3 seem to be caused by the presence of tight node sets $A_i \in \mathcal{F}$ such that $|\phi_i| = 1$. To illustrate the main ideas in the proof, we first prove a weaker version of Theorem 4.3. In the weaker version, the required upper bound of $k(n-1)/(k+1)$ is relaxed to $(n-1)$, and the laminar family of tight node sets $\mathcal{F} = \{A_1, \ldots, A_\ell\}$ is restricted such that every $A_i \in \mathcal{F}$ has $|\phi_i| \geq 2$. (The motivation for putting the restriction on $\mathcal{F}$ is expository. Such restricted laminar families $\mathcal{F}$ do not seem to be of mathematical interest.)

PROPOSITION 4.6. *Let H, M, F, and $\mathcal{F}$ be as in Theorem 4.3, and moreover, suppose that each tight node set $A_i \in \mathcal{F}$ has $|\phi_i| \geq 2$. Then*

$$|F| \leq \left| \bigcup_{i=1}^{\ell} A_i \right| \leq n - 1.$$

Proof. For an arbitrary $i = 1, \ldots, \ell$, consider A_i, ϕ_i, and let p denote $|\phi_i|$. By assumption, $p \geq 2$. Suppose that $p \leq k$ (the other case $p \geq k+1$ turns out to be easy). Then

$$|M \cap \delta(\phi_i)| \geq p(k - (p-1)), \tag{4.5}$$

since for every node $v \in \phi_i$, there are at most $(p-1)$ incident edges $vw \in E(H)$ with $w \in \phi_i$. Adding $2|\phi_i|$ to both sides of inequality (4.5) gives

$$2|\phi_i| + |M \cap \delta(\phi_i)| \geq 2p + p(k - (p-1)) \geq -p^2 + (k+2)p. \tag{4.6}$$

Subtracting $2k$ from both sides of inequality (4.6) gives

$$2|\phi_i| + |M \cap \delta(\phi_i)| - 2k \geq -p^2 + (k+2)p - 2k = -(p-k)(p-2) \geq 0, \tag{4.7}$$

where the last inequality $-(p-k)(p-2) \geq 0$ holds because $2 \leq p \leq k$. Inequality (4.7) implies

$$|\phi_i| + \frac{1}{2}|M \cap \delta(\phi_i)| \geq k = |\delta(A_i)|. \tag{4.8}$$

If $|\phi_i| \geq (k+1)$, then obviously inequality (4.8) holds.

Summing up inequality (4.8) over $i = 1, \ldots, \ell$ gives

$$\left|\bigcup_{i=1}^{\ell} \delta(A_i)\right| \leq \sum_{i=1}^{\ell} |\delta(A_i)| = k \cdot \ell \leq \sum_{i=1}^{\ell} |\phi_i| + \frac{1}{2} \sum_{i=1}^{\ell} |M \cap \delta(\phi_i)|. \tag{4.9}$$

The proof of Fact 4.5 shows that inequality (4.9) implies the inequality in the proposition,

$$|F| \leq \left|\bigcup_{i=1}^{\ell} A_i\right| \leq n - 1. \qquad \square$$

Proof of Theorem 4.3. Without loss of generality (w.l.o.g.) assume that $\mathcal{F}$ is minimal, i.e., for every $A_i \in \mathcal{F}$ there is an edge $e_i \in F$ such that $e_i \in \delta(A_i)$ and $e_i \notin \delta(A)$ $\forall A \in \mathcal{F}$, $A \neq A_i$. Since $\mathcal{F}$ is minimal, every $A_i \in \mathcal{F}$ has $|\phi_i| \geq 1$. Let T be the tree representing $\mathcal{F} \cup \{V(H)\}$. The proof examines the node sets $A_i \in \mathcal{F}$, ϕ_i, but the node set $V(H) \setminus \bigcup\{A_i : A_i \in \mathcal{F}\}$ is not relevant for the proof. Every inclusionwise minimal $A_i \in \mathcal{F}$ has $|A_i| \geq (k+1)$, since $\delta(A_i) \cap F \neq \emptyset$ implies that A_i contains a node v with $\deg_H(v) \geq (k+1)$, so Lemma 4.2 implies this bound on $|A_i|$. Hence, every $A_i \in \mathcal{F}$ with $|\phi_i| = 1$ has at least one child in the tree T.

Two key assumptions are needed to complete the proof.

Assumption 1. For $1 \leq i \leq \ell$, every ϕ_i induces a complete subgraph of H, and moreover, every edge of this complete subgraph is in M, i.e., for $i = 1, \ldots, \ell$ $\forall v, w \in \phi_i, vw \in E(H)$ and $vw \in M$.

Assumption 2. For every $A_i \in \mathcal{F}$ with $|\phi_i| = 1$, there is an $A_j \in \mathcal{F}$ such that $|\phi_j| \leq k$ and A_j is a child of A_i in the tree T.

Claim 2. Assumption 1 causes no loss of generality.

Here is the proof of Claim 2. For an arbitrary $i = 1, \ldots, \ell$, consider ϕ_i and $E(\phi_i)$, the set of edges of H with both end nodes in ϕ_i. Clearly, an edge $vw \in E(\phi_i)$ is *not* in F, since vw is in none of the k-cuts $\delta(A_j)$ $(j = 1, \ldots, \ell)$. Therefore, all the missing edges vw with $v \in \phi_i, w \in \phi_i$ can be added to H (say, vw is first added to $E \setminus (M \cup F)$) such that ϕ_i induces a clique and this will keep M, F, and $\mathcal{F}$ unchanged. Moreover, every edge $vw \in E(\phi_i)$ can be placed in M, and the minimum degree requirement on (V, M) will continue to hold. By repeating this for each $i = 1, \ldots, \ell$, we obtain $H', M', F' = F$ and $\mathcal{F}' = \mathcal{F}$ that satisfy Assumption 1 and the conditions in the theorem. Clearly, if the inequality in the theorem holds for $H', M', F', \mathcal{F}'$, then it also holds for $H, M, F, \mathcal{F}$.

Claim 3. Assumption 2 causes no loss of generality.

Proof of Claim 3. Consider an $A_i \in \mathcal{F}$ $(i = 1, \ldots, \ell)$ such that $|\phi_i| = 1$ and in the tree T every child $A_j \in \mathcal{F}$ of A_i has $|\phi_j| \geq (k+1)$. Let $\phi_i = \{v^*\}$. Let $A_j \in \mathcal{F}$ be an arbitrary T-child of A_i with $|\phi_j| \geq (k+1)$. Clearly, by Assumption 1, the subgraph of H induced by ϕ_j is a clique, and every edge in the clique is in M. Suppose that H has an edge wv^* such that $w \in A_j \setminus \phi_j$, i.e., $wv^* \in \delta(A_j) \setminus \delta(\phi_j)$. (Figure 6(c) illustrates this.) Then we replace wv^* by a pair of new edges wx, yv^* with $x \in \phi_j, y \in \phi_j$ (possibly, $x = y$) such that the resulting graph H' is simple (i.e., H' has no multiedges); this can be done always, since $|\phi_j| \geq (k+1)$ and both $\delta(A_j)$ and $\delta(A_q)$ are k-cuts, where $A_q \in \mathcal{F}$ is the T-child of A_j containing node w. The resulting graph H' is k-edge connected. To see this, note that H is k-edge connected, and H' is obtained from H by replacing one edge wv^* by two edges wx, yv^*, where the nodes x and y are contained in the $(k+1)$-clique induced by ϕ_j. The formal proof of the k-edge connectivity of H' is easy and is left to the reader. If $wv^* \in M$,

then we take $M' = (M \backslash \{wv^*\}) \cup \{wx, yv^*\}$, $F' = F$; otherwise, we take $M' = M$, $F' = (F \backslash \{wv^*\}) \cup \{wx, yv^*\}$. In either case $\mathcal{F}$ covers F'. By repeating this maneuver for all relevant $i = 1, \ldots, \ell$, we obtain H', M', F', and $\mathcal{F}' = \mathcal{F}$ with $|F'| \geq |F|$ that satisfy the conditions in the theorem. Clearly, if the inequality in the theorem holds for $H', M', F', \mathcal{F}'$, then it also holds for $H, M, F, \mathcal{F}$. Moreover, the following condition $(*)$ holds:

$(*)$ For every $A_i \in \mathcal{F}$ with $|\phi_i| = 1$, for every T-child $A_j \in \mathcal{F}$ of A_i with $|\phi_j| \geq (k+1)$, every edge in $\delta(A_j) \cap \delta(\phi_i)$ is in $\delta(\phi_j)$.

Now w.l.o.g. suppose that H, M, F, and $\mathcal{F}$ satisfy condition $(*)$. Call an $A_i \in \mathcal{F}$ *bad* if $|\phi_i| = 1$ and every T-child $A_j \in \mathcal{F}$ of A_i has $|\phi_j| \geq (k+1)$. Suppose that there is a bad $A_i \in \mathcal{F}$ with $\phi_i = \{v^*\}$ such that one of the edges $v^*x \in \delta(A_i) \cap \delta(\phi_i)$ is not in M. (Figure 6(d) illustrates this.) Then since $|\delta(A_i)| = k$, $|\delta(A_i) \cap F| \geq 1$, and $|M \cap \delta(\phi_i)| \geq k$, there must be an M-edge wv^* in $\delta(\phi_i) \backslash \delta(A_i)$. Let $A_j \in \mathcal{F}$ be the T-child of A_i such that $w \in A_j$. Since A_i is bad, $|\phi_j| \geq (k+1)$, therefore condition $(*)$ applies and ensures that the node w is in ϕ_j. Moreover, by Assumption 1, w is incident to $\geq k$ edges of M that have both end nodes in ϕ_j. Take $M' = (M \backslash \{wv^*\}) \cup \{v^*x\}$, $F' = (F \backslash \{v^*x\}) \cup \{wv^*\}$, and observe that $|M| = |M'|$, $|F| = |F'|$, every node $v \in V(H)$ is incident to $\geq k$ edges of M', F' consists of critical edges in $E(H) \backslash M'$, and $\mathcal{F}$ covers F'. By repeating this maneuver for all relevant $i = 1, \ldots, \ell$, we obtain H, M', F' and $\mathcal{F}$ that satisfy the conditions in the theorem such that $|F'| = |F|$, and for every bad $A_i \in \mathcal{F}$, no edge in $\delta(A_i) \cap \delta(\phi_i)$ is in F'. Then we can start with $\mathcal{F}$ and remove each bad A_i from $\mathcal{F}$ to obtain another laminar family $\mathcal{F}'$ covering F' such that $|\cup_{A \in \mathcal{F}'} A| \leq |\cup_{A \in \mathcal{F}} A|$, and $\mathcal{F}'$ satisfies Assumption 2. Clearly, if the inequality in the theorem holds for $H', M', F', \mathcal{F}'$, then it also holds for $H, M, F, \mathcal{F}$. This completes the proof of Claim 3.

Instead of proving that $F, \mathcal{F}$ satisfy inequality (4.1), we prove that under Assumption 2, M, F, and $\mathcal{F} = \{A_1, \ldots, A_\ell\}$ satisfy the following sharper inequality (see Fact 4.5):

$$\left| \bigcup_{i=1}^{\ell} \delta(A_i) \right| \leq \frac{k}{k+1} \sum_{i=1}^{\ell} |\phi_i| + \frac{1}{2} \sum_{i=1}^{\ell} |M \cap \delta(\phi_i)|. \tag{4.10}$$

Clearly, every $A_i \in \mathcal{F}$ with $|\phi_i| \geq (k+1)$ satisfies the inequality

$$|\delta(A_i)| \leq \frac{k}{k+1} |\phi_i|. \tag{4.11}$$

From the proof of Proposition 4.6 (see inequalities (4.5), (4.6), (4.7), (4.8)), it follows that every $A_i \in \mathcal{F}$ with $2 \leq |\phi_i| \leq k$ satisfies the inequality

$$|\delta(A_i)| + \frac{k-1}{2(k+1)} |\phi_i| \leq \frac{k}{k+1} |\phi_i| + \frac{1}{2} |M \cap \delta(\phi_i)|, \tag{4.12}$$

where the surplus term on the left-hand side (l.h.s.) is the difference between $k|\phi_i|/(k+1)$ and $|\phi_i|/2$. Every $A_i \in \mathcal{F}$ with $|\phi_i| = 1$ satisfies the inequality

$$\begin{aligned} &|\delta(A_i) \cap \delta(\phi_i)| + \frac{1}{2} |\delta(A_i) \backslash \delta(\phi_i)| + \frac{k}{k+1} - \frac{1}{2} |\delta(A_i) \cap \delta(\phi_i)| \\ &\leq \frac{k}{k+1} |\phi_i| + \frac{1}{2} |M \cap \delta(\phi_i)|, \end{aligned} \tag{4.13}$$

because $|\delta(A_i) \cap \delta(\phi_i)| + |\delta(A_i) \backslash \delta(\phi_i)| = |\delta(A_i)| = k \leq |M \cap \delta(\phi_i)|$.

Claim 4. Under Assumption 2, the inequality (σ) obtained by summing up over all $A_i \in \mathcal{F}$ the appropriate one of inequalities (4.11), (4.12), (4.13) implies inequality (4.10), i.e., the l.h.s. of inequality (σ) is $\geq$ the l.h.s. of inequality (4.10), and the right-hand side (r.h.s.) of inequality (σ) is $\leq$ the r.h.s. of inequality (4.10).

Proof of Claim 4. Clearly, inequality (σ) will imply inequality (4.10) if for every $A_i \in \mathcal{F}$, every edge in $\delta(A_i) \cap \delta(\phi_i)$ contributes ≥ 1 to the l.h.s. of inequality (σ). This property holds for $A_i \in \mathcal{F}$ with $|\phi_i| \geq 2$ by inequalities (4.11), (4.12), but for $A_i \in \mathcal{F}$ with $|\phi_i| = 1$ the property fails to hold (see inequality (4.13)). Fortunately, there is a way around this difficulty. For $A_i \in \mathcal{F}$ with $|\phi_i| = 1$, we allow A_i, ϕ_i to contribute a deficit of $\frac{1}{2}|\delta(A_i) \cap \delta(\phi_i)|$ on the l.h.s. of inequality (σ); using this deficit, we can ensure that every edge in $\delta(A_i) \cap \delta(\phi_i)$ (in $\delta(A_i) \backslash \delta(\phi_i)$) contributes ≥ 1 ($\geq 1/2$) to the l.h.s. of inequality (σ); see inequality (4.13). (Figure 6(a) illustrates the general scheme.) For each $A_i \in \mathcal{F}$ with $|\phi_i| = 1$, let $A_{c(i)} \in \mathcal{F}$ be an arbitrary T-child of A_i such that $1 \leq |\phi_{c(i)}| \leq k$; $A_{c(i)}$ exists by Assumption 2. Inequality (σ) implies inequality (4.10) because the deficit contributed by each $A_i \in \mathcal{F}$ with $|\phi_i| = 1$ is compensated by the surplus contributed by $A_{c(i)}, \phi_{c(i)}$. To see this, focus on an arbitrary $A_i \in \mathcal{F}$ with $|\phi_i| = 1$ and let $j = c(i)$. First observe that if an edge $vw \in \delta(A_j)$ with $v \in A_j$ is not in $\delta(A_i)$, then there are three possibilities: (i) $v \in \phi_j,\ w \in \phi_i$, (ii) $v \notin \phi_j,\ w \in \phi_i$, i.e., $v \in A_g$, where $A_g \in \mathcal{F}$ corresponds to a child of A_j in the tree T, and (iii) $v \in A_j,\ w \in A_i \backslash [A_j \cup \phi_i]$, i.e., $w \in A_q$, where $A_q \in \mathcal{F}$ corresponds to a sibling of A_j in the tree T. (Figure 6(b) illustrates the three possibilities.) Second, observe that

$$\begin{aligned}|\delta(A_i) \cap \delta(\phi_i)| &\leq |\delta(A_j) \backslash \delta(A_i)| \\ &= |\delta(A_j \backslash \phi_j) \cap \delta(\phi_i)| + |\delta(A_j) \cap \delta(A_i \backslash [A_j \cup \phi_i])| + |\delta(\phi_j) \cap \delta(\phi_i)|.\end{aligned}$$

For each of the first two terms t on the r.h.s., A_j, ϕ_j contributes a surplus of at least $t/2$ to the l.h.s. of inequality (σ), because (i) every edge in two distinct k-cuts $\delta(A_g)$ and $\delta(A_j)$, $A_g \in \mathcal{F}$, $A_j \in \mathcal{F}$, $A_g \subset A_j$, contributes a surplus of $1/2$ or more, since $A_h \in \mathcal{F}$ such that $\delta(\phi_h) \cap \delta(A_h)$ contains the edge contributes one for the edge, and every other $A \in \mathcal{F}$ such that $\delta(A)$ contains the edge contributes $\geq 1/2$ for the edge; (ii) every edge in two distinct k-cuts $\delta(A_q)$ and $\delta(A_j)$, $A_q \in \mathcal{F}$ disjoint from $A_j \in \mathcal{F}$, contributes a surplus of one or more.

Focus on the third term $|\delta(\phi_j) \cap \delta(\phi_i)|$, and note that its value is $\leq |\phi_j|$, since $|\phi_i| = 1$ and the graph is simple. If $|\phi_j| = 1$, then the deficit contributed by A_i, ϕ_i (to the l.h.s. of inequality (σ)) is compensated, because the surplus of $\frac{k}{k+1}$ (on the l.h.s. of A_i's inequality) is $\geq \frac{1}{2}$ (for $k \geq 1$), hence

$$\frac{1}{2}|\delta(A_i) \cap \delta(\phi_i)| \quad \leq \quad \frac{1}{2}|\delta(A_j \backslash \phi_j) \cap \delta(\phi_i)| + \frac{1}{2}|\delta(A_j) \cap \delta(A_i \backslash [A_j \cup \phi_i])| + \frac{k}{k+1}.$$

If $2 \leq |\phi_j| \leq k$, then the deficit contributed by A_i, ϕ_i (to the l.h.s. of inequality (σ)) is compensated, because the surplus of $\frac{k-1}{2(k+1)}|\phi_j| + \frac{k}{k+1}$ (on the l.h.s. of A_j's and A_i's inequalities) is $\geq |\phi_j|/2$ (for $k \geq |\phi_j| \geq 1$), hence

$$\begin{aligned}\frac{1}{2}|\delta(A_i) \cap \delta(\phi_i)| \leq{}& \frac{1}{2}|\delta(A_j \backslash \phi_j) \cap \delta(\phi_i)| \\ &+ \frac{1}{2}|\delta(A_j) \cap \delta(A_i \backslash [A_j \cup \phi_i])| + \frac{k-1}{2(k+1)}|\phi_j| + \frac{k}{k+1}.\end{aligned}$$

This completes the proof of Claim 4 and the proof of the theorem. □

THEOREM 4.7. *Let $G = (V, E)$ be a graph of edge connectivity $\geq k \geq 1$. The heuristic described above finds a k-edge connected spanning subgraph (V, E') such that*

$|E'| \leq (1 + [2/(k+1)])|E_{opt}|$, *where* $|E_{opt}|$ *denotes the cardinality of an optimal solution. The running time is* $O(k^3|V|^2 + |E|^{1.5}(\log |V|)^2)$.

4.2. Directed graphs. The heuristic for finding an approximately minimum-size k-edge connected spanning subgraph of a digraph has two steps. Similarly to section 3.4, the first step finds a minimum-cardinality arc set $M \subseteq E$ such that for every node v, there are $\geq k$ arcs of M going out of v and $\geq k$ arcs of M coming into v. Clearly, $|M| \leq |E_{opt}|$, where $E_{opt} \subseteq E$ denotes a minimum-cardinality arc set such that (V, E_{opt}) is k-edge connected. The second step of the heuristic finds an (inclusionwise) minimal arc set $F \subseteq E \backslash M$ such that $E' = M \cup F$ is the arc set of a k-edge connected spanning subgraph. To prove the approximation guarantee, we need to estimate $|F|$. We use the notion of special arcs to estimate $|F|$. Call an arc (v, w) of a k-edge connected digraph *special* if the arc is critical, and in addition, if $\deg_{out}(v) \geq (k+1)$ and $\deg_{in}(w) \geq (k+1)$. Clearly, every arc in F is a special arc of the digraph $G' = (V, E')$, $E' = M \cup F$, returned by the heuristic. We can deduce a bound of $O(\sqrt{k}|V|)$ on the number of special arcs in G' by examining chains of tight node sets $S_1 \subset S_2 \subset \cdots \subset S_q$, where a node set S_i is called tight if G' has exactly k arcs in $\delta_{out}(S_i)$.

THEOREM 4.8. *Let* $k \geq 1$ *be an integer, and let* H *be a* k*-edge connected,* n*-node digraph. The number of special arcs in* H *is at most* $4\sqrt{k} \cdot n$.

Proof. Let V denote $V(H)$ for this proof. Each special arc e is in a k-dicut $\delta_{out}(A_e) = \delta_{in}(V \backslash A_e)$, where $2 \leq |A_e| \leq n-2$. As in section 4.1, we obtain *two* laminar families of tight node sets $\mathcal{F}_{out}$ and $\mathcal{F}_{in}$ that cover all the special arcs: that is, for each $A_i \in \mathcal{F}_{out}$ ($A_i \in \mathcal{F}_{in}$), A_i is a set of H-nodes, $\delta_{out}(A_i)$ ($\delta_{in}(A_i)$) has k arcs including at least one special arc, and each special arc is in some $\delta_{out}(A_i)$, $A_i \in \mathcal{F}_{out}$, or is in some $\delta_{in}(A_i)$, $A_i \in \mathcal{F}_{in}$. Focus on $\mathcal{F}_{out}$; the analysis is symmetric for $\mathcal{F}_{in}$. Let $\mathcal{F}_{out} = \{A_1, A_2, \ldots, A_\ell\}$. To estimate the number of special arcs, we need to examine the tree T corresponding to $\mathcal{F}_{out} \cup \{V(H)\}$. For $i = 1, \ldots, \ell$, recall that the T-node corresponding to a node set $A_i \in \mathcal{F}_{out}$ is also denoted A_i (the T-node corresponding to $V(H)$ is denoted by V), and recall that ϕ_i denotes $A_i \backslash \cup\{A \in \mathcal{F}_{out} : A \subset A_i,\ A \neq A_i\}$. Partition the set $\{A_1, \ldots, A_\ell\}$ of T-nodes into two sets R_1 and R_2, where R_2 consists of the T-nodes incident to precisely two T-edges and $R_1 = \{A_1, \ldots, A_\ell\} \backslash R_2$. Note that $V \notin R_1$ and $V \notin R_2$.

Claim 5. $|R_1| \leq 2|V_1|/(k+1)$, where V_1 denotes the set of H-nodes in $\bigcup\{\phi_i : A_i \in R_1\}$.

Here is the proof of Claim 5. Let T_1 be the tree obtained from the tree T by "unsubdividing" all the T-nodes in R_2, i.e., by repeatedly replacing a degree-2 T-node in R_2 and its two incident edges by an edge between the two neighbors. Then T_1 is a tree whose nonleaf T-nodes in R_1 have T_1-degree ≥ 3, whereas the T-node V may have T_1-degree 1, 2, or ≥ 3. Let ℓ_1 be the number of leaf nodes (degree-1 nodes) of T_1 in R_1. Then, $|R_1| \leq \ell_1 + (\ell_1 + 1) - 2 \leq 2\ell_1$. Now, Claim 5 follows because $\ell_1 \leq |V_1|/(k+1)$, because for each (inclusionwise) minimal $A_i \in \mathcal{F}_{out}$, the set $\phi_i = A_i$ of H-nodes has cardinality at least $(k+1)$ by the digraph version of Lemma 4.2(ii). (A_i contains a node v with $\deg_{out}(v) \geq (k+1)$ since $\delta_{out}(A_i)$ contains a special arc.)

Now focus on a maximal path $P = A_0, A_1, \ldots, A_{q+1}$ of T such that every T-node A_i with $1 \leq i \leq q$ is in R_2. In H, the node sets $A_0, A_1, \ldots, A_{q+1}$ satisfy $A_0 \subset A_1 \subset \cdots \subset A_{q+1}$, and for $i = 1, \ldots, q$, if $A' \in \mathcal{F}_{out}$ is contained in A_i, then either $A' = A_{i-1}$ or $A' \subset A_{i-1}$. Let V_P denote the set of H-nodes $\phi_1 \cup \phi_2 \cup \cdots \cup \phi_q$. Also, note that for $i = 1, 2, \ldots, q$, $A_i = A_0 \cup \phi_1 \cup \phi_2 \cup \cdots \cup \phi_i$.

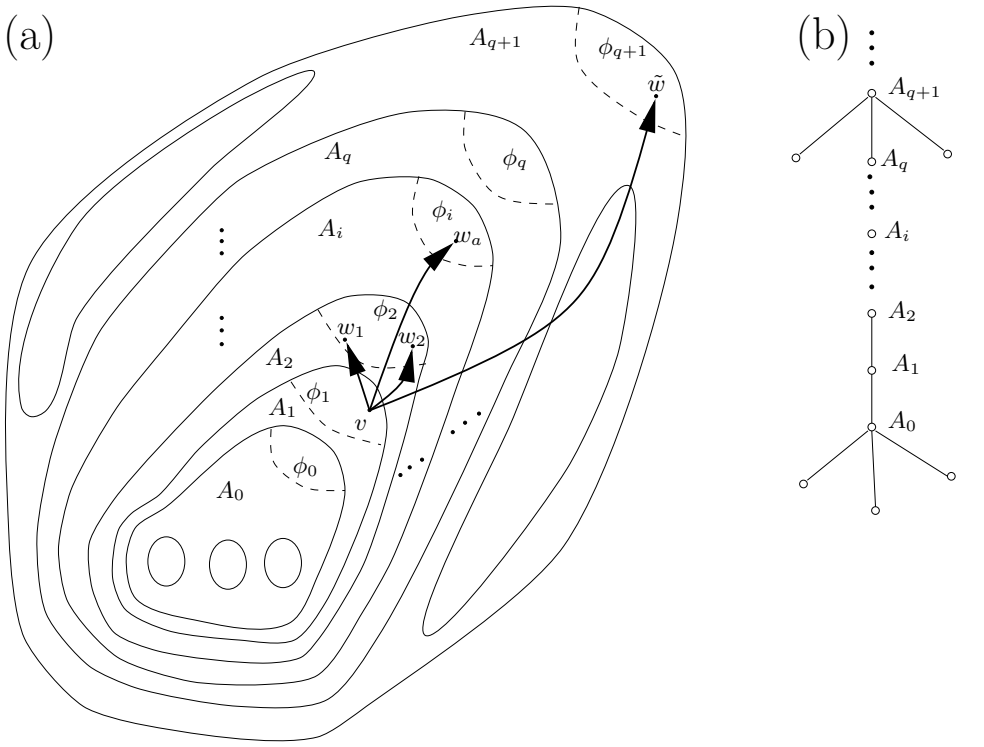


FIG. 7. *An illustration of Claim* 6 *in the proof of Theorem* 4.8. (a) *A subfamily of the laminar family of tight node sets* $\mathcal{F}_{out}$ *that covers (some of) the special arcs.* (b) *The subtree corresponding to the subfamily of* $\mathcal{F}_{out}$ *in* (a). *Each of the* T*-nodes* $A_1, A_2, \ldots, A_q$ *is incident to exactly two edges of* T, *where* T *is the tree corresponding to* $\mathcal{F}_{out}$.

Claim 6. The number of arcs (v, w) such that $v \in V_P$ and $(v, w) \in \bigcup\{\delta_{out}(A_i) : 1 \le i \le q\}$ is at most $k + 2\sqrt{k} \cdot |V_P|$.

Here is the proof of the Claim 6; see Figure 7 for an illustration. The additional term of k in the upper bound accounts for the arcs with start nodes in A_q and end nodes in $V \backslash A_q$; there are at most k such arcs, since each such arc is in $\delta_{out}(A_q)$. Now ignore the arcs in $\delta_{out}(A_q)$. Linearly order the H-nodes in V_P such that for each i, $1 \le i < q$, the H-nodes in ϕ_i precede the H-nodes in ϕ_{i+1}. Let v be an arbitrary node in V_P. Let $\Gamma_v \subseteq V_P$ denote the set of end nodes w_j of the arcs (v, w_j) outgoing from v such that $w_j \in V_P$ and $(v, w_j) \in \bigcup\{\delta_{out}(A_i) : 1 \le i \le q\}$. Let the linear ordering of the nodes in Γ_v be $w_1, w_2, \ldots, w_{|\Gamma_v|}$. Call an arc (v, w_j) *short* if $j \le \sqrt{k}$, otherwise, call the arc *long*. We "charge" each long arc (v, w_j) to the first $\sqrt{k}$ nodes $w_1, w_2, \ldots, w_{\sqrt{k}}$ in Γ_v, i.e., each of these nodes is charged $1/\sqrt{k}$ for each arc (v, w_j), $w_j \in \Gamma_v$, and $j > \sqrt{k}$. Now consider the total charge on an arbitrary node $w_a \in V_P$ due to all long arcs $(x, y) \in \bigcup\{\delta_{out}(A_i) : 1 \le i \le q\}$ with $x \in V_P$ and $y \in V_P$. The key fact is this: the total charge on w_a is at most $\sqrt{k}$. To see this suppose that $w_a \in \phi_i$, where $1 \le i \le q$. Then for every arc (v, w_j) charged to w_a, $(v, w_j) \in \delta_{out}(A_{i-1})$, because $v \in A_i \backslash \phi_i$ (if $v \in V \backslash A_i$ or $v \in \phi_i$, then clearly Γ_v does not contain a node of ϕ_i such as w_a). Furthermore, by the linear ordering of Γ_v, $w_j \in \phi_i \cup \phi_{i+1} \cup \cdots \cup \phi_q$, i.e., $w_j \notin A_{i-1}$. Since $\delta_{out}(A_{i-1})$ has k arcs, the total charge to w_a is at most $k \cdot (1/\sqrt{k}) = \sqrt{k}$. Finally, consider the total number, m_P, of short arcs $(x, y) \in \bigcup\{\delta_{out}(A_i) : 1 \le i \le q\}$ with $x \in V_P$ and $y \in V_P$. Obviously, m_P is at

most $\sqrt{k}|V_P|$. Claim 6 is completed by summing up the three terms: k (for arcs in $\delta_{out}(A_q)$), $\sqrt{k}|V_P|$ (for the total charge on nodes $w \in V_P$), and $\sqrt{k}|V_P|$ (for m_P).

We account for the special arcs in $\delta_{out}(A_q)$ by "charging" the additional term of k to the "unsubdivided edge" A_0A_{q+1} of the tree T_1 in the proof of Claim 5. Thus each edge A_iA_{i+q+1}, $A_i \subset A_{i+q+1}$, of T_1 is "charged" for $\leq 2k$ special arcs (these are the special arcs in $\delta_{out}(A_i) \cup \delta_{out}(A_{i+q})$). Since the number of edges in T_1 is $\leq |R_1|$, the number of special arcs contributed by the T-nodes in R_1 is $\leq 2k|R_1|$. We "charge" $2\sqrt{k}$ to each H-node v such that $v \in \phi_i$ for a T-node $A_i \in R_2$. Combining the contributions of special arcs from the T-nodes in R_1 and R_2 and applying Claim 5, we see that the number of special arcs is at most

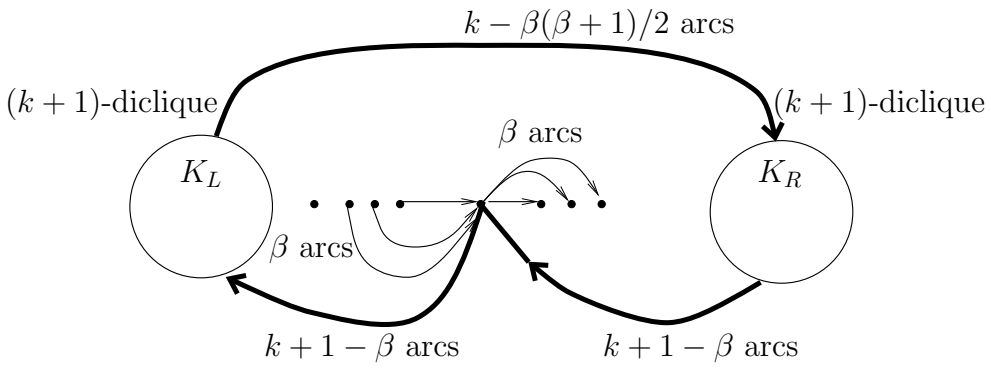
$$2k|R_1| + 2\sqrt{k} \cdot n_2 \leq \frac{4kn_1}{(k+1)} + 2\sqrt{k} \cdot n_2,$$

where n_1 and n_2 denote the cardinalities of $V_1 = \bigcup\{\phi_i : A_i \in R_1\}$ and $V_2 = \bigcup\{\phi_i : A_i \in R_2\}$, respectively. For $k \geq 1$, the number of special arcs is maximized when n_2 is the maximum possible and n_1 is the minimum possible. Since the tree T has at least two leaves, n_2 is at most $n - (2k+2)$. Hence, the number of special arcs contributed by $\mathcal{F}_{out}$ is at most $4k(2k+2)/(k+1) + 2\sqrt{k}(n-(2k+2))$. The total number of special arcs in H is at most $16k + 4\sqrt{k}(n-(2k+2)) \leq 4\sqrt{k}n$. □

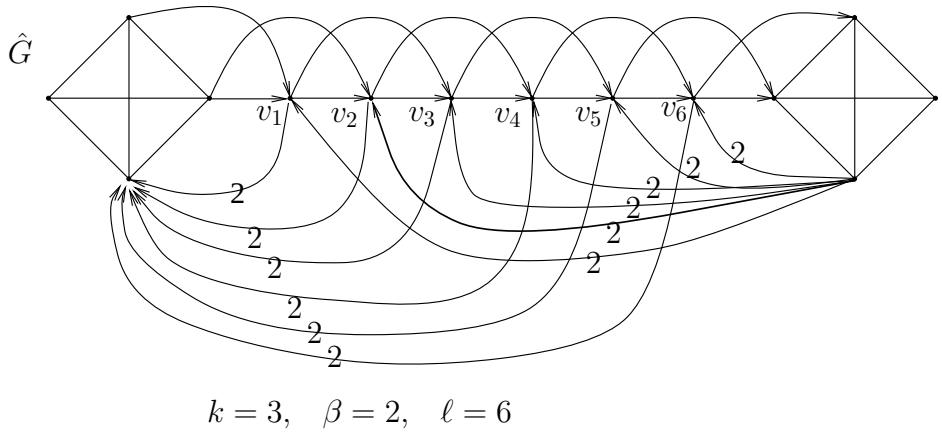
The heuristic clearly runs in time $O(k|E|^2)$. This can be improved by implementing the second step to run in time $O(k^3|V|^2)$. We run Gabow's algorithm [Ga 95] as a preprocessing step to compute a sparse certificate $\widetilde{E}$ of G for k-edge connectivity, i.e., $\widetilde{E} \subseteq E$, $|\widetilde{E}| \leq 2k|V|$, and for all nodes v, w, $(V, \widetilde{E})$ has k arc-disjoint $v \to w$ directed paths iff G has k arc-disjoint $v \to w$ directed paths. In detail, we fix a node $a \in V(G)$ and take $\widetilde{E} = \widetilde{E}_{out} \cup \widetilde{E}_{in}$, where $\widetilde{E}_{out}$ ($\widetilde{E}_{in}$) is the union of k arc-disjoint out-branchings (in-branchings) rooted at a. Gabow's algorithm [Ga 95] runs in time $O(k|V|^2)$, and the second step runs in time $O(k|\widetilde{E} \cup M|^2) = O(k^3|V|^2)$.

THEOREM 4.9. *Let $G = (V, E)$ be a digraph of edge connectivity $\geq k$. The heuristic described above finds a k-edge connected spanning subgraph (V, E') such that $|E'| \leq (1 + [4/\sqrt{k}])|E_{opt}|$, where $|E_{opt}|$ denotes the cardinality of an optimal solution. The running time is $O(k^3|V|^2 + |E|^{1.5}(\log |V|)^2)$.*

The upper bound on the number of special arcs in Theorem 4.8 is not tight but is within a factor of (roughly) 3 of the tight bound for $n \gg k$. To see this, take $n \geq 3k+2$ and consider the following k-edge connected, n-node digraph $\hat{G}$ with at least $\beta n - 2\beta(k+1) + k$ special arcs, where β is the maximum integer such that $\beta(\beta+1)/2 \leq k$, i.e., $\beta = \lfloor \sqrt{2k + 0.25} - 0.5 \rfloor$. See Figure 8 for an illustration of $\hat{G}$. $\hat{G}$ has a "left" $(k+1)$-directed clique K_L and a "right" $(k+1)$-directed clique K_R. Let $v_1, v_2, \ldots, v_\ell$ be a linear ordering of the remaining nodes, where $\ell = n - 2(k+1) \geq k$. There is one arc from v_i $(1 \leq i \leq \ell)$ to each of the next β nodes $v_{i+1}, \ldots, v_{i+\beta}$; hence, each node v_i has one arc coming in from each of the previous β nodes $v_{i-1}, \ldots, v_{i-\beta}$. (Take $v_0, v_{-1}, v_{-2}, \ldots, v_{-\beta+1}$ to mean nodes in K_L, and take $v_{\ell+1}, v_{\ell+2}, \ldots, v_{\ell+\beta}$ to mean nodes in K_R.) These β left-to-right arcs starting from v_i will turn out to be special arcs. Additionally, there are $(k+1-\beta)$ arcs from K_R to each of the nodes $v_1, v_2, \ldots, v_\ell$, and there are $(k+1-\beta)$ arcs from each of the nodes $v_1, v_2, \ldots, v_\ell$ to K_L. Finally, there are $(k - \beta(\beta+1)/2)$ arcs from K_L to K_R. This completes the construction of $\hat{G}$. It can be checked that $\hat{G}$ is k-edge connected. (Note that besides the $(k - \beta(\beta+1)/2)$ arcs from K_L to K_R, there are $\beta(\beta+1)/2$ arc-disjoint directed paths from K_L to K_R, such that there is one "one-hop" directed path, two "two-hop" directed paths, ..., β "β-hop" directed paths.) For each node set A in the laminar



Sketch of $\hat{G}$



$k = 3, \quad \beta = 2, \quad \ell = 6$

FIG. 8. *The digraph $\hat{G}$ described in the last paragraph of section* 4.2. *$\hat{G}$ has $n \geq 3k+2$ nodes and has $\geq \beta(n-2(k+1))+k$ special arcs, $\sqrt{k} \leq \beta < \sqrt{2k}$, showing that the upper bound on the number of special arcs in Theorem* 4.8 *is within a small constant factor of being tight for $n \gg k$.*

family of node sets $\{K_L, (K_L \cup \{v_1\}), \ldots, (K_L \cup \{v_1, v_2, \ldots, v_\ell\})\}$, the out-directed cut $\delta_{out}(A)$ has cardinality k, and every arc in $\delta_{out}(A)$ is a special arc.

5. Conclusions. Our analyses of the heuristics exploit results from graph theory, such as Mader's theorem [Ma 72, Theorem 1], and raise new problems in the areas of approximation algorithms and graph theory.

For a graph G and an integer $k \geq 1$, let $\mu(k, G)$ denote the minimum number of edges in a spanning subgraph of minimum degree k. For a digraph G and integer $k \geq 1$, define $\mu(k, G)$ similarly. For a graph (or digraph) G and integer $k \geq 1$, let $\mu'(k, G)$ denote the minimum number of edges (arcs) in a k-ECSS, and let $\mu''(k, G)$ denote the minimum number of edges (arcs) in a k-NCSS. While $\mu(k, G)$ can be computed efficiently via b-matchings, computing either $\mu'(k, G)$ or $\mu''(k, G)$ is NP-hard. This paper shows that (i) by computing $\mu(k-1, G)$, we can efficiently approximate $\mu''(k, G)$ to within a factor of $1 + [1/k]$ for both graphs and digraphs, and (ii) by computing $\mu(k, G)$, we can efficiently approximate $\mu'(k, G)$ to within a factor of $1 + [2/(k+1)]$

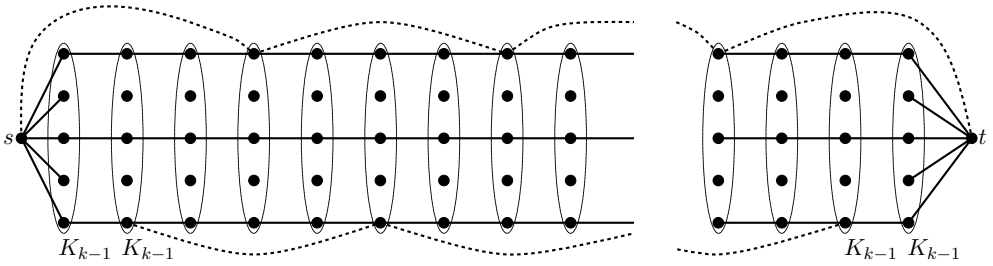


FIG. 9. *A k-node connected graph $G = (V, E)$ (with $k \geq 2$) such that the minimum size μ'' of a k-node connected spanning subgraph decreases by $(|V(G)| - 3k + 1)/(2k - 2)$ on adding one edge.*

G consists of nodes s, t, and ℓ copies of the $(k-1)$-clique, and has $k-1$ openly disjoint $s \leftrightarrow t$ paths such that each path uses exactly one node from each $(k-1)$-clique; also, G has $(\ell-1)/2$ dashed edges. Every edge in G is critical with respect to k-node connectivity. Adding the edge st to G, and then removing all the dashed edges leaves a k-node connected graph, so μ'' decreases from $|E|$ to $|E| + 1 - (\ell - 1)/2$.

A k-edge connected (and k-node connected) graph $\widetilde{G}$ such that the minimum size μ' of a k-edge connected spanning subgraph decreases by $\frac{|V(\widetilde{G})|-4k+2}{3k-3}$ on adding one edge can be obtained by modifying G as follows: "split" every $(k-1)$-clique incident with a dashed edge into a pair of $(k-1)$-cliques connected by a matching of size $(k-1)$.

for graphs, and a factor of $1 + [4/\sqrt{k}]$ for digraphs. Theorem 3.6 shows that for a k-node connected graph G, $\frac{\mu''(k,G)}{\mu'(k,G)} \leq \frac{k+1}{k}$, and Theorem 3.20 shows that for a k-node connected digraph G, $\frac{\mu''(k,G)}{\mu(k,G)} \leq \frac{k+1}{k}$. Propositions 3.4 and 3.19 show that for a k-node connected graph or digraph G, $\frac{\mu''(k,G)}{\mu(k-1,G)} \leq \frac{k+1}{k-1}$. Theorem 4.7 shows that for a k-edge connected graph G, $\frac{\mu'(k,G)}{\mu(k,G)} \leq \frac{k+3}{k+1}$.

For minimum-size k-ECSS (k-NCSS) problems, there appears to be a difficulty in achieving approximation guarantees of $1 + \frac{\omega(1)}{k^2}$. A graph theoretic function g is said to satisfy the edge Lipschitz condition if whenever graphs H and H' differ in only one edge, then $|g(H) - g(H')| \leq 1$; see [AS 92, p. 86]. Observe that $\mu(k, G)$ satisfies the edge Lipschitz condition. In contrast, both $\mu'(k, G)$ and $\mu''(k, G)$ violate this condition. First, focus on $\mu'(k, G)$ for graphs G and $k \geq 2$. Let G be the minimal k-edge connected graph obtained by "stringing" ℓ copies of the $(k+1)$-clique, i.e., take ℓ copies of the $(k+1)$-clique, and for each copy i, $1 \leq i \leq \ell$, designate a pair of distinct nodes as s_i and t_i, and then identify t_i and s_{i+1} for $i = 1, 2, \ldots, \ell - 1$. Adding the edge $s_1 t_\ell$ decreases μ' by $\ell = (|V(G)| - 1)/k$, since removing all the edges $s_i t_i$, $1 \leq i \leq \ell$, leaves a k-edge connected graph. Now consider $\mu''(k, G)$ for graphs G and $k \geq 2$. For each $k \geq 2$, there exists a k-node connected graph G such that adding a particular new edge decreases μ'' by $\frac{|V(G)|-3k+1}{2k-2}$; see Figure 9 for an illustration. For $k = 2$ and the graph in Figure 1(a), observe that μ'' decreases from $1.5|V| - 5$ to $|V| + 1$ upon adding the edge e_*. A k-edge connected (and k-node connected) graph $\widetilde{G}$ such that adding a particular new edge decreases μ' by $\frac{|V(\widetilde{G})|-4k+2}{3k-3}$ can be obtained by modifying the graph in Figure 9 as indicated in the figure caption. Garg, Santosh, and Singla [GSS 93] discuss similar issues for the minimum-size 2-NCSS problem on graphs.

Another drawback of the analysis of the k-NCSS heuristic for graphs in section 3.1 is that the size of the edge set $E' = M \cup F$ returned by the heuristic is compared against $\mu'(k, G)$, the minimum size of a k-ECSS. Given an integer $k \geq 2$, for each integer $n = 2k(i + k) + k$, where $i = 0, 1, 2, \ldots$, there exists a k-node connected,

n-node graph $\hat{G}$ such that

$$\frac{\mu''(k,\hat{G})}{\mu'(k,\hat{G})} = 1 + \frac{(k-2)}{(2k^2+k)}.$$

In view of this, for large k, a sharper lower bound will have to be employed for proving approximation guarantees substantially better than $1 + [1/2k]$ for the minimum-size k-NCSS problem. For $k = 2$ or $k = 3$, larger values of $\mu''(k,G)/\mu'(k,G)$ are given by the graph G in Figure 9 with the parameter k fixed at 2 or 3 and with $|V(G)| \gg k$: for $k = 2$, the ratio approaches $6/5$, and for $k = 3$, the ratio approaches $14/13$.

Here is another consequence of Gupta's result (see the proof of Proposition 3.8): for a bipartite graph G with minimum degree $\geq k$,

$$\frac{\mu(k-1,G)}{\mu(k,G)} \leq \frac{(k-1)}{k}.$$

This inequality does not hold for nonbipartite graphs, since for $G = K_{(k+1)}$, $\mu(k-1,G)/\mu(k,G)$ equals $(k-1)/k$ for k odd and equals $k/(k+1)$ for k even. Another result of Gupta (see [BM 76, problem 6.2.8]) shows that $\mu(k-2,G)/\mu(k,G) \leq (k-2)/(k-1)$ for all graphs G of minimum degree $\geq k$.

Acknowledgments. Thanks to W. H. Cunningham, H. R. Hind, A. V. Kotlov, U. S. R. Murty, and A. Sebő for helpful discussions. U. S. R. Murty suggested the use of Gupta's result for proving Proposition 3.8. This paper was posted on the Web sites of the authors in November 1996.

REFERENCES

[AS 92] N. Alon and J. H. Spencer, *The Probabilistic Method*, John Wiley, New York, 1992.

[Bo 78] B. Bollobás, *Extremal Graph Theory*, Academic Press, London, 1978.

[BM 76] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, American Elsevier Publishing, New York, 1976.

[Ca 93] M. Cai, *The number of vertices of degree k in a minimally k-edge-connected graph*, J. Combin. Theory Ser. B, 58 (1993), pp. 225–239.

[CKT 93] J. Cheriyan, M.-Y. Kao, and R. Thurimella, *Scan-first search and sparse certificates: An improved parallel algorithm for k-vertex connectivity*, SIAM J. Comput., 22 (1993), pp. 157–174.

[CSS 98] J. Cheriyan, A. Sebő, and Z. Szigeti, *An improved approximation algorithm for minimum size 2-edge connected spanning subgraphs*, in Proceedings of the Sixth International IPCO Conference, R. E. Bixby, E. A. Boyd, and R. Z. Rios-Mercado, eds., Lecture Notes in Comput. Sci. 1412, Springer, Berlin, 1998, pp. 126–136.

[CL 95] K. W. Chong and T. W. Lam, *Approximating biconnectivity in parallel*, Algorithmica, 21 (1998), pp. 395–410.

[CL 96] K. W. Chong and T. W. Lam, *Improving biconnectivity approximation via local optimization*, in Proceedings of the Seventh Annual ACM–SIAM Symposium on Discrete Algorithms, Atlanta, GA, ACM, New York, 1996, pp. 26–35.

[CL 96b] K. W. Chong and T. W. Lam, *Towards more precise parallel biconnectivity approximation*, in Proceedings of the Seventh International Symposium on Algorithms and Computation, Osaka, Japan, 1996.

[Ch 76] N. Christofides, *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*, Technical report 388, G.S.I.A., Carnegie-Mellon University, Pittsburgh, PA, 1976.

[CL 99] A. Czumaj and A. Lingas, *On approximability of the minimum-cost k-connected spanning subgraph problem*, in Proceedings of the 10th Annual ACM–SIAM Symposium on Discrete Algorithms, San Diego, CA, ACM, New York, 1999, pp. 281–290.

[Ed 72] J. Edmonds, *Edge-disjoint branchings*, in Combinatorial Algorithms, R. Rustin, ed., Algorithmics Press, New York, 1972, pp. 91–96.

[Fe 97] C. G. Fernandes, *A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem*, J. Algorithm, 28 (1998), pp. 105–124.

[FGHP 93] T. Fischer, A. V. Goldberg, D. J. Haglin, and S. Plotkin, *Approximating matchings in parallel*, Inform. Process. Lett., 46 (1993), pp. 115–118.

[Fr 93] A. Frank, *Submodular functions in graph theory*, Discrete Math., 111 (1993), pp. 231–243.

[FIN 93] A. Frank, T. Ibaraki, and H. Nagamochi, *On sparse subgraphs preserving connectivity properties*, J. Graph Theory, 17 (1993), pp. 275–281.

[Ga 85] H. N. Gabow, *A scaling algorithm for weighted matching on general graphs*, in Proceedings of the 26th Annual IEEE Symposium on the Foundations of Computer Science, 1985, pp. 90–100.

[Ga 95] H. N. Gabow, *A matroid approach to finding edge connectivity and packing arborescences*, J. Comput. System Sci., 50 (1995), pp. 259–273.

[GaTa 91] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for general graph matching problems*, J. ACM, 38 (1991), pp. 815–853.

[GJ 79] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[GSS 93] N. Garg, V. S. Santosh, and A. Singla, *Improved approximation algorithms for biconnected subgraphs via better lower bounding techniques*, in Proceedings of the Fourth Annual ACM–SIAM Symposium on Discrete Algorithms, Austin, TX, ACM, New York, 1993, pp. 103–111.

[Ge 95] A. M. H. Gerards, *Matching*, in Handbook of Operations Research and Management Science, M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, eds., North-Holland, Amsterdam, 1995.

[Gu 67] R. P. Gupta, *A decomposition theorem for bipartite graphs*, in *Théorie des Graphes Rome I. C. C.*, P. Rosenstiehl, ed., Dunod, Paris, 1967, pp. 135–138.

[HKe+ 95] X. Han, P. Kelsen, V. Ramachandran, and R. Tarjan, *Computing minimal spanning subgraphs in linear time*, SIAM J. Comput., 24 (1995), pp. 1332–1358.

[Jo 93] T. Jordán, *Increasing the vertex-connectivity in directed graphs*, in Proceedings of the Algorithms—ESA'93, First Annual European Symposium, Lecture Notes in Comput. Sci. 726, Springer, New York, 1993, pp. 236–247.

[Jo 95] T. Jordán, *On the optimal vertex-connectivity augmentation*, J. Combin. Theory Ser. B, 63 (1995), pp. 8–20.

[Ka 94] D. R. Karger, *Random sampling in cut, flow, and network design problems*, Math. Oper. Res., 24 (1999), pp. 383–413.

[KeR 95] P. Kelsen and V. Ramachandran, *On finding minimal two-connected subgraphs*, J. Algebra, 18 (1995), pp. 1–49.

[K 96] S. Khuller, *Approximation algorithms for finding highly connected subgraphs*, in Approximation algorithms for NP-hard Problems, D. S. Hochbaum, ed., PWS Publishing, Boston, 1996.

[KR 96] S. Khuller and B. Raghavachari, *Improved approximation algorithms for uniform connectivity problems*, J. Algebra, 21 (1996), pp. 434–450.

[KRY 95] S. Khuller, B. Raghavachari, and N. Young, *Approximating the minimum equivalent digraph*, SIAM J. Comput., 24 (1995), pp. 859–872.

[KRY 96] S. Khuller, B. Raghavachari, and N. Young, *On strongly connected digraphs with bounded cycle length*, Discrete Appl. Math., 69 (1996), pp. 281–289.

[KV 94] S. Khuller and U. Vishkin, *Biconnectivity approximations and graph carvings*, J. ACM, 41 (1994), pp. 214–235.

[L 93] L. Lovász, *Combinatorial Problems and Exercises*, North-Holland, Amsterdam, 1993.

[LP 86] L. Lovász and M. D. Plummer, *Matching Theory*, North-Holland, Amsterdam, 1986.

[Ma 71] W. Mader, *Minimale n-fach kantenzusammenhängende Graphen*, Math. Ann., 191 (1971), pp. 21–28.

[Ma 72] W. Mader, *Ecken vom Grad n in minimalen n-fach zusammenhängenden Graphen*, Arch. Math. (Basel), 23 (1972), pp. 219–224.

[Ma 85] W. Mader, *Minimal n-fach zusammenhängenden Digraphen*, J. Combin. Theory Ser. B, 38 (1985), pp. 102–117.

[NI 92] H. Nagamochi and T. Ibaraki, *A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph*, Algorithmica, 7 (1992), pp. 583–596.

[Th 89] R. Thurimella, *Techniques for the Design of Parallel Graph Algorithms*, Ph.D. thesis, The University of Texas at Austin, 1989.

[VV 99] S. Vempala and A. Vetta, *On the Minimum 2-Edge Connected Subgraph*, 1999, manuscript.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 561–575

RANDOM SAMPLING, HALFSPACE RANGE REPORTING, AND CONSTRUCTION OF ($\leq k$)-LEVELS IN THREE DIMENSIONS*

TIMOTHY M. CHAN†

Abstract. Given n points in three dimensions, we show how to answer halfspace range reporting queries in $O(\log n + k)$ expected time for an output size k. Our data structure can be preprocessed in optimal $O(n \log n)$ expected time. We apply this result to obtain the first optimal randomized algorithm for the construction of the ($\leq k$)-level in an arrangement of n planes in three dimensions. The algorithm runs in $O(n \log n + nk^2)$ expected time. Our techniques are based on random sampling. Applications in two dimensions include an improved data structure for "k nearest neighbors" queries and an algorithm that constructs the order-k Voronoi diagram in $O(n \log n + nk \log k)$ expected time.

Key words. computational geometry, range searching, levels in arrangements, nearest neighbor searching, Voronoi diagrams, randomized data structures, randomized algorithms

AMS subject classifications. 68U05, 68Q25, 68P05, 52B30

PII. S0097539798349188

1. Introduction.

1.1. Halfspace range reporting. Let P be a set of n points in d-dimensional space $\mathbb{R}^d$. We consider the problem of preprocessing P so that given any query halfspace γ, one can quickly report all points in $P \cap \gamma$. A vast literature in computational geometry has been devoted to this fundamental problem called *halfspace range reporting*, a special case of *range searching* [4, 37, 49, 53, 55]. Here are some of the major known results.

First, halfspace range reporting in the planar case ($d = 2$) was solved optimally by Chazelle, Guibas, and Lee [28]. Their data structure takes linear space and answers a query in $O(\log n + k)$ time, where k is the number of reported points. The preprocessing can be accomplished in $O(n \log n)$ time using Chazelle's algorithm for *convex layers* [22]. Unfortunately, the approach does not generalize to higher dimensions.

In the $d = 3$ case, Chazelle and Preparata [29] gave a method for answering halfspace range reporting queries in optimal time $O(\log n + k)$. The space complexity is $O(n \log^2 n \log \log n)$, as noted by Clarkson and Shor [32]. The preprocessing is $O(n \log^c n)$ for a small constant c and uses shallow *levels* in arrangements (see section 1.2). Aggarwal, Hansen, and Leighton [10] subsequently improved the space bound to $O(n \log n)$ while maintaining optimal query time. However, the preprocessing is more complex, exploiting advanced techniques such as *planar separators*; a deterministic version requires near-cubic time, while a Monte Carlo version needs $O(n \log^2 n \log \log n)$ time. In particular, it remained open whether $O(n \log n)$ preprocessing time is attainable. If one is willing to sacrifice optimality in the query bound, then a simple method [25] involving a tree of *planar point location* structures solves the problem with $O(k \log^2 n)$ query time and $O(n \log n)$ preprocessing time and space.

*Received by the editors December 14, 1998; accepted for publication (in revised form) September 24, 1999; published electronically June 3, 2000. A preliminary version of this paper appeared in Proceedings of the 39th IEEE Symposium on the Foundations of Computer Science, Palo Alto, CA, 1998. This work was done while the author was at the Department of Mathematics and Computer Science, University of Miami.

http://www.siam.org/journals/sicomp/30-2/34918.html

†Department of Computer Science, University of Waterloo, Waterloo, Ontario N2L 3G1, Canada (tmchan@math.uwaterloo.ca).

For a larger fixed dimension $d \geq 4$, Clarkson and Shor [32] used shallow *cuttings* to achieve $O(\log n + k)$ query time with $O(n^{\lfloor d/2 \rfloor + \varepsilon})$ preprocessing time and space, where ε is an arbitrarily small positive constant. Matoušek [48] obtained a certain *partition theorem* that implies a data structure with $O(n \log n)$ preprocessing time, $O(n \log \log n)$ space, and $O(n^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} n + k)$ query time. (This method can be specialized to handle the $d = 3$ case; the query time appears to be $O((\log n)^{O(\log \log n)} + k)$.) Tradeoffs between preprocessing and query time were also described by Matoušek. These higher-dimensional results were conjectured to be optimal up to n^{ε} or polylogarithmic factors.

The first result of this paper is to close the existing gap for halfspace range reporting for the case $d = 3$. In section 2, we give a relatively simple, randomized (Las Vegas) method that can answer queries in $O(\log n + k)$ expected time. This data structure requires $O(n \log n)$ space and can be preprocessed in $O(n \log n)$ expected time. The expectation here is with respect to the random choices made by the preprocessing; it is assumed that the given query halfspaces are independent of these choices. Both time bounds on preprocessing and querying are optimal.

Our approach is to consider random samples of various sizes of the dual planes and examine the *canonical triangulations* of their (≤ 0)-*levels*. With good chances, it turns out that the answer to a query with output size k can be found within some *conflict list* of the triangulation for a sample of size approximating n/k; the expected size of this list is $O(k)$. The preprocessing of all such conflict lists is done by imitating a known algorithm for the (≤ 0)-level (i.e., *convex hull* in primal space). We should remark that ideas like random sampling, canonical triangulations, and conflict lists are hardly new in this area; what is not apparent is how they can be put together to derive the optimal result. For instance, Clarkson and Shor's halfspace range reporting structure [32] uses already a top-down form of random sampling, which seems inherently suboptimal; we avoid such difficulties by adopting a bottom-up approach instead.

Our result has several important applications. For example, we can now preprocess n points in the Euclidean plane in $O(n \log n)$ expected time, such that *"k nearest neighbors" queries* can be answered in $O(\log n + k)$ expected time; this and the related *circular range reporting* problem are two of the most basic proximity problems, dating back to the beginning of computational geometry. For another application, we can enumerate the k *bichromatic closest pairs* of n planar points in $O(n \log n + k)$ expected time. See section 2.3. One less obvious application—the construction of levels in arrangements—is the second main topic of this paper.

1.2. Levels in arrangements. Given a set H of n hyperplanes in d-dimensional space $\mathbb{R}^d$ and $0 \leq k \leq n$, define the region

$$\mathrm{lev}_k(H) \;=\; \{q \in \mathbb{R}^d : \text{at most } k \text{ hyperplanes of } H \text{ pass strictly below } q\}.$$

The $(\leq k)$-*level* in the arrangement of H is the collection of faces in the arrangement that are contained in $\mathrm{lev}_k(H)$. The related notion of the k-*level* can be defined as the collection of faces contained in the boundary of $\mathrm{lev}_k(H)$. Levels in arrangements are among the most well-studied, yet most puzzling, geometric structures in both their combinatorial and computational aspects [8, 37, 43, 53].

Our initial focus is on $(\leq k)$-levels. The main combinatorial question here was answered by Clarkson and Shor [32], who showed via a beautiful random sampling argument that the $(\leq k)$-level in a fixed dimension d can have at most $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ faces. This bound is tight in the worst case. The computational complexity of $(\leq k)$-levels was also essentially resolved for $d = 2$ and $d \geq 4$. Everett, Robert, and Van

Kreveld [42] gave an $O(n \log n + nk)$-time algorithm in the plane (see also [12]). Mulmuley [52] gave a randomized algorithm for dimensions four and higher with expected running time $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$. These two results are optimal in view of Clarkson and Shor's bound and the trivial $\Omega(n \log n)$ lower bound.

The important case $d = 3$, however, remains an open problem, even though a number of algorithms have been developed. A randomized algorithm of Mulmuley [52], for instance, runs in expected time $O(nk^2 \log(n/k))$, which is just a logarithmic factor away from optimal. Agarwal et al. [2] proposed a different randomized algorithm with expected running time $O(n \log^3 n + nk^2)$, which is optimal for sufficiently large k. In section 3, we settle the complexity of the three-dimensional case completely by giving a randomized algorithm with an $O(n \log n + nk^2)$ expected time bound; this is optimal for all values of k. Before, this time bound was known only for the special case where the input planes are *nonredundant*, i.e., they all bound the (≤ 0)-level [9, 52].

Our approach deviates from the previous algorithms of Mulmuley and Agarwal et al. [2] in that it does not follow the randomized incremental paradigm [32, 53]. Instead, the idea (at a high level) is to choose a random sample of size n/k and use the canonical triangulation of its arrangement to divide the problem into roughly $O(n/k)$ subproblems of average size $O(k)$. These subproblems are created from conflict lists—computable by halfspace range reporting—and are then solved by brute force in $O(k^3)$ average time. The analysis of our algorithm is based on a *shallow cutting lemma* of Matoušek.

A similar approach can be taken to compute the k-level, although optimality is not yet attained for this problem. First, the combinatorial problem of determining the worst-case size of the k-level is wide open (the dual is related to the famous *k-set problem* [8, 11, 37]). Significant development occurred recently in the planar case $d = 2$ when Dey [34] improved the long-standing upper bound of (roughly) $O(n\sqrt{k})$ to $O(nk^{1/3})$, but the current best lower bound remains $\Omega(n \log k)$ [41]. For $d = 3$, the most recent upper bound is $O(nk^{5/3})$ [1, 35]. For higher dimensions, the current upper bound is only slightly better than the $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})$ bound for the larger ($\leq k$)-level. There is a case where the exact complexity of the k-level is known: if $d = 3$ and all input planes are nonredundant, then the k-level has size $\Theta(nk)$ for $k \leq n/2$ [32, 46]. The k-level in this case is related to the *order-k Voronoi diagram* of n points in the Euclidean plane—a natural extension to one of the most fundamental and useful geometric structures, the *Voronoi diagram*.

Some of the algorithmic results obtained by Agarwal et al. [2] on the k-level can be improved by our techniques. Specifically, their expected time bound of $O(n \log^2 n + nk^{1/3} \log^{2/3} n)$ for the construction of k-levels in the plane can be reduced to $O(n \log n + nk^{1/3} \log^{2/3} k)$. Furthermore, their $O(n \log^3 n + nk \log n)$ algorithm for order-k Voronoi diagrams in the plane can be sped up to run in $O(n \log n + nk \log k)$ expected time. These results are described in section 3.3.

2. Halfspace range reporting in $\mathbb{R}^3$.

2.1. Preliminaries. Let H be a given set of n (nonvertical) hyperplanes in $\mathbb{R}^d$. For simplicity, we assume that they are in general position; standard perturbation techniques [38, 17] can be applied to remove this assumption. The halfspace range reporting problem by *duality* [37, 53] is equivalent to the following: preprocess H so that given a query point q, one can quickly report all hyperplanes of H below q. We will actually solve a slightly harder problem: preprocess H so that given a query vertical line ℓ and a number k, one can quickly report the k lowest hyperplanes along ℓ

(i.e., hyperplanes defining the k lowest intersections with ℓ). The connection between halfspace range reporting queries and such *"k lowest hyperplanes" queries* will be explained later.

Given a subset $R \subset H$, the (≤ 0)-level in the arrangement of R (also called the *lower envelope* of R) is a convex polyhedron. Computing this polyhedron is equivalent to constructing the *convex hull* [37, 53, 55] by duality. For $d = 3$, several $O(|R| \log |R|)$-time algorithms are known [32, 54], among the simplest of which are based on the randomized incremental paradigm.

Let $\mathrm{CT}_0(R)$ denote the collection of (closed) full-dimensional simplices in the *canonical triangulation* of the (≤ 0)-level, as defined for instance by Clarkson [31] (also called the *bottom-vertex triangulation*). The precise definition of this triangulation is not important to us except in the proof of the sampling lemma below. The only facts we need is that the triangulation is linear-time constructible and that the simplices in $\mathrm{CT}_0(R)$ are all vertical cylinders containing the point $(0, \ldots, 0, -\infty)$.

Thus, a vertical line ℓ hits precisely one simplex Δ in $\mathrm{CT}_0(R)$, if one ignores degenerate cases. For $d = 3$, this simplex Δ can be identified in $O(\log |R|)$ time after an $O(|R| \log |R|)$-time preprocessing, as the problem projects down to *planar point location* [37, 55].

Given a simplex Δ, the *conflict list* H_Δ is defined as the set of all hyperplanes of H intersecting Δ. The following two sampling results are needed in the analysis of our data structure. Both follow from the general probabilistic technique by Clarkson and Shor [32]. (The first is often used in analyses of randomized convex hull algorithms.)

LEMMA 2.1. *Let $1 \leq r \leq n$ and consider a random sample $R \subset H$ of size r.*

(i) *The expected value of the sum $\sum_\Delta |H_\Delta|$ over all simplices $\Delta \in \mathrm{CT}_0(R)$ is $O(r^{\lfloor d/2 \rfloor} \cdot n/r)$.*

(ii) *For any fixed vertical line ℓ, the expected value of $|H_\Delta|$ for the simplex $\Delta \in \mathrm{CT}_0(R)$ hit by ℓ is $O(n/r)$.*

2.2. The data structure for $d = 3$. We take a common approach called *bottom-up sampling* by Mulmuley [53]. Choose a random permutation $h_1, \ldots, h_n$ of the set H. Define $R_i = \{h_1, \ldots, h_{2^i}\}$ for $i = 0, 1, \ldots, \log n$ (without loss of generality, say n is a power of 2; logarithms are in base 2). The result is a sequence ("hierarchy") of random samples

$$R_0 \subset R_1 \subset \cdots \subset R_{\log n} = H,$$

where $|R_i| = 2^i$. Our basic data structure is simple: it consists of location structures for $\mathrm{CT}_0(R_i)$, along with the conflict list H_Δ for all the simplices $\Delta \in \mathrm{CT}_0(R_i)$. For each R_i, the space needed is $O(\sum_{\Delta \in \mathrm{CT}_0(R_i)} |H_\Delta|)$, which has expected value $O(n)$ by Lemma 2.1(i). The total expected space is therefore $O(n \log n)$. We can guarantee that space is within a constant factor of this bound by repeating for an expected constant number of trials.

We now describe how to build this data structure efficiently. First the location structures can all be constructed in time $O(\sum_{i=1}^{\log n} |R_i| \log |R_i|) = O(n \log n)$. The nontrivial part is the computation of the conflict lists. It turns out that this can be done by just modifying a known randomized algorithm for convex hulls in $\mathbb{R}^3$. We consider here Clarkson and Shor's original method [32], which maintains global conflict information incrementally. (Note that online methods based on *history* [53] are not suitable for our purposes.)

Specifically, at the 2^ith step of Clarkson and Shor's method (in the version described by Mulmuley's or Motwani and Raghavan's text [53, 50]), we not only have

all vertices of the (≤ 0)-level in the arrangement of R_i, but in addition have for each plane $h \in H$ a pointer to some vertex lying above h (if one exists). By a graph search, we can generate all vertices lying above each h. As a result, we have for each vertex v a list of all planes of H below v. Given a simplex $\Delta \in \mathrm{CT}_0(R_i)$ with vertices v_1, v_2, v_3, the conflict list H_Δ is just the union of the list of planes of H below the v_j's.

Clarkson and Shor's method runs in $O(n \log n)$ expected time. The extra work done at the 2^ith step to produce the conflict lists is $O(\sum_{\Delta \in \mathrm{CT}_0(R_i)} |H_\Delta|)$, which has expected value $O(n)$ by Lemma 2.1(i). Hence, the whole preprocessing of our data structure can be performed in $O(n \log n)$ expected time.

We now describe the basic query algorithm. Given vertical line ℓ and number k as input parameters, the algorithm is usually able to find the k lowest planes of H along ℓ but occasionally may report failure instead. The probability of failure is controlled by a third input parameter $\delta > 0$.

ALGORITHM ANSWER-QUERY(ℓ, k, δ)

1. let $i = \lceil \log \lceil n\delta/k \rceil \rceil$
2. identify the simplex $\Delta \in \mathrm{CT}_0(R_i)$ cut by ℓ
3. if $|H_\Delta| > k/\delta^2$ then return "failed"
4. if fewer than k planes of H_Δ intersect $\ell \cap \Delta$ then return "failed"
5. return the k lowest planes of H_Δ along ℓ

The algorithm is correct: what is returned in line 5 is precisely the k lowest planes of H along ℓ if failure is not reported in line 4. The running time of ANSWER-QUERY() is $O(\log n + k/\delta^2)$, since line 2 takes $O(\log n)$ time by planar point location, and lines 4 and 5 take $O(k/\delta^2)$ time if failure is not reported in line 3. (Line 5 is an instance of what Chazelle referred to as *filtering search* [23].)

We now bound the failure probability for any fixed choice of ℓ, k, and δ. By Lemma 2.1(ii), the expected value of $|H_\Delta|$ is $O(n/|R_i|) = O(k/\delta)$, and so by Markov's inequality, the probability that H_Δ exceeds k/δ^2 is $O(\delta)$. Thus line 3 reports failure with probability $O(\delta)$. On the other hand, letting q denote the kth lowest intersection of H along ℓ, we see that line 4 reports failure only if $q \notin \Delta$, or equivalently, $q \notin \mathrm{lev}_0(R_i)$. This is true only if one of the k planes below q is chosen to be in the sample R_i. As q is independent of R_i, this can happen with probability at most $k|R_i|/n = O(\delta)$.

We can summarize our result as follows.

THEOREM 2.2. *In $O(n \log n)$ expected time, one can preprocess n planes in $\mathbb{R}^3$ into a randomized data structure of $O(n \log n)$ size, such that there is a procedure with the following behavior. Given any fixed vertical line ℓ, number k, and $\delta > 0$, the procedure either reports the k lowest planes along ℓ or reports failure. The probability of failure is $O(\delta)$, but the procedure always runs within $O(\log n + k/\delta^2)$ time.*

2.3. Consequences. It is desirable to modify the data structure to have a query algorithm that never fails. This modification can be done in two stages. First we observe that having three independent versions of the basic data structure can reduce the failure probability to $O(\delta^3)$ in Theorem 2.2.

COROLLARY 2.3. *In $O(n \log n)$ expected time, one can preprocess n planes in $\mathbb{R}^3$ into a randomized data structure of $O(n \log n)$ size, such that there is a procedure satisfying the criteria stated in Theorem 2.2 but with failure probability $O(\delta^3)$.*

Next we apply the basic query algorithm on a sequence of choices for the parameter δ in order to guarantee success.

COROLLARY 2.4. *In $O(n \log n)$ expected time, one can preprocess n planes in $\mathbb{R}^3$ into a randomized data structure of $O(n \log n)$ size, such that any "k lowest planes"*

query can be answered in $O(\log n + k)$ expected time.

Proof. Let $\delta_i = 2^{-i}$. Run the procedure of Corollary 2.3 for $\delta = \delta_1, \delta_2, \ldots$ until it succeeds. Let X_i denote the 0-1 random variable with value 1 when the procedure fails for $\delta = \delta_i$. The total running time is bounded asymptotically by

$$(\log n + k/\delta_1^2) + \sum_{i>1} (\log n + k/\delta_i^2)\, X_{i-1},$$

which has expected value $O(\sum_{i>1} (\log n + k/\delta_i^2)\, \delta_{i-1}^3) = O(\log n + k)$. □

We still have to explain how halfspace range reporting queries reduces to "k lowest planes" queries. This can be done by a standard technique of "guessing" the parameter k.

COROLLARY 2.5. *In $O(n \log n)$ expected time, one can preprocess n points in $\mathbb{R}^3$ into a randomized data structure of $O(n \log n)$ size, such that a halfspace range reporting query can be answered in $O(\log n + k)$ expected time, where k is the number of points reported.*

Proof. Recall that a halfspace range reporting query in dual space corresponds to finding all k planes below a given point q; the value of k is not known in advance. Let $k_i = 2^i \log n$, and let ℓ be the vertical line at q. This task can be accomplished by searching for the k_ith lowest plane along ℓ for $i = 1, 2, \ldots$ until such a plane lies above q; then we simply examine the k_i lowest planes along ℓ and report those that are actually below q. The expected running time is asymptotically bounded by

$$(\log n + k_1) + \sum_{k_{i-1} < k} (\log n + k_i) = O(\log n + k),$$

because of Corollary 2.4. □

Remarks.

1. While the preprocessing and query time are optimal, they are only expected bounds; furthermore, the space complexity can be improved. Using advanced tools and a larger preprocessing time, Appendix C gives a modification of our data structure that is deterministic and uses only $O(n \log \log n)$ space.

2. It is worthwhile to compare Chazelle, Guibas, and Lee's optimal method [28] with the specialization of our method in two dimensions. Ours seems easier to implement as convex layers are not involved.

3. Higher-dimensional extensions are possible, although we do not see any significant improvements.

By a standard lifting map, Corollaries 2.4 and 2.5 imply a new method for *"k nearest neighbors"* and *circular range reporting queries* [10, 16, 25, 37, 53, 55, 57] in the Euclidean plane.

COROLLARY 2.6. *In $O(n \log n)$ expected time, one can preprocess n point sites in $\mathbb{R}^2$ into a randomized data structure of $O(n \log n)$ size, such that the k closest (or farthest) sites of a given point can be found in $O(\log n + k)$ expected time.*

COROLLARY 2.7. *In $O(n \log n)$ expected time, one can preprocess n point sites in $\mathbb{R}^2$ into a randomized data structure of $O(n \log n)$ size, such that all sites inside (or outside) a given circle can be reported in $O(\log n + k)$ expected time, where k is the output size.*

Another proximity application in the Euclidean plane is an optimal randomized algorithm for enumerating the k *bichromatic closest pairs* [45]. The author [19] gave a randomized reduction of this problem to a reporting problem, which in turn reduces

to answering n offline halfspace range reporting queries in $\mathbb{R}^3$, where the total output size is k.

COROLLARY 2.8. *Given n-point sets P and Q in $\mathbb{R}^2$, one can enumerate the k closest (or farthest) pairs from $P \times Q$ (not necessarily in sorted order) in expected time $O(n \log n + k)$.*

3. ($\leq k$)-levels in $\mathbb{R}^3$.

3.1. Preliminaries. Let H be a given set of n hyperplanes in $\mathbb{R}^d$ in general position. Our goal in this section is to construct the facial structure of the ($\leq k$)-level in the arrangement of H. As before, the key concept is the canonical triangulation. Given a subset $R \subset H$, define $\mathrm{CT}(R)$ to be the collection of full-dimensional simplices in the canonical triangulation of the faces in the arrangement of R. Define $\mathrm{CT}_k(R)$ similarly for the faces of the ($\leq k$)-level in the arrangement of R.

It is somewhat more involved to compute conflict lists for simplices arising from the construction of ($\leq k$)-levels, e.g., the simplices are not necessarily vertical cylinders now. For this reason, we define $\Delta^* = \mathrm{conv}(\Delta \cup \{(0, \ldots, 0, -\infty)\})$, the *vertical extension* of a simplex Δ. For $d = 3$, the extended conflict list H_{Δ^*} can be computed in $O(\log n + |H_{\Delta^*}|)$ expected time after $O(n \log n)$-time preprocessing by Corollary 2.5: listing all planes of H below a given point reduces to halfspace range reporting, and if $v_1, \ldots, v_4$ denote the vertices of Δ, then H_{Δ^*} is just the union of the lists of planes of H below the v_j's.

One other important definition is the following: a simplex Δ is *relevant* if it intersects the region $\mathrm{lev}_k(H)$. The sampling lemma below is stated implicitly in the proof of Matoušek's *shallow cutting lemma* [48] (see Appendix B) and is needed in the analysis of our algorithm. Matoušek's proof uses probabilistic arguments of Chazelle and Friedman [27]; see Appendix A for more details.

LEMMA 3.1. *Let $1 \leq r \leq n$ and $q = kr/n + 1$. Let $f(n)$ be a* regular *function, i.e., a nondecreasing function satisfying $f(2n) = O(f(n))$. Consider a random sample $R \subset H$ of size r. The expected value of the sum $\sum_\Delta f(|H_\Delta|)$ over all relevant simplices $\Delta \in \mathrm{CT}(R)$ is bounded by $O(r^{\lfloor d/2 \rfloor} q^{\lceil d/2 \rceil} f(n/r))$.*

3.2. The algorithm for $d = 3$. The basic outline of our ($\leq k$)-level algorithm is quite simple: we pick a random sample $R \subset H$ of size n/k, generate all relevant simplices of $\mathrm{CT}(R)$, compute the ($\leq k$)-level within each such simplex Δ, and then combine the solutions.

The ($\leq k$)-level in the arrangement of H within a relevant simplex Δ can be constructed by the following procedure:

ALGORITHM SOLVE-SUBPROBLEM(Δ), where Δ is a relevant simplex

1. compute conflict list H_{Δ^*}
2. construct the ($\leq k$)-level in the arrangement of H_{Δ^*} by brute force
3. clip the resulting faces to Δ

The correctness of the procedure is easy to see (as $\mathrm{lev}_k(H) \cap \Delta = \mathrm{lev}_k(H_{\Delta^*}) \cap \Delta$). Line 1 takes $O(\log n + |H_{\Delta^*}|)$ expected time, whereas lines 2 and 3 take $O(|H_{\Delta^*}|^3)$ time by a brute force method: construct the entire arrangement of H_{Δ^*} [39] and extract the desired substructure. Since Δ is relevant, we can bound $|H_{\Delta^*}|$ by $k + |H_\Delta|$. Therefore, SOLVE-SUBPROBLEM(Δ) runs in expected time $O(\log n + (k + |H_\Delta|)^3)$.

We now describe how to generate all relevant simplices of $\mathrm{CT}(R)$. Observe that any relevant simplex of $\mathrm{CT}(R)$ must belong to $\mathrm{CT}_k(R)$ (since $\mathrm{lev}_k(H) \subseteq \mathrm{lev}_k(R)$). Assume that $\mathrm{CT}_k(R)$ is available. To generate all relevant simplices of $\mathrm{CT}_k(R)$, we perform a graph search. Say that two simplices of $\mathrm{CT}_k(R)$ are *adjacent* if they share a

common facet. Notice that a simplex can be adjacent to at most four other simplices in $\mathrm{CT}_k(R)$. Consider the following procedure.

ALGORITHM GENERATE-SUBPROBLEMS(Δ), where $\Delta \in \mathrm{CT}_k(R)$

1. mark Δ
2. SOLVE-SUBPROBLEM(Δ)
3. for each unmarked simplex Δ' adjacent to Δ do
4. let ξ be the facet shared by Δ and Δ'
5. if ξ intersects $\mathrm{lev}_k(H)$ then GENERATE-SUBPROBLEMS(Δ')

The test in line 5 can be easily accomplished from the information obtained from line 2. An initial relevant simplex $\Delta_0 \in \mathrm{CT}_k(R)$ is easy to find (e.g., pick any simplex from $\mathrm{CT}_0(R)$). If all simplices are initially unmarked and GENERATE-SUBPROBLEMS(Δ_0) is called, then all relevant simplices of $\mathrm{CT}_k(R)$ (and thus of $\mathrm{CT}(R)$) will be marked because the region $\mathrm{lev}_k(H)$ is connected.

Combining the solutions is straightforward. We just take all the output from line 2 of GENERATE-SUBPROBLEMS() in this process and "stitch" these substructures together to get the complete $(\leq k)$-level in the arrangement of H. (The implementation details of stitching are not entirely trivial though; for example, one needs to match features along common facets between simplices. This can be done in linear time, for instance, by labeling planes as integers and faces as tuples and then applying a radix sort.)

Excluding the $O(n \log n)$ preprocessing time, the expected time bound for the calls to GENERATE-SUBPROBLEMS() is

$$O\left(\sum_{\Delta}(\log n + (k + |H_\Delta|)^3)\right),$$

where the sum is taken over all relevant simplices Δ of $\mathrm{CT}(R)$. Lemma 3.1 tells us that the expectation of the above expression with respect to R for a sample size $r = n/k$ is

$$O((n/k)(\log n + k^3)) \;=\; O(n \log n + nk^2).$$

For the final piece of the algorithm, we still have to explain how $\mathrm{CT}_k(R)$ is obtained in the first place. This can be done by recursively constructing the $(\leq k)$-level in the arrangement of R. The total expected running time of our algorithm is then given by the recurrence

$$T_k(n) \;=\; T_k(n/k) \,+\, O(n \log n + nk^2),$$

which solves to $T_k(n) = O(n \log n + nk^2)$, assuming (without loss of generality) that $k \geq 2$.

THEOREM 3.2. *The $(\leq k)$-level in an arrangement of n planes in $\mathbb{R}^3$ can be constructed in $O(n \log n + nk^2)$ expected time.*

Remarks.

1. Although our algorithm is optimal for worst-case output, the running time is not the best possible if the $(\leq k)$-level has size smaller than $\Theta(nk^2)$. It remains open to devise an optimal *output-sensitive* algorithm [6, 18, 51].

2. Derandomization seems to require modification to our algorithm because the sample size is quite large. Appendix B sketches an optimal deterministic method for $(\leq k)$-levels for large k (specifically, $\log k = \Omega(\log n)$). This method works in any fixed dimension but uses advanced tools.

3. Specialization to $d = 2$ yields a new algorithm for ($\leq k$)-levels in the plane with expected running time $O(n \log n + nk)$.

3.3. Application to *k*-levels. The algorithm can be modified to construct the k-level in the arrangement of H for $d = 3$. To get good results, we need to have available an algorithm for constructing the k-level in subcubic time, say $O(f(n))$ time (randomized or deterministic).

We will make SOLVE-SUBPROBLEM(Δ) output the faces of the k-level clipped to Δ. This can be done by changing line 2 to construct just the k-level in the arrangement of H_{Δ^*} in $O(f(|H_{\Delta^*}|))$ time. In GENERATE-SUBPROBLEMS(Δ), the test in line 5 can also be performed in $O(f(|H_{\Delta^*}|))$ time, since it is equivalent to deciding whether ξ intersects the region beneath the k-level in the arrangement of H_{Δ^*}.

Thus, ignoring the $O(n \log n)$ preprocessing, the expected time to construct the entire k-level is

$$O\left(\sum_{\Delta}(\log n + f(k + |H_\Delta|))\right),$$

where the sum is taken over all relevant simplices Δ of $\mathrm{CT}(R)$. By Lemma 3.1, the expectation with respect to R for a sample size $r = n/k$ and a regular function f is $O((n/k)(\log n + f(k)))$.

Note that the recursive call to compute $\mathrm{CT}_k(R)$ is not needed now: by Theorem 3.2, the ($\leq k$)-level in the arrangement of R can be constructed in expected time $O((n/k)\log(n/k) + (n/k)k^2)$. Since $f(k) = \Omega(k^2)$, the overall complexity is $O(n \log n + (n/k)f(k))$.

The same strategy applies to k-levels in dimension $d = 2$. We therefore have shown the following.

THEOREM 3.3. *Let $\mathcal{H}$ be a class of lines in $\mathbb{R}^2$ or planes in $\mathbb{R}^3$. Suppose that the k-level in the arrangement of any set $H \subset \mathcal{H}$ of size n can be computed in $O(f(n))$ expected time, where $f(n)$ is a regular function. Then the k-level can be constructed in $O(n \log n + (n/k)f(k))$ expected time.*

Specifically, Agarwal et al. [2] have obtained the following bounds via randomized incremental construction: for lines in $\mathbb{R}^2$, $f(n) = n^{4/3}\log^{2/3} n$; and for planes in $\mathbb{R}^3$ that are tangent to the unit paraboloid, $f(n) = n^2 \log n$. The first case takes into account Dey's recent combinatorial bound on the k-level (or more precisely, on the $O(\log n)$ consecutive levels below the k-level). The second case occurs in the construction of *order-k Voronoi diagrams* [13, 37, 53, 55, 57] in the Euclidean plane by the standard lifting map; see Table 3.1 for some previous algorithms. For arbitrary planes in $\mathbb{R}^3$, we have $f(n) = n^{8/3+\varepsilon}$ by an algorithm of Agarwal and Matoušek [6] (with the known combinatorial bound [35]). Hence, Theorem 3.3 implies the following corollary.

COROLLARY 3.4. *The k-level in an arrangement of n lines in $\mathbb{R}^2$ can be constructed in expected time $O(n \log n + nk^{1/3}\log^{2/3} k)$.*

COROLLARY 3.5. *The order-k Voronoi diagram of n point sites in $\mathbb{R}^2$—i.e., the planar subdivision where two points belong to the same region iff they have the same set of k closest (or farthest) sites—can be constructed in expected time $O(n \log n + nk \log k)$.*

COROLLARY 3.6. *The k-level in an arrangement of n planes in $\mathbb{R}^3$ can be constructed in expected time $O(n \log n + nk^{5/3+\varepsilon})$.*

TABLE 3.1
*History of algorithms for the order-k Voronoi diagram in $\mathbb{R}^2$ ($k \le n/2$). Year refers to date of journal publication. (Bounds marked * actually apply to all diagrams of order 1 to k.)*

Year	Time bound	References
1982	$O(nk^2 \log n)^*$	Lee [46]
1986	$O(n^3)^*$	Edelsbrunner, O'Rourke, and Seidel [39]
1986	$O(nk\sqrt{n}\log n)$	Edelsbrunner [36]
1987	$O(n^2 + nk\log^2 n)$	Chazelle and Edelsbrunner [26]
1987	$O(n^{1+\varepsilon}k)$ rand.	Clarkson [30]
1989	$O(n\log n + nk^2)^*$	Aggarwal, Guibas, Saxe, and Shor [9]
1991	$O(n\log n + nk^2)^*$ rand.	Mulmuley [52]
1992	$O(nk\log^2 n + nk^2)$ rand. online	Aurenhammer and Schwarzkopf [14]
1993	$O(n\log n + nk^3)^*$ rand. online	Boissonnat, Devillers, and Teillaud [15]
1995	$O(n^{1+\varepsilon}k)$	Agarwal and Matoušek [6]
1998	$O(n\log^3 n + nk\log n)$ rand.	Agarwal, de Berg, Matoušek, and Schwarzkopf [2]
now	$O(n\log n + nk\log k)$ rand.	this paper
	$O(nk\log^2 k\,(\log n/\log k)^{O(1)})$	appendix

Remarks.

1. Theorem 3.3 can be viewed as an algorithmic version of a combinatorial result of Agarwal et al. [1], who showed basically that if the k-level in $\mathbb{R}^2$ or $\mathbb{R}^3$ has worst-case complexity $O(f(n))$, then the k-level has complexity $O((n/k)f(k))$.

2. Corollary 3.4 compares favorably with an output-sensitive algorithm of Edelsbrunner and Welzl [40], which runs in time $O(n\log n + f\log^2 n)$ for an f-face output. Only slight improvements of this algorithm were known: Cole, Sharir, and Yap [33] discussed how to reduce the second term to $O((n+f)\log^2 k)$ and the author [18] showed how to reduce the first term to $O(n\log f)$.

3. Corollary 3.5 is currently the best result for the construction of a single order-k Voronoi diagram in the plane. It is optimal for $k = O(\log n/\log\log n)$. If $f(n)$ can be improved to $O(n^2)$, then Theorem 3.3 would imply a randomized algorithm running in optimal $O(n\log n + nk)$ expected time for any $k \le n/2$.

4. It is interesting to note that while in the past, levels in arrangements have been used to solve the halfspace range reporting problem, we have taken just the reverse approach, using halfspace range reporting to construct levels. Is there a more direct way to construct levels with the same efficiency?

Update. After the conference version of this paper, Ramos [56] has recently improved Agarwal et al.'s algorithm [2] for the planar order-k Voronoi diagram and has obtained $f(n) = n^2 2^{O(\log^* n)}$, thereby reducing the expected time bound in Corollary 3.5 to $O(n\log n + nk2^{O(\log^* k)})$. For k-levels of lines in the plane, the author [21] has noted an improved bound $f(n) = n^{4/3}$, which reduces the expected time bound in Corollary 3.4 to $O(n\log n + nk^{1/3})$. There are also some new breakthroughs in output-sensitive algorithms for two-dimensional levels by Har-Peled [44] and the author [20, 21].

Appendix A. Proof sketch of Lemma 3.1. As Lemma 3.1 is the key in proving the optimality of our randomized ($\le k$)-level algorithm, a proof sketch of the lemma may be appropriate to give the reader an idea why the probabilistic bound holds. As mentioned earlier, the details of the proof are essentially embedded in the paper by Matoušek [48]. We highlight the main points here in somewhat different notation.

First the following definition is helpful: we say that a simplex Δ is *j-good* if it is contained inside the region $\mathrm{lev}_j(H)$. An immediate observation is that any relevant

simplex Δ is $(k + |H_\Delta|)$-good. It turns out that bounding the number of j-good simplices is easier than bounding the number of relevant simplices:

> The expected number of j-good simplices $\Delta \in \mathrm{CT}(R)$ is bounded by $O((r/n)^d n^{\lfloor d/2 \rfloor} j^{\lceil d/2 \rceil})$.

The reason is this: as can be seen from the definition of canonical triangulations, the number of j-good simplices of $\mathrm{CT}(R)$ is linear in the number of vertices of $\mathrm{CT}(R)$ that are contained in $\mathrm{lev}_j(H)$. We know that $\mathrm{lev}_j(H)$ has $O(n^{\lfloor d/2 \rfloor} j^{\lceil d/2 \rceil})$ vertices, and the probability that a fixed vertex in the arrangement of H appears in the arrangement of R is $O((r/n)^d)$.

The second step is to observe that the conflict size $|H_\Delta|$ is usually of the order of n/r. For instance, arguments by Clarkson and Shor [32] show that the average conflict size is expected to be $O(n/r)$. We will actually need a stronger result by Chazelle and Friedman [27] (see also [7]), stating that the number of conflict sizes of the order of tn/r decreases exponentially with t. Specifically, one can prove the following statement from our earlier bound on j-good simplices:

> The expected number of j-good simplices $\Delta \in \mathrm{CT}(R)$ with $|H_\Delta| \geq (t-1)n/r$ is $O(2^{-t}(r/n)^d n^{\lfloor d/2 \rfloor} j^{\lceil d/2 \rceil})$.

As a consequence, we can then bound a similar quantity for relevant simplices if we just substitute $k + tn/r$ for j:

> The expected number of relevant simplices $\Delta \in \mathrm{CT}(R)$ with $(t-1)n/r \leq |H_\Delta| \leq tn/r$ is $O(2^{-t}(r/n)^d n^{\lfloor d/2 \rfloor}(k + tn/r)^{\lceil d/2 \rceil})$.

Finally, we can bound the expectation of the sum $\sum_\Delta f(|H_\Delta|)$ over all relevant simplices $\Delta \in \mathrm{CT}(R)$ asymptotically by

$$\sum_{t=1}^{\infty} 2^{-t}(r/n)^d n^{\lfloor d/2 \rfloor}(k + tn/r)^{\lceil d/2 \rceil} f(tn/r) \;=\; O((r/n)^d n^{\lfloor d/2 \rfloor}(k + n/r)^{\lceil d/2 \rceil} f(n/r)),$$

since f is regular. Notice the above expression is identical to $O(r^{\lfloor d/2 \rfloor} q^{\lceil d/2 \rceil} f(n/r))$.

Appendix B. A deterministic algorithm for ($\leq k$)-levels in $\mathbb{R}^d$. In the second appendix, we briefly consider derandomization of our ($\leq k$)-level algorithm in an arbitrary fixed dimension. An optimal bound is obtained only when k is sufficiently large. The approach employs a deterministic version of Lemma 3.1, derived by Matoušek [48] using tools such as the *method of conditional probabilities* and *ε-approximations*.

LEMMA B.1 (Matoušek's shallow cutting lemma). *Let $1 \leq r \leq n$ and $q = kr/n + 1$. One can cover $\mathrm{lev}_k(H)$ by a collection of $O(r^{\lfloor d/2 \rfloor} q^{\lceil d/2 \rceil})$ simplices such that $|H_\Delta| \leq n/r$ for each simplex Δ. Furthermore, the simplices have disjoint interiors. They can be constructed in $O(n \log r)$ time, provided that $r \leq n^\alpha$ for a sufficiently small constant $\alpha > 0$ depending on d.*

Worst-case efficiency demands us to use small values of r, so we will construct the ($\leq k$)-level by divide-and-conquer: (i) pick $r = \min\{n/k, n^\alpha\}$ and find the collection of simplices by the above lemma; (ii) for each simplex Δ, compute conflict list H_{Δ^*}; (iii) construct the ($\leq k$)-level in the arrangement of H_{Δ^*} by recursion; and (iv) finally combine the solutions by stitching.

Step (ii) requires some explanation. From the discussion of our three-dimensional algorithm, we see that a conflict list can be computed by answering a constant number of halfspace range reporting queries. With known results [48], this requires $O(n^\beta + |H_{\Delta^*}|)$ time after $O(n \log n)$-time preprocessing, for a constant $\beta \approx 1 - 1/\lfloor d/2 \rfloor$. Notice

that if $|H_{\Delta^*}|$ exceeds $k+|H_\Delta|$, the simplex Δ is not relevant and need not be considered in step (iii). So, we can ensure that a query runs within time $O(n^\beta + k + n/r)$.

Accounting all the costs, we derive this recurrence for the total running time:

$$T_k(n) \;=\; O(n \log n) \;+\; O(r^{\lfloor d/2 \rfloor} q^{\lceil d/2 \rceil})\,(n^\beta + T_k(k + n/r)).$$

Assuming that $\alpha < (1-\beta)/\lfloor d/2 \rfloor$ without loss of generality, we can simplify the above to

$$T_k(n) \;=\; O(n \log n) \;+\; O(r^{\lfloor d/2 \rfloor})\,T_k(k + n/r).$$

Our base case is when $n^{1-\alpha} \le k$. Here, $r = n/k$, and we solve the subproblems directly by constructing entire arrangements in $T_k(2k) = O(k^d)$ time [39]; the overall time bound is

$$T_k(n) \;=\; O(n \log n + (n/k)^{\lfloor d/2 \rfloor} k^d) \;=\; O(n \log n + n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil}).$$

If $n^{1-\alpha} > k$, then $r = n^\alpha$, and the recurrence becomes

$$T_k(n) \;=\; O(n \log n) \;+\; O(n^{\alpha \lfloor d/2 \rfloor})\,T_k(2n^{1-\alpha}),$$

which solves to

$$T_k(n) \;=\; O\left((n \log n + n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil})\left(\frac{\log n}{\log k}\right)^{O(1)}\right) \;=\; O\left(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil}\left(\frac{\log n}{\log k}\right)^{O(1)}\right).$$

THEOREM B.2. *The $(\le k)$-level in an arrangement of n hyperplanes in $\mathbb{R}^d$ can be constructed deterministically in time $O(n^{\lfloor d/2 \rfloor} k^{\lceil d/2 \rceil} (\log n/\log k)^{O(1)})$.*

A similar approach works for k-levels and order-k Voronoi diagrams. We mention that for the latter problem in two dimensions, the best deterministic result can be achieved with Chazelle and Edelsbrunner's $O(n^2 \log^2 n)$ bound [26] for the base cases.

THEOREM B.3. *The order-k Voronoi diagram of n point sites in $\mathbb{R}^2$ can be constructed deterministically in time $O(nk \log^2 k\,(\log n/\log k)^{O(1)})$.*

Remarks.

1. The use of the shallow cutting lemma to construct levels deterministically has been noted before in a paper by Agarwal, Efrat, and Sharir [3]; however, our deterministic bounds appear new.

2. Theorem B.2 is worst-case optimal if $k = \Omega(n^\varepsilon)$ for some constant $\varepsilon > 0$. For small k, optimal derandomization for arbitrary dimensions appears difficult, as can be seen from Chazelle's work on convex hulls [24].

Update. The author [20, 21] has recently improved the bound in Theorem B.3 slightly to $O(nk \log^{1+\varepsilon} n\,(\log n/\log k)^{O(1)})$.

Appendix C. A deterministic data structure for halfspace range reporting in $\mathbb{R}^3$. In this final appendix, we revisit the halfspace range reporting problem in $\mathbb{R}^3$ and give a deterministic data structure with space $O(n \log\log n)$ and worst-case query time $O(\log n + k)$. The space bound improves the one by Aggarwal, Hansen, and Leighton [10].

The approach is based on our randomized method, but to obtain a successful derandomization, we need to replace the (≤ 0)-levels of the samples (and their canonical triangulations) with suitable structures. Shallow cuttings serve exactly this purpose, but first a slight variant is stated for convenience.

LEMMA C.1. *Let $d = 3$. One can cover $\mathrm{lev}_k(H)$ by a collection $\mathcal{T}_k$ of $O(n/k)$ simplices such that $|H_\Delta| = O(k)$ for each $\Delta \in \mathcal{T}_k$. Furthermore, the simplices have disjoint interiors, each containing $(0, 0, -\infty)$.*

Proof. Let Ξ be a collection of simplices satisfying Lemma B.1 with $r = n/k$. Without loss of generality, we may assume that each simplex $\Delta \in \Xi$ is relevant; thus, each vertex has at most $|H_\Delta| + k \leq 2k$ planes of H below it. Now, let U be the union of Ξ and define $\mathcal{T}_k$ to be a triangulation of U into vertical cylinders. Because a plane in H_Δ must lie below one of the vertices of Δ, we have $|H_\Delta| \leq 6k$ for each $\Delta \in \mathcal{T}_k$. □

The chief ingredient to reduce space from $O(n \log n)$ to $O(n \log\log n)$ is bootstrapping with an $O(n)$-space structure that has query time $O(n^\beta + k)$ for a constant $\beta < 1$. Known results on the simplex range searching [47] imply that $\beta \approx 2/3$ is possible in $\mathbb{R}^3$; the time bound applies to "k lowest planes" queries as well [5].

Define a sequence $k_1, k_2, \ldots$ by the formula $k_i = \left\lfloor k_{i-1}^{1/\beta} \right\rfloor$, starting with a constant and ending when the term reaches n. Evidently, there are $O(\log\log n)$ terms. Our data structure consists of a hierarchy of triangulations constructed by Lemma C.1: $\mathcal{T}_{k_1}, \mathcal{T}_{k_2}, \ldots$. In addition, we build a location structure for $\mathcal{T}_{k_i}$ and store the linear-space range searching structure for each conflict list H_Δ ($\Delta \in \mathcal{T}_{k_i}$). The storage requirement is $O((n/k)k) = O(n)$ for each $\mathcal{T}_{k_i}$, and $O(n \log\log n)$ overall.

To find the k lowest planes of H along a vertical line ℓ, let index i satisfy $k_{i-1} \leq k < k_i$, and determine the simplex $\Delta \in \mathcal{T}_{k_i}$ hit by ℓ; by projection, this is a planar point location problem and takes $O(\log n)$ time. As the answer is a subset of the conflict list H_Δ (since $\mathcal{T}_{k_i}$ covers $\mathrm{lev}_k(H)$), we can use the range searching structure for H_Δ to answer the query. The total query time is

$$O(\log n + |H_\Delta|^\beta + k) \;=\; O(\log n + k_i^\beta + k) \;=\; O(\log n + k).$$

THEOREM C.2. *Given n planes in $\mathbb{R}^3$, there exists a data structure of $O(n \log\log n)$ size, such that any "k lowest planes" query can be answered in $O(\log n + k)$ time deterministically.*

Consequences on halfspace range reporting, k nearest neighbors, and circular range reporting can be immediately derived, as in Corollaries 2.5, 2.6, and 2.7.

Remarks.

1. The preprocessing time is prohibitively large (though polynomial), so our randomized method is still more practical.

2. It remains an open problem to bring down the space complexity to linear while maintaining optimal query time. (Note that if the same k is used in all queries and is given in advance, then our approach achieves linear space.)

Update. Theorem C.2 has also been independently shown by Ramos [56] using ideas from the conference version of this paper, with a better expected preprocessing time of $O(n \log n)$. His proof is more complicated, though, and appears to be only of theoretical interest.

REFERENCES

[1] P. K. AGARWAL, B. ARONOV, T. M. CHAN, AND M. SHARIR, *On levels in arrangements of lines, segments, planes, and triangles*, Discrete Comput. Geom., 19 (1998), pp. 315–331.

[2] P. K. AGARWAL, M. DE BERG, J. MATOUŠEK, AND O. SCHWARZKOPF, *Constructing levels in arrangements and higher order Voronoi diagrams*, SIAM J. Comput., 27 (1998), pp. 654–667.

[3] P. K. Agarwal, A. Efrat, and M. Sharir, *Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications*, in Proceedings of the 11th ACM Symposium on Computational Geometry, Vancouver, Canada, 1995, pp. 39–50.
[4] P. K. Agarwal and J. Erickson, *Geometric range searching and its relatives*, in Advances in Discrete and Computational Geometry, B. Chazelle, J. E. Goodman, and R. Pollack, eds., AMS, Providence, RI, 1999, pp. 1–56.
[5] P. K. Agarwal and J. Matoušek, *Ray shooting and parametric search*, SIAM J. Comput., 22 (1993), pp. 794–806.
[6] P. K. Agarwal and J. Matoušek, *Dynamic half-space range reporting and its applications*, Algorithmica, 13 (1995), pp. 325–345.
[7] P. K. Agarwal, J. Matoušek, and O. Schwarzkopf, *Computing many faces in arrangements of lines and segments*, SIAM J. Comput., 27 (1998), pp. 491–505.
[8] P. K. Agarwal and M. Sharir, *Arrangements and their applications*, in Handbook of Computational Geometry, J. Urrutia and J. Sack, eds., North-Holland, Amsterdam, to appear.
[9] A. Aggarwal, L. J. Guibas, J. Saxe, and P. W. Shor, *A linear-time algorithm for computing the Voronoi diagram of a convex polygon*, Discrete Comput. Geom., 4 (1989), pp. 591–604.
[10] A. Aggarwal, M. Hansen, and T. Leighton, *Solving query-retrieval problems by compacting Voronoi diagrams*, in Proceedings of the 22nd ACM Symposium on the Theory of Computing, ACM, New York, 1990, pp. 331–340.
[11] A. Andrzejak and E. Welzl, *k-Sets and j-Facets: A Tour of Discrete Geometry*, 1997, manuscript.
[12] T. Asano and T. Tokuyama, *Topological walk revisited*, in Proceedings of the 6th Canadian Conference on Computational Geometry, Saskatoon, Saskatchewan, Canada, 1994, pp. 1–6.
[13] F. Aurenhammer, *Voronoi diagrams: A survey of a fundamental geometric data structure*, ACM Comput. Surv., 23 (1991), pp. 345–405.
[14] F. Aurenhammer and O. Schwarzkopf, *A simple on-line randomized incremental algorithm for computing higher order Voronoi diagrams*, Internat. J. Comput. Geom. Appl., 2 (1992), pp. 363–381.
[15] J.-D. Boissonnat, O. Devillers, and M. Teillaud, *A semidynamic construction of higher-order Voronoi diagrams and its randomized analysis*, Algorithmica, 9 (1993), pp. 329–356.
[16] J. L. Bentley and H. A. Maurer, *A note on Euclidean near neighbor searching in the plane*, Inform. Process. Lett., 8 (1979), pp. 133–136.
[17] I. Z. Emiris and J. F. Canny, *A general approach to removing degeneracies*, SIAM J. Comput., 24 (1995), pp. 650–664.
[18] T. M. Chan, *Output-sensitive results on convex hulls, extreme points, and related problems*, Discrete Comput. Geom., 16 (1996), pp. 369–387.
[19] T. M. Chan, *On enumerating and selecting distances*, in Proceedings of the 14th ACM Symposium on Computational Geometry, Minneapolis, MN, 1998, pp. 279–286.
[20] T. M. Chan, *Dynamic planar convex hull operations in near-logarithmic amortized time*, in Proceedings of the 40th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 92–99.
[21] T. M. Chan, *Remarks on k-Level Algorithms in the Plane*, 1999, manuscript.
[22] B. Chazelle, *On the convex layers of a planar set*, IEEE Trans. Inform. Theory, IT-31 (1985), pp. 509–517.
[23] B. Chazelle, *Filtering search: A new approach to query-answering*, SIAM J. Comput., 15 (1986), pp. 703–724.
[24] B. Chazelle, *An optimal convex hull algorithm in any fixed dimension*, Discrete Comput. Geom., 10 (1993), pp. 377–409.
[25] B. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap, *New upper bounds for neighbor searching*, Inform. and Control, 68 (1986), pp. 105–124.
[26] B. Chazelle and H. Edelsbrunner, *An improved algorithm for constructing kth-order Voronoi diagrams*, IEEE Trans. Comput., C-36 (1987), pp. 1349–1354.
[27] B. Chazelle and J. Friedman, *A deterministic view of random sampling and its use in geometry*, Combinatorica, 10 (1990), pp. 229–249.
[28] B. Chazelle, L. Guibas, and D. T. Lee, *The power of geometric duality*, BIT, 25 (1985), pp. 76–90.
[29] B. Chazelle and F. P. Preparata, *Halfspace range search: An algorithmic application of k-sets*, Discrete Comput. Geom., 1 (1986), pp. 3–93.
[30] K. L. Clarkson, *New applications of random sampling in computational geometry*, Discrete Comput. Geom., 2 (1987), pp. 195–222.
[31] K. L. Clarkson, *A randomized algorithm for closest-point queries*, SIAM J. Comput., 17 (1988), pp. 830–847.

[32] K. L. Clarkson and P. W. Shor, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.
[33] R. Cole, M. Sharir, and C. K. Yap, *On k-hulls and related problems*, SIAM J. Comput., 16 (1987), pp. 61–77.
[34] T. K. Dey, *Improved bounds on planar k-sets and k-levels*, Discrete Comput. Geom., 19 (1998), pp. 373–382.
[35] T. K. Dey and H. Edelsbrunner, *Counting triangle crossings and halving planes*, Discrete Comput. Geom., 12 (1994), pp. 281–289.
[36] H. Edelsbrunner, *Edge-skeletons in arrangements with applications*, Algorithmica, 1 (1986), pp. 93–109.
[37] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.
[38] H. Edelsbrunner and E. P. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms*, ACM Trans. on Graphics, 9 (1990), pp. 66–104.
[39] H. Edelsbrunner, J. O'Rourke, and R. Seidel, *Constructing arrangements of lines and hyperplanes with applications*, SIAM J. Comput., 15 (1986), pp. 341–363.
[40] H. Edelsbrunner and E. Welzl, *Constructing belts in two-dimensional arrangements with applications*, SIAM J. Comput., 15 (1986), pp. 271–284.
[41] P. Erdős, L. Lovász, A. Simmons, and E. Straus, *Dissection graphs of planar point sets*, in A Survey of Combinatorial Theory, J. N. Srivastava, ed., North-Holland, Amsterdam, 1973, pp. 139–154.
[42] H. Everett, J.-M. Robert, and M. van Kreveld, *An optimal algorithm for the* ($\leq k$)*-levels, with applications to separation and transversal problems*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 247–261.
[43] D. Halperin, *Arrangements*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press, Boca Raton, FL, 1997, pp. 389–412.
[44] S. Har-Peled, *Taking a walk in a planar arrangement*, in Proceedings of the 40th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 100–110.
[45] N. Katoh and K. Iwano, *Finding k farthest pairs and k closest/farthest bichromatic pairs for points in the plane*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 37–51.
[46] D. T. Lee, *On k-nearest neighbor Voronoi diagrams in the plane*, IEEE Trans. Comput., C-31 (1982), pp. 478–287.
[47] J. Matoušek, *Efficient partition trees*, Discrete Comput. Geom., 8 (1992), pp. 315–334.
[48] J. Matoušek, *Reporting points in halfspaces*, Comput. Geom., 2 (1992), pp. 169–186.
[49] J. Matoušek, *Geometric range searching*, ACM Comput. Surv., 26 (1994), pp. 421–461.
[50] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, 1995.
[51] K. Mulmuley, *Output sensitive construction of levels and Voronoi diagrams in* R^d *of order* 1 *to k*, in Proceedings of the 22nd ACM Symposium on the Theory of Computing, ACM, New York, 1990, pp. 322–330.
[52] K. Mulmuley, *On levels in arrangements and Voronoi diagrams*, Discrete Comput. Geom., 6 (1991), pp. 307–338.
[53] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice-Hall, Englewood Cliffs, NJ, 1994.
[54] F. P. Preparata and S. J. Hong, *Convex hulls of finite sets of points in two and three dimensions*, Commun. ACM, 20 (1977), pp. 87–93.
[55] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.
[56] E. Ramos, *On range reporting, ray shooting, and k-level construction*, in Proceedings of the 15th ACM Symposium on Computational Geometry, ACM, New York, 1999, pp. 390–399.
[57] M. I. Shamos and D. Hoey, *Closest-point problems*, in Proceedings of the 16th IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1977, pp. 151–162.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 576–601

A GENERALIZATION OF RESOURCE-BOUNDED MEASURE, WITH APPLICATION TO THE BPP VS. EXP PROBLEM*

HARRY BUHRMAN†, DIETER VAN MELKEBEEK‡, KENNETH W. REGAN§, D. SIVAKUMAR¶, AND MARTIN STRAUSS‖

Abstract. We introduce *resource-bounded betting games* and propose a generalization of Lutz's resource-bounded measure in which the choice of the next string to bet on is fully adaptive. Lutz's martingales are equivalent to betting games constrained to bet on strings in lexicographic order. We show that if strong pseudorandom number generators exist, then betting games are equivalent to martingales for measure on E and EXP. However, we construct betting games that succeed on certain classes whose Lutz measures are important open problems: the class of polynomial-time Turing-complete languages in EXP and its superclass of polynomial-time Turing-autoreducible languages. If an EXP-martingale succeeds on either of these classes, or if betting games have the "finite union property" possessed by Lutz's measure, one obtains the nonrelativizable consequence BPP $\neq$ EXP. We also show that if EXP $\neq$ MA, then the polynomial-time truth-table-autoreducible languages have Lutz measure zero, whereas if EXP = BPP, they have measure one.

Key words. computational complexity, theory of computation, probabilistic computation, complexity classes, resource-bounded measure, betting games, polynomial reductions, pseudorandom generators, sampling, autoreducibility

AMS subject classification. 68Q15

PII. S0097539798343891

1. Introduction.

Lutz's theory of measure on complexity classes is now usually defined in terms of resource-bounded martingales. A martingale can be regarded as a gambling game played on unseen languages A. Let $s_1, s_2, s_3, \ldots$ be the standard lexicographic ordering of strings. The gambler G starts with capital $C_0 = \$1$ and places a bet $B_1 \in [0, C_0]$ on either "$s_1 \in A$" or "$s_1 \notin A$." Given a fixed particular language A, the bet's outcome depends only on whether $s_1 \in A$. If the bet wins, then the new capital C_1 equals $C_0 + B_1$, while if the bet loses, $C_1 = C_0 - B_1$. The gambler then places a bet $B_2 \in [0, C_1]$ on (or against) membership of the string s_2, then on

*Received by the editors August 27, 1998; accepted for publication (in revised form) November 1, 1999; published electronically June 27, 2000. A preliminary version of this paper without proofs appeared in the Proceedings of the Symposium on Theoretical Aspects of Computer Science, Paris, France, 1998, under the title "A generalization of resource-bounded measure, with an application."
http://www.siam.org/journals/sicomp/30-2/34389.html

†CWI, Kruislaan 413, 1098SJ Amsterdam, The Netherlands (burhman@cwi.nl). This author was partially supported by the Dutch foundation for scientific research (NWO) through SION project 612-34-002, by the European Union through NeuroCOLT ESPRIT Working Group 8556, and by HC&M grant ERB4050PL93-0516.

‡DIMACS Center, Rutgers University, 96 Frelinghuysen Road, Piscataway, NJ 08854-8018 (dieter@dimacs.rutgers.edu). This author was partly supported by the European Union through Marie Curie Research Training grant ERB-4001-GT-96-0783 at CWI and at the University of Amsterdam; by NSF grant CCR 92-53582; and by the Fields Institute of the University of Toronto; research performed mainly at the University of Chicago.

§Department of Computer Science and Engineering, State University of NY at Buffalo, 226 Bell Hall, Buffalo, NY 14260-2000 (regan@cse.buffalo.edu). The research of this author was supported in part by the National Science Foundation under grant CCR-9409104.

¶IBM Almaden Research Center, Department K-53, 650 Harry Road San Jose, CA 95120 (siva@almaden.ibm.com). Part of this research was performed while at the State University of N.Y. at Buffalo, supported in part by National Science Foundation grant CCR-9409104.

‖AT&T Labs, 180 Park Ave, Florham Park, NJ 07932-0971 (mstrauss@research.att.com). This research was performed while the author was at Rutgers University and Iowa State University, supported by NSF grants CCR-9204874 and CCR-9157382.

s_3, and so forth. The gambler *succeeds* if G's capital C_i grows toward $+\infty$. The class $\mathcal{C}$ of languages A on which G succeeds (and any subclass) is said to have *measure zero*. One also says G *covers* $\mathcal{C}$. Lutz and others (see [Lutz97]) have developed a rich and extensive theory around this measure-zero notion and have shown interesting connections to many other important problems in complexity theory.

We propose the generalization obtained by lifting the requirement that G must bet on strings in lexicographic order. That is, G may begin by choosing any string x_1 on which to place its first bet and, after the oracle tells the result, may choose any other string x_2 for its second bet, and so forth. Note that the sequences $x_1, x_2, x_3, \ldots$ (as well as $B_1, B_2, B_3, \ldots$) may be radically different for different oracle languages A—in complexity-theory parlance, G's queries are *adaptive*. The lone restriction is that G may not query (or bet on) the same string twice. We call G a *betting game*.

Our betting games remedy a possible lack in the martingale theory, one best explained in the context of languages that are "random" for classes $\mathcal{D}$ such as E or EXP. In this paper, E stands for deterministic time $2^{O(n)}$, and EXP stands for deterministic time $2^{n^{O(1)}}$. A language L is $\mathcal{D}$-*random* if L cannot be covered by a $\mathcal{D}$-martingale. Based on one's intuition about random 0-1 sequences, the language $L' = \{\, \mathit{flip}(x) : x \in L \,\}$ should likewise be $\mathcal{D}$-random, where $\mathit{flip}(x)$ changes every 0 in x to a 1 and vice-versa. However, this closure property is not known for E-random or EXP-random languages, because of the way martingales are tied to the fixed lexicographic ordering of Σ^*. Betting games can adapt to easy permutations of Σ^* such as that induced by *flip*. Similarly, a class $\mathcal{C}$ that is *small* in the sense of being covered by a ($\mathcal{D}$-) betting game remains small if the languages $L \in \mathcal{C}$ are so permuted. In the r.e./recursive theory of random languages, our generalization is similar to "Kolmogorov–Loveland place-selection rules" (see [Lov69]). We make this theory work for complexity classes via a novel definition of "running in time $t(n)$" for an infinite process.

Our new angle on measure theory may be useful for attacking the problem of separating BPP from EXP, which has recently gained prominence in [ImWi98]. In Lutz's theory it is open whether the class of EXP-complete sets—under polynomial-time Turing reductions—has EXP-measure zero. If so (in fact if this set does not have measure one), then by results of Allender and Strauss [AlSt94], BPP $\neq$ EXP. Since there are oracles A such that $\mathrm{BPP}^A = \mathrm{EXP}^A$ [Hel86], this kind of absolute separation would be a major breakthrough. We show that the EXP-complete sets *can* be covered by an EXP-betting game—in fact, by an E-betting game. The one technical lack in our theory as a notion of measure is also interesting here: If the "finite unions" property holds for betting games (viz., $\mathcal{C}_1$ small $\wedge$ $\mathcal{C}_2$ small $\Longrightarrow$ $\mathcal{C}_1 \cup \mathcal{C}_2$ small), then EXP $\neq$ BPP. Likewise, if Lutz's martingales do enjoy the permutation-invariance of betting games, then BPP $\neq$ EXP. Finally, we show that if a pseudorandom number generator of security $2^{n^{\Omega(1)}}$ exists, then for every EXP-betting game G one can find an EXP-martingale that succeeds on all sets covered by G. Pseudorandom generators of higher security $2^{\Omega(n)}$ likewise imply the equivalence of E-betting games and E-measure. Ambos-Spies, Lempp, and Mainhardt [ALM98] proved that the EXP-complete sets have E-measure zero under a different hypothesis, namely P = PSPACE.

Measure theory and betting games help us to dig further into questions about pseudorandom generators and complexity-class separations. Our tool is the notion of an *autoreducible* set, whose importance in complexity theory was argued by Buhrman, Fortnow, van Melkebeek, and Torenvliet [BFvMT98] (after [BFT95]). A

language L is $\leq_T^p$-*autoreducible* if there is a polynomial-time oracle Turing machine Q such that for all inputs x, Q^L correctly decides whether $x \in L$ without ever submitting x itself as a query to L. If Q is nonadaptive (i.e., computes a polynomial-time truth-table reduction), we say L is $\leq_{tt}^p$-*autoreducible*. We show that the class of $\leq_T^p$-autoreducible sets is covered by an E-betting game. Since every EXP-complete set is $\leq_T^p$-autoreducible [BFvMT98], this implies results given above. The subclass of $\leq_{tt}^p$-autoreducible sets provides the following tighter connection between measure statements and open problems about EXP.

- If the $\leq_{tt}^p$-autoreducible sets do not have E-measure zero, then EXP = MA.
- If the $\leq_{tt}^p$-autoreducible sets do not have E-measure one in EXP, then EXP $\neq$ BPP.

Here MA is Babai's "Merlin-Arthur" class, which contains BPP and NP and is contained in the level $\Sigma_2^p \cap \Pi_2^p$ of the polynomial hierarchy [Bab85, BaMo88]. Since EXP $\neq$ MA is strongly believed, one would expect the class of $\leq_{tt}^p$-autoreducible sets to have E-measure zero, but *proving* this—or proving any of the dozen other measure statements in Corollaries 6.2 and 6.5—would yield a proof of EXP $\neq$ BPP.

In sum, the whole theory of resource-bounded measure has progressed far enough to wind the issues of (pseudo)randomness and stochasticity within exponential time very tightly. We turn the wheels a few more notches and seek greater understanding of complexity classes in the places where the boundary between "measure one" and "measure zero" seems tightest.

Section 2 reviews the formal definitions of Lutz's measure and martingales. Section 3 introduces betting games and shows that they are a generalization of martingales. Section 4 shows how to simulate a betting game by a martingale of perhaps unavoidably higher time complexity. Section 5, however, demonstrates that strong pseudorandom generators (if there are any) allow one to compute the martingale in the same order of time. Section 6 presents our main results pertaining to autoreducible sets, including our main motivating example of a concrete betting game. The concluding section 7 summarizes open problems and gives prospects for future research.

2. Martingales. A *martingale* is abstractly defined as a function d from $\{0,1\}^*$ into the nonnegative reals that satisfies the following "average law": for all $w \in \{0,1\}^*$,

$$d(w) = \frac{d(w0) + d(w1)}{2}. \tag{1}$$

The interpretation in Lutz's theory is that a string $w \in \{0,1\}^*$ stands for an initial segment of a language over an arbitrary alphabet Σ as follows: Let $s_1, s_2, s_3, \ldots$ be the standard lexicographic ordering of Σ^*. Then for any language $A \subseteq \Sigma^*$, write $w \sqsubseteq A$ if for all i, $1 \leq i \leq |w|$, $s_i \in A$ iff the ith bit of w is a 1. We also regard w as a function with *domain* $dom(w) = \{s_1, \ldots, s_{|w|}\}$ and range $\{0,1\}$, writing $w(s_i)$ for the ith bit of w. A martingale d *succeeds on* a language A if the sequence of values $d(w)$ for $w \sqsubseteq A$ is unbounded.

Let $S^\infty[d]$ stand for the (possibly empty, often uncountable) class of languages on which d succeeds. Lutz originally defined the complexity of a martingale d in terms of computing fast-converging rational approximations to d. Subsequently he showed that for certain classes of time bounds one loses no generality by requiring that martingales themselves have rational values a/b such that all digits of the integers a and b (not necessarily in lowest terms) are output within the time bound. That

is, given any martingale d meeting the original definition of computability within the time bound, one can obtain a rational-valued d' computable within that bound such that $S^\infty[d] \subseteq S^\infty[d']$ [May94t, JuLu95]. We adopt this requirement throughout the paper and specify that integers are represented in standard binary notation and rationals as pairs of integers, not necessarily in lowest terms. We use the fact that a sum $a_1/b_1 + \cdots a_m/b_m$ can be computed and written down in $\ell^{O(1)}$ time, where ℓ is the sum of the lengths of the integers a_i and b_i.

DEFINITION 2.1 (cf. [Lutz92, May94t]). *Let Δ be a complexity class of functions. A class $\mathcal{C}$ of languages* has Δ-measure zero*, written $\mu_\Delta(\mathcal{C}) = 0$, if there is a martingale d computable in Δ such that $\mathcal{C} \subseteq S^\infty[d]$. One also says that d* covers *$\mathcal{C}$.*

Lutz measured the time to compute $d(w)$ in terms of the length N of w, but one can also work in terms of the largest length n of a string in the domain of w. For $N > 0$, n equals $\lfloor \log_2 N \rfloor$; all we care about is that $n = \Theta(\log N)$ and $N = 2^{\Theta(n)}$. Because complexity bounds on languages we want to analyze will naturally be stated in terms of n, we prefer to use n for martingale complexity bounds. The following correspondence is helpful:

$$\begin{array}{lcrcrcl} \text{Lutz's } \text{“}p\text{”} & \sim & N^{O(1)} & = & 2^{O(n)} & \sim & \text{measure on E,} \\ \text{Lutz's } \text{“}p_2\text{”} & \sim & 2^{(\log N)^{O(1)}} & = & 2^{n^{O(1)}} & \sim & \text{measure on EXP.} \end{array}$$

Since we measure the time to compute $d(w)$ in terms of n, we write "μ_{E}" for E-measure, "μ_{EXP}" for EXP-measure, and generally μ_Δ for any Δ that names both a language and function class. Abusing notation similarly, we define the following.

DEFINITION 2.2 (see [Lutz92]). *A class $\mathcal{C}$ has Δ-measure one, written $\mu_\Delta(\mathcal{C}) = 1$, if $\mu_\Delta(\Delta \setminus \mathcal{C}) = 0$.*

The concept of resource-bounded measure is known to be robust under several changes [May94t]. The following lemma has appeared in various forms [May94t, BuLo00]. It essentially says that we can assume a martingale grows almost monotonically (sure winnings) and not too fast (slow winnings).

LEMMA 2.3 ("slow-but-sure-winnings" lemma for martingales). *Let d be a martingale. Then there is a martingale d' with $S^\infty[d] \subseteq S^\infty[d']$ such that*

$$(\forall w)(\forall u) : d'(wu) > d'(w) - 2d(\lambda), \quad \text{and} \tag{2}$$

$$(\forall w) : d'(w) < 2(|w| + 1)d(\lambda). \tag{3}$$

If d is computable in time $t(n)$, then d' is computable in time $(2^n t(n))^{O(1)}$.

The idea is to play the strategy of d but in a more conservative way. Say we start with an initial capital of \$1. We will deposit a part c of our capital in a bank and only play the strategy underlying d on the remaining liquid part e of our capital. We start with no savings and a liquid capital of \$1. If our liquid capital reaches or exceeds \$2, we deposit an additional \$1 or \$2 to our savings account c so as to keep the liquid capital in the range $\$[1, 2)$ at all times. If d succeeds, it will push the liquid capital infinitely often to \$2 or above, so c grows to infinity, and d' succeeds too. Since we never take money out of our savings account c and the liquid capital e is bounded by \$2, once our total capital $d' = c + e$ has reached a certain level, it will never go more than \$2 below that level anymore, no matter how bad the strategy underlying d is. On the other hand, since we add at most \$2 to c in each step, $d'(w)$ cannot exceed $2(|w| + 1)$ either.

We now give the formal proof.

Proof of Lemma 2.3. Define $d' : \Sigma^* \to [0, \infty)$ by

$$d'(w) = (c(w) + e(w))d(\lambda),$$

where $c(\lambda) = 0$ and $e(\lambda) = 1$, and

$$\begin{array}{llll}
c(wb) = c(w) & \text{and} & e(wb) = e(w) & \text{if } d(w) = 0; \text{ else:} \\
c(wb) = c(w) + 2 & \text{and} & e(wb) = \frac{d(wb)}{d(w)} e(w) - 2 & \text{if } \frac{d(wb)}{d(w)} e(w) \geq 3; \\
c(wb) = c(w) + 1 & \text{and} & e(wb) = \frac{d(wb)}{d(w)} e(w) - 1 & \text{if } 2 \leq \frac{d(wb)}{d(w)} e(w) < 3; \\
c(wb) = c(w) & \text{and} & e(wb) = \frac{d(wb)}{d(w)} e(w) & \text{if } \frac{d(wb)}{d(w)} e(w) < 2.
\end{array}$$

To see that the recursion does not excessively blow up the time complexity or size of the answer, note that owing to cancellation of values of d, every value $e(w)$ where $d(w) \neq 0$ is given by a sum of the form

$$\sum_{k=0}^{N} a_k \frac{d(w)}{d(w[1 \ldots k])},$$

where each a_k is in $\{ -2, -1, 0, 1 \}$, $N = |w|$, and $w[1 \ldots k]$ stands for the first k bits of w. Each term in the sum is computable in time $O(t(n)^2 N)$ (using the naive quadratic algorithms for multiplication and integer division). Then by the property noted just before Definition 2.1, these terms can be summed in time $(Nt(n))^{O(1)}$.

By induction on $|w|$ we observe that

$$0 \leq e(w) < 2, \tag{4}$$

and that

$$d'(wb) = \begin{cases} \left[c(w) + \frac{d(wb)}{d(w)} e(w) \right] d(\lambda) & \text{if } d(w) \neq 0, \\ d'(w) & \text{otherwise,} \end{cases}$$

from which it follows that d' is a martingale.

Now let ω be an infinite 0-1 sequence denoting a language on which d succeeds. Then $e(w)$ will always remain positive for $w \sqsubseteq \omega$, and $\frac{d(wb)}{d(w)} e(w)$ will become 2 or more infinitely often. Consequently, $\lim_{w \sqsubseteq \omega, |w| \to \infty} c(w) = \infty$. Since $d'(w) \geq c(w) d(\lambda)$, it follows that $S^\infty[d] \subseteq S^\infty[d']$. Moreover, by (4) and the fact that c does not decrease along any sequence, we have that

$$d'(wu) \geq c(wu) d(\lambda) \geq c(w) d(\lambda) = d'(w) - e(w) d(\lambda) > d'(w) - 2d(\lambda).$$

Since c can increase by at most 2 in every step, $c(w) \leq 2|w|$. Together with (4), this yields that

$$d'(w) = (c(w) + e(w)) d(\lambda) < 2(|w| + 1) d(\lambda). \qquad \square$$

One can also show that $S^\infty[d'] \subseteq S^\infty[d]$ in Lemma 2.3, so the success set actually remains intact under the above transformation.

As with Lebesgue measure, the property of having resource-bounded measure zero is monotone and closed under union ("finite unions property"). A resource-bounded version of closure under countable unions also holds. The property that becomes crucial in resource-bounded measure is that the whole space Δ does not have measure zero, which Lutz calls the "measure conservation" property. With a slight abuse of meaning for "$\neq$," this property is written $\mu_\Delta(\Delta) \neq 0$. In particular, $\mu_{\mathrm{E}}(\mathrm{E}) \neq 0$ and

$\mu_{\mathrm{EXP}}(\mathrm{EXP}) \neq 0$. Subclasses of Δ that require substantially fewer resources do have Δ-measure zero. For example, P has E-measure zero. Indeed, for any fixed $c > 0$, DTIME$[2^{cn}]$ has E-measure zero and DTIME$[2^{n^c}]$ has EXP-measure zero [Lutz92].

Apart from formalizing rareness and abundance in computational complexity theory, resource-bounded martingales are also used to define the concept of a random set in a resource-bounded setting.

DEFINITION 2.4. *A set A is* Δ-random *if* $\mu_\Delta(\{A\}) \neq 0$.

In other words, A is Δ-random if no Δ-martingale succeeds on A.

3. Betting games. To capture intuitions that have been expressed not only for Lutz measure but also in many earlier papers on random sequences, we formalize a betting game as an *infinite* process, rather than as a Turing machine that has *finite* computations on string inputs.

DEFINITION 3.1. *A* betting game *G is an oracle Turing machine that maintains a "capital tape" and a "bet tape," in addition to its standard query tape and worktapes, and works in* stages $i = 1, 2, 3 \ldots$ *as follows. Beginning each stage i, the capital tape holds a nonnegative rational number C_{i-1}. The* initial capital C_0 *is some positive rational number. G computes a query string x_i to bet on, a* bet amount B_i, $0 \leq B_i \leq C_{i-1}$, *and a* bet sign $b_i \in \{-1, +1\}$. *The computation is* legal *so long as x_i does not belong to the set $\{x_1, \ldots, x_{i-1}\}$ of strings queried in earlier stages. G ends stage i by entering a special query state. For a given oracle language A, if $x_i \in A$ and $b_i = +1$, or if $x_i \notin A$ and $b_i = -1$, then the new capital is given by $C_i := C_{i-1} + B_i$, else by $C_i := C_{i-1} - B_i$. We charge M for the time required to write the numerator and denominator of the new capital C_i down. The query and bet tapes are blanked, and G proceeds to stage $i + 1$.*

In this paper, we lose no generality by not allowing G to "crash" or to loop without writing a next bet and query. Note that every oracle set A determines a unique infinite computation of G, which we denote by G^A. This includes a unique infinite sequence $x_1, x_2, \ldots$ of query strings, and a unique sequence $C_0, C_1, C_2, \ldots$ telling how the gambler fares against A .

DEFINITION 3.2. *A betting machine G* runs in time $t(n)$ *if for all oracles A, every query of length n made by G^A is made in the first $t(n)$ steps of the computation.*

DEFINITION 3.3. *A betting game G* succeeds *on a language A, written $A \in S^\infty[G]$, if the sequence of values C_i in the computation G^A is unbounded. If $A \in S^\infty[G]$, then we also say G* covers *A.*

Our main motivating example where one may wish not to bet in lexicographic order, or according to any fixed ordering of strings, is deferred to section 6. There we will construct an E-betting game that succeeds on the class of $\leq_T^p$-autoreducible languages, which is not known to have Lutz measure zero in E or EXP.

We now want to argue that the more liberal requirement of being covered by a time $t(n)$ betting game still defines a smallness concept for subclasses of DTIME$[t(n)]$ in the intuitive sense Lutz established for his measure-zero notion. The following result is a good beginning.

THEOREM 3.4. *For every time-$t(n)$ betting game G, we can construct a language in* DTIME$[t(n)]$ *that is not covered by G.*

Proof. Let Q be a nonoracle Turing machine that runs as follows on any input x. The machine Q simulates up to $t(|x|)$ steps of the single computation of G on empty input. Whenever G bets on and queries a string y, Q gives the answer that causes G to lose money, rejecting in the case of a zero bet. If and when G queries x, Q does likewise. If $t(|x|)$ steps go by without x being queried, then Q rejects x.

The important point is that Q's answer to a query $y \neq x$ is the same as the answer when Q is run on input y. The condition that G cannot query a string x of length n after $t(n)$ steps have elapsed ensures that the decision made by Q when x is not queried does not affect anything else. Hence Q defines a language on which G never does better than its initial capital C_0 and so does not succeed. □

In particular, the class E cannot be covered by an E-betting game, nor EXP by an EXP-betting game. Put another way, the "measure conservation axiom" [Lutz92] of Lutz's measure carries over to betting games.

To really satisfy the intuition of "small," however, it should hold that the union of two small classes is small. (Moreover, "easy" countable unions of small classes should be small, as in [Lutz92].) Our lack of meeting this "finite union axiom" will later be excused insofar as it has the nonrelativizing consequence BPP $\neq$ EXP. Theorem 3.4 is still good enough for the "measure-like" results in this paper.

We note also that several robustness properties of Lutz's measure treated in section 2 carry over to betting games. This is because we can apply the underlying transformations to the *capital function* c_G of G, which is defined as follows.

DEFINITION 3.5. *Let G be a betting game, and let $i \geq 0$ be an integer.*

(a) *A* play α *of length i is a sequence of i-many oracle answers. Note that α determines the first i-many stages of G, together with the query and bet for the next stage.*

(b) *$c_G(\alpha)$ is the capital C_i that G has at the end of the play α (before the next query).*

Note that the function c_G is a martingale over plays α. The proof of Lemma 2.3 works for c_G. We obtain the following lemma.

LEMMA 3.6 ("slow-but-sure winnings" lemma for betting games). *Let G be a betting game that runs in time $t(n)$. Then we can construct a betting game G' that runs in time $(2^n t(n))^{O(1)}$ such that $S^\infty[G] \subseteq S^\infty[G']$, G' always makes the same queries in the same order as G, and*

$$\forall\, \beta, \forall\, \gamma : c_{G'}(\beta\gamma) > c_{G'}(\beta) - 2c_G(\lambda), \tag{5}$$

$$\forall\, \alpha : c_{G'}(\alpha) < 2(|\alpha| + 1)c_G(\lambda). \tag{6}$$

Proof. The proof of Lemma 2.3 carries over. □

To begin comparing betting games and martingales, we note first that the latter can be considered a direct special case of betting games. Say a betting game G is *lex-limited* if for all oracles A, the sequence $x_1, x_2, x_3 \ldots$ of queries made by G^A is in lexicographic order. (It need not equal the lexicographic enumeration $s_1, s_2, s_3, \ldots$ of Σ^*.)

THEOREM 3.7. *Let $\mathcal{T}(n)$ be a collection of time bounds that is closed under squaring and under multiplication by 2^n, such as $2^{O(n)}$ or $2^{n^{O(1)}}$. Then a class $\mathcal{C}$ has time-$\mathcal{T}(n)$ measure zero iff $\mathcal{C}$ is covered by a time-$\mathcal{T}(n)$ lex-limited betting game.*

Proof. From a martingale d to a betting game G, each stage i of G^A bets on s_i an amount B_i with sign $b_i \in \{-1, +1\}$ given by $b_i B_i = d(w1) - d(w)$, where w is the first $i-1$ bits of the characteristic sequence of A. This takes $O(2^n)$ evaluations of d to run G up through queries of length n, hence the hypothesis on the time bounds $\mathcal{T}(n)$. In the other direction, when G is lex-limited, one can simulate G on a finite initial segment w of its oracle up to a stage where all queries have been answered by w and G will make no further queries in the domain of w. One can then define $d(w)$ to be the capital entering this stage. That this is a martingale and fulfills the success and run-time requirements is left to the reader. □

Hence in particular for measure on E and EXP, martingales are equivalent to betting games constrained to bet in lexicographic order. Now we will see how we can transform a *general* betting game into an equivalent martingale.

4. From betting games to martingales. This section associates to every betting game G a martingale d_G such that $S^\infty[G] \subseteq S^\infty[d_G]$, and begins examining the complexity of d_G. Before defining d_G, however, we pause to discuss some tricky subtleties of betting games and their computations.

Given a finite initial segment w of an oracle language A, one can define the partial computation G^w of the betting game up to the stage i at which it first makes a query x_i that is not in the domain of w. Define $d(w)$ to be the capital C_{i-1} that G had entering this stage. It is tempting to think that d is a martingale and succeeds on all A for which G succeeds—but neither statement is true in general. The most important reason is that d may fail to be a martingale.

To see this, suppose x_i itself is the lexicographically least string not in the domain of w. That is, x_i is indexed by the bit b of wb, and $w1 \sqsubseteq A$ iff $x_i \in A$. It is possible that G^A makes a small (or even zero) bet on x_i, *and then goes back to make more bets in the domain of* w, *winning lots of money on them*. The definitions of both $d(w0)$ and $d(w1)$ will then reflect these added winnings, and both values will be greater than $d(w)$. For example, suppose G^A first puts a zero bet on $x_i = s_j$, then bets all of its money on $x_{i+1} = s_{j-1}$ not being in A, and then proceeds with $x_{i+2} = s_{j+1}$. If $w(s_{j-1}) = 0$, then $d(w0) = d(w1) = 2d(w)$.

Put another way, a finite initial segment w may carry much more "winnings potential" than the above definition of $d(w)$ reflects. To capture this potential, one needs to consider potential plays of the betting game outside the domain of w. Happily, one can bound the length of the considered plays via the running time function t of G. Let n be the maximum length of a string indexed by w; i.e., $n = \lfloor \log_2(|w|) \rfloor$. Then after $t(n)$ steps, G cannot query any more strings in the domain of w, so w's potential is exhausted. We will define $d_G(w)$ as an *average* value of those plays that can happen, given the query answers fixed by w. We use the following definitions and notation.

DEFINITION 4.1. *For any $t(n)$ time-bounded betting game G and string $w \in \Sigma^*$, define the following.*

(a) *A play α is* t-maximal *if G completes the first $|\alpha|$ stages, but* not *the query and bet of the next stage, within t steps.*

(b) *A play α is* G-consistent with w, *written $\alpha \sim_G w$, if for all stages j such that the queried string x_j is in the domain of w, $\alpha_j = w(x_j)$. That is, α is a play that could possibly happen given the information in w. Also let $m(\alpha, w)$ stand for the number of such stages j whose query is answered by w.*

(c) *Finally, put $d_G(\lambda) = c_G(\lambda)$, and for nonempty w, with $n = \lfloor \log_2(|w|) \rfloor$ as above, let*

$$d_G(w) = \sum_{\alpha \ t(n)-maximal, \alpha \sim_G w} c_G(\alpha)\, 2^{m(\alpha,w)-|\alpha|} \quad . \tag{7}$$

The weight $2^{m(\alpha,w)-|\alpha|}$ in (7) has the following meaning. Suppose we extend the simulation of G^w by flipping a coin for every query outside the domain of w for exactly i stages. Then the number of coin-flips in the resulting play α of length i is $i - m(\alpha, w)$, so $2^{m(\alpha,w)-i}$ is its probability. Thus $d_G(w)$ returns the suitably-weighted average of $t(n)$-step computations of G with w fixed. The interested reader may verify that this is the same as averaging $d(wv)$ over all v of length $2^{t(n)}$ (or any fixed longer length), where d is the nonmartingale defined at the beginning of this section.

LEMMA 4.2. *The function $d_G(w)$ is a martingale.*

Proof. First we argue that

$$d_G(w) = \sum_{|\alpha'|=t(n),\alpha' \sim_G w} c_G(\alpha')\, 2^{m(\alpha',w)-t(n)}. \tag{8}$$

Observe that when $\alpha' = \alpha\beta$ and α is $t(n)$-maximal, $\alpha \sim_G w \iff \alpha' \sim_G w$. This is because none of the queries answered by β can be in the domain of w, else the definition of G running in time $t(n)$ would be violated. Likewise if $\alpha \sim_G w$, then $m(\alpha', w) = m(\alpha, w)$. Finally, since c_G is a martingale, $c_G(\alpha) = \sum_{|\beta|=t(n)-|\alpha|} c_G(\alpha\beta)\, 2^{|\alpha|-t(n)}$. These facts combine to show the equality of (7) and (8).

By the same argument, the right-hand side of (8) is unchanged on replacing "$t(n)$" by any $t' > t(n)$.

Now consider w such that $|w|+1$ is not a power of 2. Then the "n" for $w0$ and $w1$ is the same as the "n" for $d_G(w)$. Let P_0 stand for the set of α of length $t(n)$ that are G-consistent with $w0$ but not with $w1$, P_1 for those that are G-consistent with $w1$ but not $w0$, and P for those that are consistent with both. Then the set $\{\, \alpha : |\alpha| = t(n),\ \alpha \sim_G w \,\}$ equals the disjoint union of P, P_0, and P_1. Furthermore, for $\alpha \in P_0$ we have $m(\alpha, w0) = m(\alpha, w) + 1$, and similarly for P_1, while for $\alpha \in P$ we have $m(\alpha, w0) = m(\alpha, w1) = m(\alpha, w)$. Hence $d_G(w0) + d_G(w1)$ is given by

$$\sum_{\alpha \in P \cup P_0} c_G(\alpha) 2^{m(\alpha,w0)-t(n)} + \sum_{\alpha \in P \cup P_1} c_G(\alpha) 2^{m(\alpha,w1)-t(n)}$$

$$= \sum_{\alpha \in P_0} c_G(\alpha) 2^{m(\alpha,w0)-t(n)} + \sum_{\alpha \in P_1} c_G(\alpha) 2^{m(\alpha,w1)-t(n)} + 2 \sum_{\alpha \in P} c_G(\alpha) 2^{m(\alpha,w)-t(n)}$$

$$= 2 \sum_{\alpha \in P_0} c_G(\alpha) 2^{m(\alpha,w)-t(n)} + 2 \sum_{\alpha \in P_1} c_G(\alpha) 2^{m(\alpha,w)-t(n)} + 2 \sum_{\alpha \in P} c_G(\alpha) 2^{m(\alpha,w)-t(n)}$$

$$= 2 d_G(w).$$

Finally, if $|w|+1$ is a power of 2, then $d_G(w0)$ and $d_G(w1)$ use $t' := t(n+1)$ for their length of α. However, by the first part of this proof, we can replace $t(n)$ by t' in the definition of $d_G(w)$ without changing its value, and then the second part goes through the same way for t'. Hence d_G is a martingale. ∎

It is still the case, however, that d_G may not succeed on the languages on which the betting game G succeeds. To ensure this, we first use Lemma 3.6 to place betting games G into a suitable "normal form" satisfying the sure-winnings condition (5).

LEMMA 4.3. *If G is a betting game satisfying the sure-winnings condition* (5)*, then $S^\infty[G] \subseteq S^\infty[d_G]$.*

Proof. First, let $A \in S^\infty[G]$, and fix $k > 0$. Find a finite initial segment $w \sqsubseteq A$ long enough to answer every query made in a play α of G such that $\alpha \sim_G w$ and $c_G(\alpha) \geq k+2$ and long enough to make $t(n)$ in the definition of $d_G(w)$ (see (7)) greater than $|\alpha|$. Then every α' of length $t(n)$ such that $\alpha' \sim_G w$ has the form $\alpha' = \alpha\beta$. The sure-winnings condition (5) implies that the right-hand side of (7) defining $d_G(w)$ is an average over terms that all have size at least k. Hence $d_G(w) \geq k$. Letting k grow to infinity gives $A \in S^\infty[d_G]$. ∎

Now we turn our attention to the complexity of d_G. If G is a time-$t(n)$ betting game, it is clear that d_G can be computed deterministically in $O(t(n))$ *space*, because we need only cycle through all α of length $t(n)$, and all the items in (7) are computable in space $O(t(n))$. In particular, every E-betting game can be simulated by a martingale whose values are computable in deterministic space $2^{O(n)}$ (even counting the output against the space bound), and every EXP-betting game can be simulated by a martingale similarly computed in space $2^{n^{O(1)}}$. However, we show in the next section that one can *estimate* $d_G(w)$ well without having to cycle through all the α, using a pseudorandom generator to "sample" only a very small fraction of them.

5. Sampling results. First we determine the accuracy to which we need to estimate the values $d(w)$ of a hard-to-compute martingale. We state a stronger version of the result than we need in this section. In the next section, we will apply it to martingales whose "activity" is restricted to subsets J of $\{0,1\}^*$ in the following sense: for all strings $x \notin J$, and all w such that $s_{|w|+1} = x$, $d(w0) = d(w1) = d(w)$. Intuitively, a martingale d is inactive on a string x if there is *no* possible "past history" w that causes a nonzero bet to be made on x. For short we say that such a d is *inactive outside* J. Recall that $N = \Theta(2^n)$.

LEMMA 5.1. *Let d be a martingale that is inactive outside $J \subseteq \{0,1\}^*$, and let $[\epsilon(i)]_{i=0}^{\infty}$ be a nonnegative sequence such that $\sum_{s_i \in J} \epsilon(i)$ converges to a number K. Suppose we can compute in time $t(n)$ a function $g(w)$ such that $|g(w) - d(w)| \leq \epsilon(N)$ for all w of length N. Then there is a martingale d' computable in time $(2^n t(n))^{O(1)}$ such that for all w, $|d'(w) - d(w)| \leq 4K + 2\epsilon(0)$.*

In this section, we will apply Lemma 5.1 with $J = \{0,1\}^*$ and $\epsilon(N) = 1/N^2 = 1/2^{2n}$. In section 6.3 we will apply Lemma 5.1 in cases where J is finite.

Proof. First note that for any w (with $N = |w|$),

$$\begin{aligned} \left| g(w) - \frac{g(w0)+g(w1)}{2} \right| &\leq |g(w)-d(w)| + \left| \frac{d(w0)-g(w0)}{2} \right| + \left| \frac{d(w1)-g(w1)}{2} \right| \\ &\leq \epsilon(N) + \epsilon(N+1). \end{aligned} \tag{9}$$

In case $J = \{0,1\}^*$, we inductively define

$$\begin{cases} d'(\lambda) &= g(\lambda) + 2K + \epsilon(0), \\ d'(wb) &= d'(w) + g(wb) - \frac{g(w0)+g(w1)}{2}. \end{cases}$$

Note that d' satisfies the average law (1), and that we can compute $d'(w)$ in time $O(2^n t(n))$.

By induction on $|w|$, we can show using the estimate provided by (9) that

$$g(w) + \epsilon(N) + 2\sum_{i=N+1}^{\infty} \epsilon(i) \;\leq\; d'(w) \;\leq\; g(w) + 2\sum_{i=0}^{N-1} \epsilon(i) + \epsilon(N) + 2K.$$

It follows that

$$\begin{aligned} d'(w) &\geq g(w) + \epsilon(N) \\ &= d(w) + (g(w) - d(w)) + \epsilon(N) \geq d(w), \end{aligned}$$

and that

$$d'(w) = d(w) + (g(w) - d(w)) + (d'(w) - g(w))$$

$$\leq d(w) + \epsilon(N) + 2\sum_{i=0}^{N-1} \epsilon(i) + \epsilon(N) + 2K$$
$$\leq d(w) + 4K + 2\epsilon(0).$$

This establishes the lemma in case $J = \{0,1\}^*$. The generalization to other subsets J of $\{0,1\}^*$ is left to the reader. □

Next, we specify precisely which function f_G we will sample in order to estimate d_G and how we will do it.

Let G be a $t(n)$ time-bounded betting game. Consider a prefix w, and let n denote the largest length of a string in the domain of w. With any string ρ of length $t(n)$, we can associate a unique "play of the game" G defined by using w to answer queries in the domain of w and the successive bits of ρ to answer queries outside it. We can stop this play after $t(n)$ steps—so that the stopped play is a $t(n)$-maximal α—and then define $f_G(w,\rho)$ to be the capital $c_G(\alpha)$. Note that we can compute $f_G(w,\rho)$ in linear time, i.e., in time $O(|w| + t(n))$. The proportion of strings ρ of length $t(n)$ that map to the same play α is exactly the weight $2^{m(\alpha,w)-|\alpha|}$ in (7) for $d_G(w)$. Letting E stand for mathematical expectation, this gives us

$$d_G(w) = E_{|\rho|=t(n)}[f_G(w,\rho)].$$

To obtain good and efficient approximations to the right-hand side, we employ *pseudorandom generators*. The following supplies all relevant definitional background.

DEFINITION 5.2 (see [NiWi94]).

(a) *The* hardness $H_A(n)$ *of a set* A *at length* n *is the largest integer* s *such that for any circuit* C *of size at most* s *with* n *inputs,*

$$\left|\Pr_x[C(x) = A(x)] - \frac{1}{2}\right| \leq \frac{1}{s},$$

where x *is uniformly distributed over* Σ^n.

(b) *A* pseudorandom generator *is a function* D *that, for each* n, *maps* Σ^n *into* $\Sigma^{r(n)}$, *where* $r(n) \geq n+1$. *The function* r *is called the* stretching *of* D.

(c) *The* security $S_D(n)$ *of* D *at length* n *is the largest integer* s *such that, for any circuit* C *of size at most* s *with* $r(n)$ *inputs,*

$$\left|\Pr_x[C(x) = 1] - \Pr_y[C(D(y)) = 1]\right| \leq \frac{1}{s},$$

where x *is uniformly distributed over* $\Sigma^{r(n)}$ *and* y *over* Σ^n.

We will use pseudorandom generators with the following characteristics:

(1) an E-computable pseudorandom generator D_1 that stretches seeds super-polynomially and has super-polynomial security at infinitely many lengths;
(2) an EXP-computable pseudorandom generator D_2 of security $2^{n^{\Omega(1)}}$; and
(3) an E-computable pseudorandom generator D_3 of security $2^{\Omega(n)}$.

D_1 will be applied in the next section; in this section we will use D_2 and D_3. None of these generators is known to exist unconditionally. However, a highly plausible hypothesis suffices for the weakest generator D_1, as follows simply by combining the work of [BFNW93] and [NiWi94] with some padding.

THEOREM 5.3. *If* MA $\neq$ EXP, *then there is an* E*-computable pseudorandom generator* D_1 *with stretching* $n^{\Theta(\log n)}$ *such that for any integer* k, *there are infinitely many* n *with* $S_{D_1}(n) > n^k$.

Proof. From the proof of Lemma 4.1 of [BFNW93], it follows that if MA $\neq$ EXP, then there is a set $A \in$ EXP such that for any integer j, there are infinitely many m such that $H_A(m) > m^j$. From the proof of the main Theorem 1 in [NiWi94], it follows that for any set $A \in$ EXP, there is an EXP-computable pseudorandom generator D with stretching $n^{\Theta(\log n)}$ such that $S_D(n) = \Omega(H_A(\sqrt{n})/n)$. Say that D is computable in time 2^{n^c} for some integer constant $c > 0$. For any $k > 0$, the infinitely many m promised above with $j = 2(ck+1)$ yield infinitely many n of the form $m^{2/c}$ such that $S_D(n^{1/c}) > n^k$. Defining $D_1(x) = D(x')$, where x' denotes the prefix of x of length $|x|^{1/c}$, yields the required pseudorandom generator. □

Exponential-time computable pseudorandom generators with exponential security have the interesting property that we can blow up the stretching exponentially without significantly reducing the security. As with Theorem 5.3, credit for this observation should be distributed among the references cited in the proof.

THEOREM 5.4.

(a) *Given an* EXP*-computable pseudorandom generator* D_0 *of security* $2^{n^{\Omega(1)}}$*, we can construct an* EXP*-computable pseudorandom generator* D_2 *of security* $2^{n^{\Omega(1)}}$ *and stretching* $2^{n^{\Omega(1)}}$.

(b) *Given an* E*-computable pseudorandom generator* D_0 *of security* $2^{\Omega(n)}$*, we can construct an* E*-computable pseudorandom generator* D_3 *of security* $2^{\Omega(n)}$ *and stretching* $2^{\Omega(n)}$.

Proof. For (a), Nisan and Wigderson [NiWi94] showed that the existence of an E-computable pseudorandom generator with stretching $n+1$ (a "quick extender" in their terminology) with security $2^{n^{\Omega(1)}}$ is equivalent to the existence of an E-computable pseudorandom generator with stretching and security $2^{n^{\Omega(1)}}$. See statements (3) and (4) of their main theorem (Theorem 1) instantiated with $s(\ell) = 2^\ell$. As used in [BFNW93], their main result carries through if we replace "E-computable" by "EXP-computable" in both statements, owing to padding. Since the existence of D_0 implies the existence of an EXP-computable extender with security $2^{n^{\Omega(1)}}$, the existence of D_2 follows.

For (b), first define $D'(x)$ to be the first $|x|+1$ bits of $D_0(x)$. Then D' is an extender with security $2^{\Omega(n)}$, and this implies that the range of D' is a language in E requiring circuits of size $2^{\Omega(n)}$. Impagliazzo and Wigderson, in their proof of Theorem 2 in [ImWi97], showed how to transform such a language into a language $A \in$ E such that $H_A(n) = 2^{\Omega(n)}$. Using this A in part (3) of Theorem 2 of [NiWi94] yields an E-computable pseudorandom generator D_3 of security and stretching $2^{\Omega(n)}$. (It is also possible to argue that the range of D' is sufficiently hard to employ the technique of [NiWi94], without going through [ImWi97].) □

Pseudorandom generators of security $2^{n^{\Omega(1)}}$ (even polynomial-time computable ones) are fairly widely believed to exist (see [BlMi84, RaRu97, Bon99]), and while those of security $2^{\Omega(n)}$ are more controversial even for EXP-computability, their existence was made more plausible by the result of [ImWi97] used in the proof of (b) above. Polynomial-time computable pseudorandom generators of security $2^{\Omega(n)}$ exist relative to a random oracle [Zim95, Imp99pc], and E-computable ones also exist if P = NP. (The latter observation follows by combining the techniques of Kannan [Kan82] with padding and the above-mentioned result of [ImWi97]; it is noted by the second author as "Corollary 2.2.19" in his dissertation [Mel99].)

The following general result shows how pseudorandom generators can be used to approximate averages. It provides the accuracy and time bounds needed for applying Lemma 5.1 to get the desired martingale.

THEOREM 5.5. *Let D be a pseudorandom generator computable in time $\delta(n)$ and with stretching $r(n)$. Let $f : \Sigma^* \times \Sigma^* \to (-\infty, \infty)$ be a function that is computed in linear time on a Turing machine, and let $s, R, m : \mathbf{N} \to \mathbf{N}$ be fully time-constructible functions such that $s(N) \geq N$ and the following relations hold for any integer $N \geq 0$, $w \in \Sigma^N$, and $\rho \in \Sigma^{s(N)}$:*

$$\begin{aligned} |f(w,\rho)| &\leq R(N), \\ r(m(N)) &\geq s(N), \\ S_D(m(N)) &\geq (s(N) + R(N))^6. \end{aligned} \tag{10}$$

Then we can approximate

$$h(w) = E_{|\rho|=s(N)}[f(w,\rho)] \tag{11}$$

to within N^{-2} in time $O(2^{m(N)} \cdot (s(N) + R(N))^4 \cdot \delta(m(N)))$.

Proof. For any $N \geq 0$, let $\mathcal{I}_N$ be a partition of the interval $[-R(N), R(N)]$ into subintervals of length $\frac{1}{2N^2}$. Note that $|\mathcal{I}_N| = 4N^2R(N)$. Define for any $I \in \mathcal{I}_N$ and any string w of length N

$$\pi(I,w) = \Pr_{|\rho|=s(N)}[f(w,\rho) \in I].$$

The predicate in $[\ldots]$ can be computed by circuits of size $O(s(N) \log s(N))$, using the t-to-$O(t \log t)$ Turing-machine-time-to-circuit-size construction of Pippenger and Fischer [PiFi79]. Since $S_D(m(N)) = \omega(s(N) \log s(N))$, it follows that

$$\tilde{\pi}(I,w) = \Pr_{|\sigma|=m(N)}[f(w, D(\sigma)[1 \ldots s(N)]) \in I]$$

approximates $\pi(I,w)$ to within an additive error of $(S_D(m(N)))^{-1}$, and we can compute it in time $O(2^{m(N)} \cdot s(N) \cdot \delta(m(N)))$. We define the approximation $\tilde{h}(w)$ for $h(w)$ as

$$\tilde{h}(w) = \sum_{I \in \mathcal{I}_N} \tilde{\pi}(I,w) \min(I).$$

Since we can write $h(w)$ as

$$h(w) = \sum_{I \in \mathcal{I}_N} \pi(I,w) E_{|\rho|=s(N)}[f(w,\rho) \mid f(w,\rho) \in I],$$

we can bound the approximation error as follows:

$$\begin{aligned} &|h(w) - \tilde{h}(w)| \\ &\leq \sum_{I \in \mathcal{I}_N} \pi(I,w) \left| E_{|\rho|=s(N)}[f(w,\rho) \mid f(w,\rho) \in I] - \min(I) \right| \\ &\quad + \sum_{I \in \mathcal{I}_N} \left| \pi(I,w) - \tilde{\pi}(I,w) \right| \min(I) \\ &\leq \max_{I \in \mathcal{I}_N}(|I|) + |\mathcal{I}_N| \cdot (S_D(m(N)))^{-1} \cdot R(N) \\ &\leq \frac{1}{2N^2} + 4N^2 \cdot R^2(N) \cdot (S_D(m(N)))^{-1} \leq \frac{1}{N^2}. \end{aligned}$$

Computing $\tilde{h}(w)$ requires $|\mathcal{I}_N| = 4N^2R(N)$ evaluations of $\tilde{\pi}$, which results in the claimed upper bound for the time complexity of $\tilde{h}$. □

Now we would like to apply Theorem 5.5 to approximate $h = d_G$ given by (7) to within N^{-2}, by setting $f = f_G$ and $s(N) = t(\log N)$. However, for a general betting game G running in time $t(n)$, we can only guarantee an upper bound of $R(N) = 2^{t(\log N)} \cdot c_G(\lambda)$ on $|f(w, \rho)|$. Since S_D can be at most exponential, condition (10) would force $m(N)$ to be $\Omega(t(\log N))$. In that case, Theorem 5.5 can only yield an approximation computable in time $2^{O(t(\log N))}$. *However*, we can assume without loss of generality that G satisfies the slow-winnings condition (6) of Lemma 3.6, in which case an upper bound of $R(N) \in O(N)$ holds. Then the term $s(N)$ in the right-hand side of (10) dominates, provided $t(n) = 2^{\Omega(n)}$.

Taking everything together, we obtain the following result about transforming E- and EXP-betting games into equivalent E-, respectively, EXP-martingales.

THEOREM 5.6. *If there is a pseudorandom generator computable in* E *with security* $2^{\Omega(n)}$*, then for every* E*-betting game* G*, there exists an* E*-martingale* d *such that* $S^\infty[G] \subseteq S^\infty[d]$*. If there is a pseudorandom generator computable in* EXP *with security* $2^{n^{\Omega(1)}}$*, then for every* EXP*-betting game* G*, there exists an* EXP*-martingale* d *such that* $S^\infty[G] \subseteq S^\infty[d]$*.*

Proof. By Lemma 3.6, we can assume that c_G satisfies both the sure-winnings condition (5) as well as the slow-winnings condition (6). Because of Lemmas 4.3 and 5.1 (since the series $\sum_{i=1}^{\infty} \frac{1}{i^2}$ converges), it suffices to approximate the function $d_G(w)$ given by (7) to within N^{-2} in time $2^{O(n)}$, respectively, $2^{n^{O(1)}}$, where $N = |w|$ and $n = \log N$.

Under the given hypothesis about the existence of an E-computable pseudorandom generator D_0, we can take D to be the pseudorandom generator D_3 provided by Theorem 5.4(b). Thus we meet the conditions for applying Theorem 5.5 to $h = d_G$ with $s(N) = N^{O(1)}$, $R(N) = O(N)$, and $m(N) = O(\log N)$, and we obtain the approximation of d_G that we need. In the case of an EXP-betting game G, to obtain an EXP-martingale we can take D to be the pseudorandom generator D_2 of weaker security guarantee $2^{n^{\Omega(1)}}$ provided by Theorem 5.4(a). Then we meet the requirements of Theorem 5.5 with $s(N) = 2^{(\log N)^{O(1)}}$, $R(N) = O(N)$, and $m(N) = (\log N)^{O(1)}$. □

6. Autoreducible sets. An oracle Turing machine M is said to *autoreduce* a language A if $L(M^A) = A$, and for all strings x, M^A on input x does not query x. That is, one can learn the membership of x by querying strings other than x itself. If M runs in polynomial time, then A is P-*autoreducible*—we also write $\leq_T^p$-autoreducible. If M is also nonadaptive, then A is $\leq_{tt}^p$-*autoreducible*.

One can always code M so that for *all* oracles, it *never* queries its own input—then we call M an *autoreduction*. Hence we can define an effective enumeration $[M_i]_{i=1}^{\infty}$ of polynomial-time autoreductions, such that a language A is autoreducible iff there exists an i such that $L(M_i^A) = A$. (For a technical aside: the same M_i may autoreduce different languages A, and some M_i may autoreduce no languages at all.) The same goes for $\leq_{tt}^p$-*autoreductions*.

Autoreducible sets were brought to the polynomial-time context by Ambos-Spies [Amb84]. Their importance was further argued by Buhrman, Fortnow, Van Melkebeek, and Torenvliet [BFvMT98], who showed that all $\leq_T^p$-complete sets for EXP are $\leq_T^p$-autoreducible (while some complete sets for other classes are not). Here we demonstrate that autoreducible sets are important for testing the power of resource-bounded measure.

6.1. Adaptively autoreducible sets. As stated in the introduction, if the $\leq_T^p$-autoreducible sets in EXP (or sufficiently the $\leq_T^p$-complete sets for EXP) are covered by an EXP-martingale, then EXP $\neq$ BPP, a nonrelativizing consequence. However, it is easy to cover them by an E-betting game. Indeed, the betting game uses its adaptive freedom only to "look ahead" at the membership of lexicographically greater strings, betting nothing on them.

THEOREM 6.1. *There is an* E*-betting game* G *that covers all* $\leq_T^p$*-autoreducible languages.*

Proof. Let $M_1, M_2, \ldots$ be an enumeration of $\leq_T^p$-autoreductions such that each M_i runs in time $n^i + i$ on inputs of length n. Our betting game G regards its capital as composed of infinitely many "shares" c_i, one for each M_i. Initially, $c_i = 1/2^i$. Letting $\langle \cdot, \cdot \rangle$ be a standard pairing function, inductively define $n_0 = 0$ and $n_{\langle i,j \rangle + 1} = (n_{\langle i,j \rangle})^i + i$.

During a stage $s = \langle i, j \rangle$, G simulates M_i on input $0^{n_{s-1}}$. Whenever M_i makes a query of length less than n_{s-1}, G looks up the answer from its table of past queries. Whenever M_i makes a query of length n_{s-1} or more, G places a bet of zero on that string and makes the same query. Then G bets all of the share c_i on $0^{n_{s-1}}$ according to the answer of the simulation of M_i. Finally, G "cleans up" by putting zero bets on all strings with length in $[n_{s-1}, n_s)$ that were not queries in the previous steps.

If M_i autoreduces A, then share c_i doubles in value at each stage $\langle i, j \rangle$ and makes the total capital grow to infinity. And G runs in time $2^{O(n)}$—indeed, only the "cleanup" phase needs this much time. □

COROLLARY 6.2. *Each of the following statements implies* BPP $\neq$ EXP*.*

1. *The class of* $\leq_T^p$*-autoreducible sets has* E*-measure zero.*
2. *The class of* $\leq_T^p$*-complete sets for* EXP *has* E*-measure zero.*
3. E*-betting games and* E*-martingales are equivalent.*
4. E*-betting games have the finite union property.*

The same holds if we replace E *by* EXP *in these statements.*

Proof. Let $\mathcal{C}$ stand for the class of languages that are not $\leq_T^p$-hard for BPP. Allender and Strauss [AlSt94] showed that $\mathcal{C}$ has E-measure zero, so trivially it is also covered by an E-betting game. Now let $\mathcal{D}$ stand for the class of $\leq_T^p$-complete sets for EXP. By Theorem 6.1 and the result of [BFvMT98] cited above, $\mathcal{D}$ is covered by an E-betting game.

If EXP = BPP, the union $\mathcal{C} \cup \mathcal{D}$ contains all of EXP, and

- if $\mathcal{D}$ would have E-measure zero, so would $\mathcal{C} \cup \mathcal{D}$ and hence EXP, contradicting the measure conservation property of Lutz measure;
- if E-betting games would have the finite-union property, then $\mathcal{C} \cup \mathcal{D}$ and EXP would be covered by an E-betting game, contradicting Theorem 3.4.

Since statement 1 implies statement 2, and statement 3 implies statement 4, then these observations suffice to establish the corollary for E. The proof for EXP is similar. □

Since there is an oracle A giving $\text{EXP}^A = \text{BPP}^A$ [Hel86], this shows that relativizable techniques cannot establish the equivalence of E-martingales and E-betting games, nor of EXP-martingales and EXP-betting games. They cannot refute it either, since there are oracles relative to which strong pseudorandom generators exist—all "random" oracles, in fact [Zim95].

6.2. Nonadaptively autoreducible sets. It is tempting to think that the *non*adaptively P-autoreducible sets should have E-measure zero, or at least EXP-measure zero, insofar as betting games are the adaptive cousins of martingales. How-

ever, it is not just adaptiveness but also the freedom to bet *out of the fixed lexicographic order* that adds power to betting games. If one carries out the proof of Theorem 6.1 to cover the class of $\leq_{tt}^{p}$-autoreducible sets, using an enumeration $[M_i]$ of $\leq_{tt}^{p}$-autoreductions, one obtains a *nonadaptive* E-betting game (defined formally below) that (independent of its oracle) bets on all strings in an order given by a single permutation of Σ^*. The permutation itself is E-computable. It might seem that an E-martingale should be able to "untwist" the permutation and succeed on all these sets. However, our next results, which strengthen the above corollary, close the same "nonrelativizing" door on proving this with current techniques.

THEOREM 6.3. *For any $k \geq 1$, the $\leq_{tt}^{p}$-complete languages for Δ_k^p are $\leq_{tt}^{p}$-autoreducible.*

Here is the proof idea, which follows techniques of [BFvMT98] for the theorem that all EXP-complete sets are $\leq_{T}^{p}$-autoreducible. Call a closed propositional formula that has at most k blocks of like quantifiers (i.e., at most $k-1$ quantifier alternations) a "QBF$_k$ formula," and let $TQBF_k$ stand for the set of true QBF formulas. Let A be a $\leq_{tt}^{p}$-complete set for $\Delta_{k+1}^{p} = \mathrm{P}^{\Sigma_k^p}$. Since $TQBF_k$ is Σ_k^p-hard, there is a deterministic polynomial-time oracle Turing machine M that accepts A with oracle $TQBF_k$. Let $q(x, i)$ stand for the ith oracle query made by M on input x. Whether $q(x, i)$ belongs to $TQBF_k$ forms a Δ_{k+1}^{p}-question, so we can $\leq_{tt}^{p}$-reduce it to A. It is possible that this latter reduction will include x itself among its queries. Let b_i^+ denote the answer it gives to the question provided that any query to x is answered "yes," and similarly define b_i^- in case x is answered "no."

If $b_i^+ = b_i^-$, which holds in particular if x is not queried, then we know the correct answer b_i to the ith query. If this situation occurs for all queries, we are finished: We just have to run M on input x using the b_i's as answers to the oracle queries. The b_i's themselves are obtained *without* submitting the (possibly adaptive) queries made by M, but rather by applying the latter $\leq_{tt}^{p}$-reduction to A to the pair $\langle x, i\rangle$, and without submitting any query on x itself. Hence this process satisfies the requirements of a $\leq_{tt}^{p}$-autoreduction of A for the particular input x.

Now suppose that $b_i^+ \neq b_i^-$ for some i, and let i be minimal. Then we will have two players play the k-round game underlying the QBF$_k$-formula that constitutes the ith oracle query. One player claims that b_i^+ is the correct value for b_i, which is equivalent to claiming that $x \in A$, while his opponent claims that b_i^- is correct and that $x \notin A$. Write $\chi_A(x) = 1$ if $x \in A$, and $\chi_A(x) = 0$ if $x \notin A$. The players' strategies will consist of computing the game history so far, determining their optimal next move, $\leq_{tt}^{p}$-reducing this computation to A, and finally producing the result of this reduction under their respective assumption about $\chi_A(x)$. This approach will allow us to recover the game history in polynomial time with nonadaptive queries to A different from x. Moreover, it will guarantee that the player making the correct assumption about $\chi_A(x)$ plays optimally. Since this player is also the one claiming the correct value for b_i, he will win the game. So, we output the winner's value for b_i.

It remains to show that we can compute the above strategies in deterministic polynomial time with a Σ_k^p oracle, i.e., in $\mathrm{FP}^{\Sigma_k^p}$. It seems crucial that the number k of alternations be constant here.

Proof of Theorem 6.3. Let A be a $\leq_{tt}^{p}$-complete set for Δ_{k+1}^{p} accepted by the polynomial-time oracle Turing machine M with oracle $TQBF_k$. Let $q(x, i)$ denote the ith oracle query of M^{TQBF_k} on input x. Then $q(x, i)$ can be written in the form $(\exists y_1)(\forall y_2)\ldots(Q_k y_k)\ \phi_{x,i}(y_1, y_2, \ldots, y_k)$, where $y_1, \ldots, y_k$ stand for the vectors of variables quantified in each block, or in the opposite form beginning with the

block ($\forall y_1$). By reasonable abuse of notation, we also let y_r stand for a string of 0-1 assignments to the variables in the rth block. Without loss of generality, we may suppose every oracle query made by M has this form where each y_j is a string of length $|x|^c$, and M makes exactly $|x|^c$ queries, taking the constant c from the polynomial time bound on M. Note that the function q belongs to $\mathrm{FP}^{\Sigma_k^p}$. Hence the language

$$L_0 = \{\, \langle x, y\rangle : q(x, i) \in TQBF_k \,\}$$

belongs to Δ_{k+1}^p. Since A is $\leq_{tt}^p$-complete for Δ_{k+1}^p, there is a polynomial-time nonadaptive oracle Turing machine N_0 that accepts L_0 with oracle A. Now define $b_i^+(x) = N_0^{A\cup\{x\}}(\langle x, i\rangle)$ and $b_i^-(x) = N_0^{A\setminus\{x\}}(\langle x, i\rangle)$. We define languages $L_1, L_2, \ldots, L_k \in \Delta_{k+1}^p$ and $\leq_{tt}^p$-reductions $N_1, N_2, \ldots, N_k$ inductively as follows.

Let $1 \leq \ell \leq k$. The set L_ℓ consists of all pairs $\langle x, j\rangle$ with $1 \leq j \leq |x|^c$, such that there is a smallest i, $1 \leq i \leq |x|^c$, for which $b_i^+(x) \neq b_i^-(x)$, and the following condition holds. For $1 \leq r \leq \ell - 1$, let the sth bit of y_r equal $N_r^{A\cup\{x\}}(\langle x, s\rangle)$ if $r \equiv b_i^+(x) \bmod 2$, and $N_r^{A\setminus\{x\}}(\langle x, s\rangle)$ otherwise. We put $\langle x, j\rangle$ into L_ℓ iff there is a lexicographically least y_ℓ such that

$$\chi[(Q_{\ell+1}y_{\ell+1})(Q_{\ell+2}y_{\ell+2})\ldots(Q_k y_k)\ \phi_{x,i}(y_1, y_2, \ldots, y_k)] \equiv \ell \bmod 2,$$

and the jth bit of y_ℓ is set to 1. The form of this definition shows that L_ℓ belongs to Δ_{k+1}^p. Hence we can take N_ℓ to be a polynomial-time nonadaptive oracle Turing machine that accepts L_ℓ with oracle A.

Now, we construct a $\leq_{tt}^p$-autoreduction for A. On input x, we compute $b_i^+(x)$ and $b_i^-(x)$ for $1 \leq i \leq |x|^c$, as well as $y_r^{(b)}$ for $b \in \{0, 1\}$ and $1 \leq r \leq |x|^c$. The latter quantity $y_r^{(b)}$ is defined as follows: for $1 \leq s \leq |x|^c$, the sth bit of $y_r^{(b)}$ equals $N_r^{A\cup\{x\}}(\langle x, s\rangle)$ if $r \equiv b \bmod 2$, and $N_r^{A\setminus\{x\}}(\langle x, s\rangle)$ otherwise. Note that we can compute all these values in polynomial time by making nonadaptive queries to A, none of which equals x.

If $b_i^+(x) = b_i^-(x)$ for every $1 \leq i \leq |x|^c$, we run M on input x using $b_i^+(x) = b_i^-(x)$ as the answer to the ith oracle query. Since it always holds that at least one of $b_i^+(x)$ and $b_i^-(x)$ equals the correct oracle answer $b_i(x)$, we faithfully simulate M on input x and hence compute $\chi_A(x)$ correctly.

Otherwise, let i be the first index for which $b_i^+(x) \neq b_i^-(x)$. Since $b_j(x) = b_j^+(x) = b_j^-(x)$ for $j < i$, we can determine $q(x, i)$ by simulating M on input x until it asks the ith query. We then output 1 if

$$b_i^+(x) = \phi_{x,i}(y_1^{(b_i^+(x))}, y_2^{(b_i^+(x))}, \ldots, y_k^{(b_i^+(x))})$$

and output 0 otherwise. We claim that this gives the correct answer to whether $x \in A$.

In order to prove the claim, consider the game history $y_1^{(b_i^+(x))}$, $y_2^{(b_i^+(x))}$, $\ldots$, $y_k^{(b_i^+(x))}$. The player claiming the correct value for $b_i(x)$ gets to play the rounds that allow him to win the game no matter what his opponent does. Since this player is also the one making the correct assumption about $\chi_A(x)$, an inductive argument shows that he plays optimally: At his stages ℓ, the string y_ℓ in the above construction of L_ℓ exists, and he plays it. The key for the induction is that at later stages $\ell' > \ell$, the value of y_ℓ at stage ℓ' remains the same as what it was at stage ℓ. Thus the player with the correct assumption about $\chi_A(x)$ wins the game—that is,

$\phi_{x,i}(y_1^{(b_i^+(x))}, y_2^{(b_i^+(x))}, \ldots, y_k^{(b_i^+(x))})$ equals his guess for $b_i(x)$ (and not the other player's guess). □

In order to formalize the strengthening of Corollary 6.2 that results from Theorem 6.3, we call a betting game G *nonadaptive* if the infinite sequence $x_1, x_2, x_3, \ldots$ of queries G^A makes is the same for all oracles A. If G runs in $2^{O(n)}$ time, and this sequence hits all strings in Σ^*, then the permutation π of the standard ordering $s_1, s_2, s_3, \ldots$ defined by $\pi(s_i) = x_i$ is both computable and invertible in $2^{O(n)}$ time. It is computable in this amount of time because in order to hit all strings, G must bet on all strings in $\{0,1\}^n$ within the first $2^{O(n)}$ steps. Hence its ith bet must be made in a number of steps that is singly-exponential in the length of s_i. And to compute $\pi^{-1}(x_i)$, G need only be run for $2^{O(|x_i|)}$ steps, since it cannot query x_i after this time. Since π and its inverse are both E-computable, π is a reasonable candidate to replace lexicographic ordering in the definition of E-martingales, and likewise for EXP-martingales. We say a class $\mathcal{C}$ has π-E-measure zero if $\mathcal{C}$ can be covered by an E-martingale that interprets its input as a characteristic string *in the order given by* π.

THEOREM 6.4. *The class of $\leq_{tt}^p$-autoreducible languages can be covered by a nonadaptive* E*-betting game. Hence there is an* E*-computable and invertible permutation π of Σ^* such that this class has π-*E*-measure zero.*

Proof. With reference to the proof of Theorem 6.1, we can let $M_1, M_2, \ldots$ be an enumeration of $\leq_{tt}^p$-autoreductions such that each M_i runs in time $n^i + i$. The machine G in that proof automatically becomes nonadaptive, and since it queries all strings, it defines a permutation π of Σ^* as above with the required properties. □

COROLLARY 6.5. *Each of the following statements implies* BPP $\neq$ EXP*, as do the statements obtained on replacing* "E" *by* "EXP."

1. *The class of $\leq_{tt}^p$-autoreducible sets has* E*-measure zero.*
2. *The class of $\leq_{tt}^p$-complete sets for* EXP *has* E*-measure zero.*
3. *Nonadaptive* E*-betting games and* E*-martingales are equivalent.*
4. *If two classes can be covered by nonadaptive* E*-betting games, then their union can be covered by an* E*-betting game.*
5. *For all classes $\mathcal{C}$ and all* E*-computable and invertible orderings π, if $\mathcal{C}$ has π-*E*-measure zero, then $\mathcal{C}$ has* E*-measure zero.*

Proof. It suffices to make the following two observations to argue that the proof of Corollary 6.2 carries over to the truth-table cases.

- The construction of Allender and Strauss [AlSt94] actually shows that the class of sets that are not $\leq_{tt}^p$-hard for BPP has E-measure zero.
- If EXP = BPP, Theorem 6.3 implies that all $\leq_{tt}^p$-complete sets for EXP are $\leq_{tt}^p$-autoreducible, because BPP $\subseteq \Sigma_2^{\mathrm{p}} \subseteq \Delta_3^{\mathrm{p}} \subseteq$ EXP.

Theorem 6.4 and the finite-unions property of Lutz's measures on E and EXP do the rest. □

The last point of Corollary 6.5 asserts that Lutz's definition of measure on E is invariant under all E-computable and invertible permutations. These permutations include *flip* from the introduction and (crucially) π from Theorem 6.4. Hence this robustness assertion for Lutz's measure implies BPP $\neq$ EXP. Our "betting-game measure" (both adaptive and nonadaptive) does enjoy this permutation invariance, but asserting the finite-unions property for it also implies BPP $\neq$ EXP. The rest of this paper explores conditions under which Lutz's martingales *can* cover classes of autoreducible sets, thus attempting to narrow the gap between them and betting games.

6.3. Covering autoreducible sets by martingales. This puts the spotlight on the question: Under what hypotheses can we show that the $\leq^p_{tt}$-autoreducible sets have E-measure zero? Any such hypothesis must be strong enough to imply EXP $\neq$ BPP, but we hope to find hypotheses weaker than assuming the equivalence of (E- or EXP-) betting games and martingales, or assuming the finite-union property for betting games. Do we need strong pseudorandom generators to cover the $\leq^p_{tt}$-autoreducible sets? How close can we come to covering the $\leq^p_T$-autoreducible sets by an E-martingale?

Our final results show that the hypothesis MA $\neq$ EXP suffices. This assumption is only known to yield pseudorandom generators of super-polynomial security (at infinitely many lengths) rather than exponential security (at almost all lengths). Recall that MA contains both BPP and NP; in fact it is sandwiched between $\mathrm{NP}^{\mathrm{BPP}}$ and $\mathrm{BPP}^{\mathrm{NP}}$.

THEOREM 6.6. *If* MA $\neq$ EXP*, then the class of* $\leq^p_{tt}$*-autoreducible sets has* E*-measure zero.*

We actually obtain a stronger conclusion.

THEOREM 6.7. *If* MA $\neq$ EXP*, then the class of languages A autoreducible by polynomial-time oracle Turing machines that always make their queries in lexicographic order has* E*-measure zero.*

To better convey the essential sampling idea, we prove the weaker Theorem 6.6 before the stronger Theorem 6.7. The extra wrinkle in the latter theorem is to use the pseudorandom generator *twice*, both to construct the set of "critical strings" to bet on and to compute the martingale.

Proof of Theorem 6.6. Let $[M_i]_{i=1}^{\infty}$ enumerate the $\leq^p_{tt}$-autoreductions, with each M_i running in time n^i. Divide the initial capital into *shares* $s_{i,m}$ for $i, m \geq 1$, with each $s_{i,m}$ valued initially at $(1/m^2)(1/2^i)$. For each share $s_{i,m}$, we will describe a martingale that is active only on a finite number of strings x. The martingale will be active only if $i \leq m/2\lceil \log_2 m \rceil$ and $m \leq |x| \leq m^i$, and further only if x belongs to a set $J = J_{i,m}$ constructed below. Hence the martingale will be inactive outside J, and we will be able to apply Lemma 5.1. We will arrange that whenever M_i autoreduces A, there are infinitely many m such that share $s_{i,m}$ attains a value above 1 (in fact, close to m) along A. Hence the martingale defined by all the shares succeeds on A. We will also ensure that each active share's bets on strings of length n are computable in time 2^{an}, where the constant a is independent of i. This is enough to make the whole martingale E-computable and complete the proof.

To describe the betting strategy for $s_{i,m}$, first construct a set $I = I_{i,m}$ starting with $I = \{\, 0^m \,\}$ and iterating as follows: Let y be the lexicographically least string of length m that does not appear among queries made by M_i on inputs $x \in I$. Then add y to I. Do this until I has $3\lceil \log_2 m \rceil$ strings in it. This is possible because the bound $3\lceil \log_2 m \rceil m^i$ on the number of queries M_i could possibly make on inputs in I is less than 2^m. Moreover, 2^m bounds the time needed to construct I. Thus we have arranged that

$$\text{for all } x, y \in I \text{ with } x < y,\ M_i(x) \text{ does not query } y. \tag{12}$$

Now let J stand for I together with all the queries M_i makes on inputs in I. Adapting ideas from Definition 4.1 to this context, let us define a finite Boolean function $\beta : J \to \{\, 0, 1 \,\}$ to be *consistent with* M_i *on* I, written $\beta \sim_I M_i$, if for all $x \in I$, M_i run on input x with oracle answers given by β agrees with the value $\beta(x)$. Given a characteristic prefix w, also write $\beta \sim w$ if $\beta(x)$ and $w(x)$ agree on all x in J and

the domain of w. Since I and J depend only on i and m, we obtain a "probability density" function for each share $s_{i,m}$ via

$$\pi_{i,m}(w) = \Pr_{\beta \sim w}[\beta \sim_I M_i]. \tag{13}$$

The martingale $d_{i,m}$ standardly associated to this density (as in [Lutz92]) is definable inductively by $d_{i,m}(\lambda) = 1$ and

$$d_{i,m}(w1) = d_{i,m}(w)\frac{\pi_{i,m}(w1)}{\pi_{i,m}(w)}, \qquad d_{i,m}(w0) = d_{i,m}(w)\frac{\pi_{i,m}(w0)}{\pi_{i,m}(w)}. \tag{14}$$

(In case $\pi_{i,m} = 0$, we already have $d_{i,m}(w) = 0$, and so both $d_{i,m}(w1)$ and $d_{i,m}(w0)$ are set to 0.)

Note that the values $\pi_{i,m}(wb)$ for $b = 0, 1$ can only differ from $\pi_{i,m}(w)$ if the string x indexed by b belongs to J, i.e., $d_{i,m}$ is inactive outside J.

CLAIM 6.8. *If M_i autoreduces A, then for all sufficiently large m, if share $s_{i,m}$ could play the strategy $d_{i,m}$, then on A its value would rise to (at least) $m/2^i$. That is, $s_{i,m}$ would multiply its initial value by (at least) m^3.*

To see this, first note that for any $w \sqsubseteq A$ long enough to contain J in its domain, $\pi_{i,m}(w) = 1$. We want to show that for any v short enough to have domain disjoint from I, $\pi_{i,m}(v) = 1/2^{|I|}$. To do this, consider any fixed 0-1 assignment β_0 to strings in $J \setminus I$ that agrees with v. This assignment determines the computation of M_i on the lexicographically first string $x \in I$, using β_0 to answer queries, and hence forces the value of $\beta(x)$ in order to maintain consistency on I. This in turn forces the value $\beta(x')$ on the next string x' in I, and so on. Hence only one out of $2^{|I|}$ possible completions of β_0 to β is consistent with M_i on I. Thus $\pi_{i,m}(v) = 1/2^{|I|}$. Since $d_{i,m}(w) = d_{i,m}(v)\cdot(\pi_{i,m}(w)/\pi_{i,m}(v))$ by (14), and $2^{|I|} = 2^{3\lceil \log_2 m \rceil} \geq m^3$, Claim 6.8 is proved.

The main obstacle now is that $\pi_{i,m}$ in (13), and hence $d_{i,m}(w)$, may not be computable in time 2^{an} with a independent of i. The number of assignments β to count is on the order of $2^{|J|} \approx 2^{m^i} \approx 2^{n^i}$. Here is where we use the E-computable pseudorandom generator D_1, with super-polynomial stretching and with super-polynomial security at infinitely many lengths, obtained via Theorem 5.3 from the hypothesis MA $\neq$ EXP. For all i and sufficiently large m, D_1 stretches a seed s of length m into at least $3\lceil \log_2 m \rceil m^i$ bits, which are enough to define an assignment β_s to J (agreeing with any given w). We estimate $\pi_{i,m}(w)$ by

$$\hat{\pi}_{i,m}(w) = \Pr_{|s|=m}[\beta_s \sim_I M_i]. \tag{15}$$

Take $\epsilon = 1/m^{i+4}$. By Theorem 5.3 there are infinitely many "good" m such that $S_{D_1}(m) > m^{i+4}$.

CLAIM 6.9. *For all large enough good m,* every *estimate $\hat{\pi}_{i,m}(w)$ gives $|\hat{\pi}_{i,m}(w) - \pi_{i,m}(w)| \leq \epsilon$.*

Suppose not. First note that (13) and (15) do not depend on all of w, just on the up-to-$3\lceil \log_2 m \rceil m^i < m^{i+1}$ bits in w that index strings in J, and these can be hardwired into circuits. The tests $[\beta \sim_I M_i]$ can also be done by circuits of size $o(m^{i+1})$, because a Turing machine computation of time r can be simulated by circuits of size $O(r \log r)$ [PiFi79]. Hence we get circuits of size less than $S_{D_1}(m)$ achieving a discrepancy greater than $1/S_{D_1}(m)$, which is a contradiction. This proves Claim 6.9.

Finally, observe that the proof of Claim 6.8 gives us not only $d_{i,m}(w) \geq \pi_{i,m}(w) \cdot m^3$, but also $d_{i,m}(w) = \Theta(\pi_{i,m}(w) \cdot m^3)$, when $w \sqsubseteq A$. For $w \sqsubseteq A$ and good m, we

thus obtain estimates $g(w)$ for $d_{i,m}(w)$ within error bounds $\epsilon' = \Theta(\epsilon) = \Theta(1/m^{i+1})$. Now applying Lemma 5.1 for this $g(w)$ and $J = J_{i,m}$ yields a martingale $d'_{i,m}(w)$ computable in time 2^{an}, where the constant a is independent of i. This $d'_{i,m}(w)$ is the martingale computed by the actions of share $s_{i,m}$. Since $K = \sum_{s_i \in J} \epsilon' = |J|\epsilon' \leq (1/m) \cdot 3\lceil \log_2 m \rceil = o(1)$, we actually obtain $|d'_{i,m}(w) - d_{i,m}(w)| = o(1)$, which is stronger than what we needed to conclude that share $s_{i,m}$ returns enough profit. This completes the proof of Theorem 6.6. □

To prove Theorem 6.7, we need to construct sets $I = I_{i,m}$ with properties similar to (12), in the case where M_i is no longer a $\leq^p_{tt}$-autoreduction but makes its queries in lexicographic order. To carry out the construction of I, we use the pseudorandom generator D_1 a second time and actually need only that M_i on input 0^m makes all queries of length $< m$ before making any query of length $\geq m$. To play the modified strategy for share $s_{i,m}$, however, appears to require that all queries observe lexicographic order.

Proof of Theorem 6.7. Recall that the hypothesis EXP $\neq$ MA yields a pseudorandom generator D_1 computable in time $2^{O(m)}$ and stretching m bits to $r(m)$ bits such that for all i, all sufficiently large m give $r(m) > m^i$, and infinitely many m give hardness $S_{D_1}(m) > m^i$. Let $[M_i]_{i=1}^{\infty}$ be a standard enumeration of $\leq^p_T$-autoreductions that are constrained to make their queries in lexicographic order, with each M_i running in time $O(n^i)$. We need to define strategies for "shares" $s_{i,m}$ such that whenever M_i autoreduces A, there are infinitely many m such that share $s_{i,m}$ grows its initial capital from $1/m^2 2^i$ to $1/2^i$ or more. The strategy for $s_{i,m}$ must still be computable in time 2^{am} where a is independent of i.

To compute the strategy for $s_{i,m}$, we note first that $s_{i,m}$ can be left inactive on strings of length $< m$. The overall running time allowance $2^{O(m)}$ permits us to suppose that by the time $s_{i,m}$ becomes active and needs to be considered, the initial segment w_0 of A (where A is the language on which the share happens to be playing) that indexes strings of length up to $m-1$ is known. Hence we may regard w_0 as fixed. For any $\alpha \in \{0,1\}^{m^i}$ let $M_i^{\alpha}(x)$ stand for the computation in which w_0 is used to answer any queries of length $< m$ and α is used to answer all other queries. Because of the order in which M_i makes its queries, those queries y answered by w_0 are the same for all α, so that those answers can be coded by a string u_0 of length at most m^i. Now for any string y of length equal to m, define

$$P(x,y) = \Pr_{\alpha}[M_i^{\alpha}(x) \text{ queries } y].$$

Note that given u_0 and α, the test "$M_i^{\alpha}(x)$ queries y" can be computed by circuits of size $O(m^{i+1})$. Hence by using the pseudorandom generator D_1 at length m, we can compute uniformly in E an approximation $P_1(x,y)$ for $P(x,y)$ such that for infinitely many m, said to be "good" m, *all* pairs x, y give $|P_1(x,y) - P(x,y)| \leq \epsilon_m$, where we choose $\epsilon_m = 1/m^4$.

Here is the algorithm for constructing $I = I_{i,m}$. Start with $I := \emptyset$, and while $|I| < 3\log_2 m$, do the following: Take the lexicographically least string $y \in \Sigma^m \setminus I$ such that for all $x \in I$, $P_1(x,y) \leq \epsilon_m$. The search for such a y will succeed within $|I| \cdot m^{i+4}$ trials, since for any particular x, there are fewer than m^{i+4} strings y overall that will fail the test. (This is so even if m is not good, because it only involves P_1, and because P_1 involves simulating $M_i^{D_1(s)}$ over all seeds s.) There is enough room to find such a y provided $|I|m^{i+4} \leq 2^m$, which holds for all sufficiently large m. The whole construction of I can be completed within time 2^{2am}. It follows that for any sufficiently large good m and $x, y \in I$ with $x < y$, $\Pr_{\alpha}[M_i^{\alpha}(x) \text{ queries } y] < 2\epsilon_m =$

$2/m^4$.

At this point we would like to define J to be "I together with the set of strings queried by M_i on inputs in I" as before, but unlike the previous case where M_i was nonadaptive, this is not a valid definition. We acknowledge the dependence of the strings queried by M_i on the oracle A by defining

$$J_A := I \cup \{\, y : (\exists x \in I) M_i^A(x) \text{ queries } y \,\}.$$

Let $r = m^i \cdot \lceil 3 \log m \rceil$. Then $|J_A| \leq r$; that is, J_A has the same size as J in the previous proof. This latter definition will be OK *because* M_i makes its queries in lexicographic order. Hence the share $s_{i,m}$, having already computed I without any reference to A, can determine the strings in J_A on which it should be active on the fly, in lexicographic order. Thus we can well-define a mapping β from $\{\,0,1\,\}^r$ to $\{\,0,1\,\}$ so that for any $k \leq r$, $\beta(k) = 1$ means that the query string y that happens to be kth in order in the on-the-fly construction of J_A is answered "yes" by the oracle. Then we may write J_β for J_A, and then write $\beta(y) = 1$ in place of $\beta(k) = 1$. Most important, given any $x \in I$, every such β well-defines a computation $M_i^\beta(x)$. This entitles us to carry over the two "consistency" definitions from the proof of Theorem 6.6:

- $\beta \sim w$ if $\beta(y) = w(y)$ for all $y \in J_\beta$;
- $\beta \sim_I M_i$ if for all $x \in I$, $M_i^\beta(x)$ equals (i.e., "agrees with") $\beta(x)$.

Finally, we may apply the latter notion to initial subsets of I and define for $1 \leq \ell \leq 3 \log m$ the predicate

$$R_\ell(\beta) = (\beta \sim_{x_1,\ldots,x_\ell} M_i) \ \wedge \\ (\forall j,k : 1 \leq j \leq k \leq \ell) M_i^\beta(x_j) \text{ does not query } x_k.$$

Claim 6.10. *For all ℓ, $\Pr_\beta[R_\ell(\beta)] \leq 1/2^\ell$.*

For the base case $\ell = 1$, $\Pr_\beta[R_1(\beta)] = 1/2$, because $M_i(x)$ does not query x_1, M_i being an autoreduction, and because whether $\beta \sim_{x_1} M_i$ depends only on the bit of β corresponding to x_1. Working by induction, suppose $\Pr_\beta[R_{\ell-1}(\beta)] \leq 1/2^{\ell-1}$. If $R_{\ell-1}(\beta)$ holds, then taking β' to be β with the bit corresponding to x_ℓ flipped, $R_{\ell-1}(\beta')$ also holds. However, at most one of $R_\ell(\beta)$ and $R_\ell(\beta')$ holds, again because $M_i(x_\ell)$ does not query x_ℓ. Hence $\Pr_\beta[R_\ell(\beta)] \leq (1/2)\Pr_\beta[R_{\ell-1}(\beta)]$, and this proves Claim 6.10. (It is possible that neither $R_\ell(\beta)$ nor $R_\ell(\beta')$ holds, as happens when $M_i^\beta(x_j)$ queries x_ℓ for some j, but this does not hurt the claim.)

Now we can rejoin the proof of Theorem 6.6 at (13), defining the probability density function $\pi_{i,m}(w) = \Pr_{\beta \sim w}[\beta \sim_I M_i]$. We get a martingale $d_{i,m}$ from $\pi_{i,m}$ as before, and this represents an "ideal" strategy for share $s_{i,m}$ to play. The statement corresponding to Claim 6.8 is the following.

Claim 6.11. *If M_i autoreduces A and m is good and sufficiently large, then the ideal strategy for share $s_{i,m}$ multiplies its value by at least $m^3/2$ along A.*

To see this, note that we constructed $I = \{\, x_1, \ldots, x_{3 \log m} \,\}$ above so that for all $j < k$, $\Pr_\alpha[M_i^\alpha(x_j) \text{ queries } x_k] \leq 2/m^4$. It follows that

$$\Pr[(\exists j,k : 1 \leq j \leq k \leq 3 \log m)\ M_i(x_j) \text{ queries } x_k] \leq \binom{\lceil 3 \log m \rceil}{2} \cdot \frac{2}{m^4} \leq \frac{1}{m^3},$$

provided $m \geq \lceil 3 \log m \rceil^2$. Hence, using Claim 6.10 with $\ell = 3 \log m$, we get

$$\Pr_\beta[\beta \sim_I M_i] \leq \frac{1}{2^{3 \log m}} + \frac{1}{m^3} = \frac{2}{m^3}.$$

Since the β defined by A satisfies $\beta \sim_I M_i$, it follows by the same reasoning as in Claim 6.8 that $d_{i,m}$ profits by at least a fraction of $m^3/2$ along A. This proves Claim 6.11.

Finally, we (re)use the pseudorandom generator D_1 as before to expand a seed s of length m into a string β_s of (at least) $r = 3\lceil \log_2 m \rceil m^i$ bits. Given any w, β_s well-defines a β and a set J_β of size at most r as constructed above by using w to answer queries in the domain of w and β_s for everything else. We again obtain the estimate $\hat{\pi}_{i,m}(w) = \Pr_{|s|=m}[\beta_s \sim_I M_i]$ from (15), with the same time complexity as before. Now we repeat Claim 6.9 in this new context as follows.

CLAIM 6.12. *For all large enough good m, every estimate $\hat{\pi}_{i,m}(w)$ gives $|\hat{\pi}_{i,m}(w) - \pi_{i,m}(w)| \leq \epsilon$.*

If not, then for some fixed w the estimate fails. The final key point is that because M_i always makes its queries in lexicographic order, the queries in the domain of w that need to be covered are the same for every β_s. Hence the corresponding bits of w can be hard-wired by circuitry of size at most r. The test $[\beta_s \sim_I M_i]$ can thus still be carried out by circuits of size less than m^{i+1}, and we reach the same contradiction of the hardness value S_{D_1}.

Finally, we want to apply Lemma 5.1 to replace $d_{i,m}(w)$ by a martingale $d'_{i,m}(w)$ that yields virtually the same degree of success and is computable in time $2^{O(n)}$. Unlike the truth-table case, we cannot apply Lemma 5.1 verbatim because we no longer have a single small set J that d' is active on. However, along any set A, the values $d'_{i,m}(w)$ and $d'_{i,m}(wb)$ ($b = 0$ or 1) can differ only for cases where b indexes a string in the small set J corresponding to A, and the reader may check that the argument and bounds of Lemma 5.1 go through unscathed in this case. This finishes the proof of Theorem 6.7. □

7. Conclusions. The initial impetus for this work was a simple question about measure: is the pseudorandomness of a characteristic sequence invariant under simple permutations such as that induced by *flip* in the introduction? The question for *flip* is tantalizingly still open. However, in section 6.2 we showed that establishing a "yes" answer for any permutation that intuitively *should* preserve the same complexity-theoretic degree of pseudorandomness, or even for a single specific such permutation as that in the simple proof of the nonadaptive version of Theorem 6.1, would have the major consequence that EXP $\neq$ BPP.

Our "betting games" in themselves are a natural extension of Lutz's measures for deterministic time classes. They preserve Lutz's original idea of "betting" as a means of "predicting" membership in a language, without being tied to a fixed order of instances that one tries to predict, or to a fixed order of how one goes about gathering information on the language. We have shown some aspects in which betting games are robust and well-behaved. We also contend that some current defects in the theory of betting games, notably the lack of a finite-unions theorem pending the status of pseudorandom generators, trade off with lacks in the resource-bounded measure theory, such as being tied to the lexicographic ordering of strings.

The main open problems in this paper are interesting in connection with recent work by Impagliazzo and Wigderson [ImWi98] on the BPP vs. EXP problem. First we remark that the main result of [ImWi98] implies that either BPP = EXP or BPP has E-measure zero [Mel98]. Among the many measure statements in the last section that imply BPP $\neq$ EXP, the most constrained and easiest to attack seems to be item 4 in Corollary 6.5. Indeed, in the specific relevant case starting with the assumption BPP = EXP, one is given a nonadaptive E-betting game G and an E-martingale

d, and to obtain the desired contradiction that proves BPP $\neq$ EXP, one needs only construct an EXP-betting game G' that covers $S^\infty[G] \cup S^\infty[d]$. What we *can* obtain is a "randomized" betting game G'' that flips one coin at successive intervals of input lengths to decide whether to simulate G or d on that interval. (The intervals come from the proof of Theorem 6.4.) Any hypothesis that can derandomize this G'' implies BPP $\neq$ EXP. We do not know whether the weak hypotheses considered in [ImWi98], some of them shown to follow from BPP $\neq$ EXP itself, are sufficient to do this.

Stepping back from trying to prove BPP $\neq$ EXP outright or trying to prove that these measure statements are equivalent to BPP $\neq$ EXP, we also have the problem of narrowing the gap between BPP $\neq$ EXP and the sufficient condition EXP $\neq$ MA used in our results. Moreover, does EXP $\neq$ MA suffice to make the $\leq_T^p$-autoreducible sets have E-measure zero? Does that suffice to simulate every betting game by a martingale of equivalent complexity? We also inquire whether there exist oracles relative to which EXP = MA, but strong pseudorandom generators still exist. Our work seems to open many opportunities to tighten the connections among pseudorandom generators, the structure of classes within EXP, and resource-bounded measure.

The kind of statistical sampling used to obtain martingales in Theorems 5.5 and 5.6 was originally applied to construct martingales from "natural proofs" in [RSC95]. The derandomization technique from [BFNW93] based on EXP $\neq$ MA that is used here is also applied in [BuMe98, KoLi98, LSW98]. "Probabilistic martingales" that can use this sampling to simulate betting games are formalized and studied in [ReSi98]. This paper also starts the task of determining how well the betting game and random-sampling ideas work for measures on classes below E. Even straightforward attempts to carry over Lutz's definitions to classes below E run into difficulties, as described in [May94t] and [AlSt94, AlSt95]. We look toward further applications of our ideas in lower-level complexity classes.

Acknowledgments. The authors especially thank Klaus Ambos-Spies, the late Ron Book, and Jack Lutz for organizing a special Schloss Dagstuhl workshop in July 1996, where preliminary versions of results and ideas in this paper were presented and extensively discussed. We thank the STACS '98 referees and the referees of this journal paper for helpful comments—one of the latter gave us many insightful notes that helped us steer away from errors and ambiguities in this final version.

REFERENCES

[AlSt94] E. Allender and M. Strauss, *Measure on Small Complexity Classes, with Applications for BPP*, Tech. Report DIMACS TR 94-18, Rutgers University and DIMACS, Piscataway, NJ, 1994.

[AlSt95] E. Allender and M. Strauss, *Measure on P: Robustness of the notion*, in Proceedings of the 20th International Symposium on Mathematical Foundations of Computer Science, Prague, Czech Republic, Lecture Notes in Comput. Sci. 969, Springer-Verlag, Berlin, 1995, pp. 129–138.

[Amb84] K. Ambos-Spies, *P-mitotic sets*, in Logic and Machines: Decision Problems and Complexity, Lecture Notes in Comput. Sci. 177, E. Börger, G. Hasenjäger, and D. Roding, eds., Springer-Verlag, Berlin, New York, 1984, pp. 1–23.

[ALM98] K. Ambos-Spies, S. Lempp, and G. Mainhardt, *Randomness vs. completeness: On the diagonalization strength of resource-bounded random sets*, in Proceedings of the 23rd International Symposium on Mathematical Foundations of Computer Science, Brno, Czech Republic, Lecture Notes in Comput. Sci. 1450, Springer-Verlag, Berlin, 1998, pp. 465–473.

[Bab85] L. Babai, *Trading group theory for randomness*, in Proceedings of the 17th Annual ACM Symposium on the Theory of Computing, Providence, RI, 1985, pp. 421–429.

[BFNW93] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson, *BPP has subexponential*

time simulations unless EXPTIME has publishable proofs, Comput. Complexity, 3 (1993), pp. 307–318.

[BaMo88] L. Babai and S. Moran, *Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[BlMi84] M. Blum and S. Micali, *How to generate cryptographically sequences of pseudo-random bits*, SIAM J. Comput., 13 (1984), pp. 850–864.

[Bon99] D. Boneh, *Twenty years of attacks on the RSA cryptosystem*, Notices Amer. Math. Soc., 46 (1999), pp. 203–213.

[BFT95] H. Buhrman, L. Fortnow, and L. Torenvliet, *Using autoreducibility to separate complexity classes*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 520–527.

[BFvMT98] H. Buhrman, L. Fortnow, D. van Melkebeek, and L. Torenvliet, *Separating complexity classes using autoreducibility*, SIAM J. Comput., 29 (2000), pp. 1497–1520. This is the journal version of [BFT95].

[BuLo00] H. Buhrman and L. Longpré, *Compressibility and resource bounded measure*, in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Grenoble, France, Lecture Notes in Comput. Sci. 1046, Springer-Verlag, Berlin, 1996, pp. 13–24.

[BuMe98] H. Buhrman and D. van Melkebeek, *Hard sets are hard to find*, in Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 170–181.

[Hel86] F. Heller, *On relativized exponential and probabilistic complexity classes*, Inform. and Control, 71 (1986), pp. 231–243.

[Imp99pc] R. Impagliazzo, *Very Strong One-Way Functions and Pseudo-Random Generators Exist Relative to a Random Oracle*, manuscript, personal communication, 1999.

[ImWi97] R. Impagliazzo and A. Wigderson, *$P = BPP$ if E requires exponential circuits: Derandomizing the XOR lemma*, in Proceedings of the 29th Annual ACM Symposium on the Theory of Computing, El Paso, TX, 1997, pp. 220–229.

[ImWi98] R. Impagliazzo and A. Wigderson, *Randomness vs. time: De-randomization under a uniform assumption*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 734–743.

[JuLu95] D. Juedes and J. Lutz, *Weak completeness in E and E_2*, Theoret. Comput. Sci., 143 (1995), pp. 149–158.

[Kan82] R. Kannan, *Circuit-size lower bounds and reducibility to sparse sets*, Inform. and Control, 55 (1982), pp. 40–56.

[KoLi98] J. Köbler and W. Lindner, *On the resource bounded measure of $P/poly$*, in Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 182–185.

[LSW98] W. Lindner, R. Schuler, and O. Watanabe, *Resource bounded measure and learnability*, in Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 261–270.

[Lov69] D. W. Loveland, *A variant of the Kolmogorov concept of complexity*, Inform. and Control, 15 (1969), pp. 510–526.

[Lutz92] J. Lutz, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 44 (1992), pp. 220–258.

[Lutz97] J. Lutz, *The quantitative structure of exponential time*, in Complexity Theory Retrospective II, L. Hemaspaandra and A. Selman, eds., Springer-Verlag, New York, 1997, pp. 225–260.

[May94t] E. Mayordomo, *Contributions to the Study of Resource-Bounded Measure*, Ph.D. thesis, Universidad Polytécnica de Catalunya, Barcelona, 1994.

[NiWi94] N. Nisan and A. Wigderson, *Hardness versus randomness*, J. Comput. System Sci., 49 (1994), pp. 149–167.

[PiFi79] N. Pippenger and M. Fischer, *Relations among complexity measures*, J. ACM, 26 (1979), pp. 361–381.

[RaRu97] A. Razborov and S. Rudich, *Natural proofs*, J. Comput. System Sci., 55 (1997), pp. 24–35.

[ReSi98] K. Regan and D. Sivakumar, *Probabilistic martingales and BPTIME classes*, in Proceedings of the 13th Annual IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 186–200.

[RSC95] K. Regan, D. Sivakumar, and J.-Y. Cai, *Pseudorandom generators, measure theory, and natural proofs*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 26–35.

[Mel98] D. van Melkebeek, *On the Measure of BPP*, Tech. Report TR-98-07, Department of Computer Science, University of Chicago, Chicago, IL, 1998.

[Mel99] D. van Melkebeek, *Randomness and Completeness in Computational Complexity*, Ph.D. thesis, University of Chicago, Chicago, IL, 1999. Available as UC CS Tech. Report TR 99-04.

[Zim95] M. Zimand, *On Randomized Cryptographic Primitives*, Tech. Report TR586, University of Rochester, Rochester, NY, 1995.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 602–624

CAVITY MATCHINGS, LABEL COMPRESSIONS, AND UNROOTED EVOLUTIONARY TREES*

MING-YANG KAO†, TAK-WAH LAM‡, WING-KIN SUNG‡, AND HING-FUNG TING‡

Abstract. We present an algorithm for computing a maximum agreement subtree of two unrooted evolutionary trees. It takes $O(n^{1.5}\log n)$ time for trees with unbounded degrees, matching the best known time complexity for the rooted case. Our algorithm allows the input trees to be mixed trees, i.e., trees that may contain directed and undirected edges at the same time. Our algorithm adopts a recursive strategy exploiting a technique called label compression. The backbone of this technique is an algorithm that computes the maximum weight matchings over many subgraphs of a bipartite graph as fast as it takes to compute a single matching.

Key words. computational biology, evolutionary trees, all-cavity maximum weight matchings, label compressions, unrooted trees, mixed trees

AMS subject classifications. 05C05, 05C85, 05C90, 68Q25, 92B05

PII. S0097539797332275

1. Introduction.

An *evolutionary* tree is one whose leaves are labeled with distinct symbols representing species. Evolutionary trees are useful for modeling the evolutionary relationship of species [1, 4, 6, 16, 17, 25]. An *agreement subtree* of two evolutionary trees is an evolutionary tree that is also a topological subtree of the two given trees. A *maximum* agreement subtree is one with the largest possible number of leaves. Different models about the evolutionary relationship of the same species may result in different evolutionary trees. A fundamental problem in computational biology is to determine how much two models of evolution have in common. To a certain extent, this problem can be solved by computing a maximum agreement subtree of two given evolutionary trees [12].

Algorithms for computing a maximum agreement subtree of two unrooted evolutionary trees as well as two rooted trees have been studied intensively in the past few years. The unrooted case is more difficult than the rooted case. There is indeed a linear-time reduction from the rooted case to the unrooted one, but the reverse is not known. Steel and Warnow [24] gave the first polynomial-time algorithm for unrooted trees, which runs in $O(n^{4.5}\log n)$ time. Farach and Thorup reduced the time to $O(n^{2+o(1)})$ for unrooted trees [10] and $O(n^{1.5}\log n)$ for rooted trees [11]. For the unrooted case, the time was improved by Lam, Sung, and Ting [22] to $O(n^{1.75+o(1)})$. Algorithms that work well for rooted trees with degrees bounded by a constant have also been revealed recently. The algorithm of Farach, Przytycka, and Thorup [9] takes $O(n\log^3 n)$ time, and that of Kao [20] takes $O(n\log^2 n)$ time. Cole and Hariharan [7]

*Received by the editors January 1, 1998; accepted for publication (in revised form) October 6, 1999; published electronically June 27, 2000. A preliminary version appeared as part of *General techniques for comparing unrooted evolutionary trees*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 54–65, and as part of *All-cavity maximum matchings*, in Proceedings of the 8th Annual International Symposium on Algorithms and Computation, 1997, pp. 364–373.

http://www.siam.org/journals/sicomp/30-2/33227.html

†Department of Computer Science, Yale University, New Haven, CT 06520 (kao-ming-yang@cs.yale.edu). This research was supported in part by NSF grant CCR-9531028.

‡Department of Computer Science and Information Systems, The University of Hong Kong, Hong Kong (twlam@csis.hku.hk, wksung@csis.hku.hk, hfting@csis.hku.hk). This research was supported in part by Hong Kong RGC grant HKU-7027/98E.

gave an $O(n \log n)$-time algorithm for the case where the input is further restricted to binary rooted trees.

This paper presents an algorithm for computing a maximum agreement subtree of two unrooted trees. It takes $O(n^{1.5} \log n)$ time for trees with unbounded degrees, matching the best known time complexity for the rooted case [11]. If the degrees are bounded by a constant, the running time is only $O(n \log^4 n)$. We omit the details of this reduction since Przytycka [23] recently devised an $O(n \log n)$-time algorithm for the same case.

Our algorithm allows the input trees to be *mixed* trees, i.e., trees that may contain directed and undirected edges at the same time [15, 18]. Such trees can handle a broader range of information than rooted and unrooted trees. To simplify the discussion, this paper focuses on unrooted trees. Our subtree algorithm adopts a conceptually simple recursive strategy exploiting a novel technique called *label compression*. This technique enables our algorithm to process overlapping subtrees iteratively while keeping the total tree size very close to the original input size. Label compression builds on an unexpectedly fast algorithm for the *all-cavity maximum weight matching* problem [21], which asks for the weight of a maximum weight matching in $G - \{u\}$ for each node u of a bipartite graph G with integer edge weights. If G has n nodes, m edges and maximum edge weight N, the algorithm takes $O(\sqrt{n}m \log(nN))$ time, which matches the best known time bound for computing a single maximum weight matching of G, due to Gabow and Tarjan [13].

In section 2, we solve the all-cavity matching problem. In section 3, we formally define maximum agreement subtrees and outline our recursive strategy for computing them. We describe label compression in section 4, detail our subtree algorithm in section 5, and discuss how to compute auxiliary information for label compression in sections 6 and 7. We conclude by extending the subtree algorithm to mixed trees in section 8.

2. All-cavity maximum weight matching. Let $G = (X, Y, E)$ be a bipartite graph with n nodes and m edges where each edge (u, v) has a positive integer weight $w(u, v) \leq N$. Let $\mathrm{mwm}(G)$ denote the weight of a maximum weight matching in G. The all-cavity matching problem asks for $\mathrm{mwm}(G - \{u\})$ for all $u \in X \cup Y$. A naive approach to solve this problem is to compute $\mathrm{mwm}(G - \{u\})$ separately for each u using the fastest algorithm for computing a single maximum weight matching [13], thus taking $O(n^{1.5} m \log(nN))$ total time. A main finding of this paper is that the matchings in different subgraphs $G - \{u\}$ are closely related and can be represented succinctly. From this representation, we can solve the problem in $O(\sqrt{n}m \log(nN))$ time. By symmetry, we detail only how to compute $\mathrm{mwm}(G - \{u\})$ for all $u \in X$. Below we assume $m \geq n/2$; otherwise, we remove the degree-zero nodes and work on the smaller resultant graph.

A node v of G is *matched* by a matching of G if v is an endpoint of an edge in the matching. In the remainder of this section, let M be a fixed maximum weight matching of G; also let $w(H)$ be the total weight of a set H of edges. An *alternating path* is a simple path P in G such that (1) P starts with an edge in M; (2) the edges of P alternate between M and $E - M$; and (3) if P ends at an edge $(u, v) \notin M$, then v is not matched by M. An *alternating cycle* is a simple cycle C in G whose edges alternate between M and $E - M$. P (respectively, C) can *transform* M to another matching $M' = P \cup M - P \cap M$ (respectively, $C \cup M - C \cap M$). The *net change* induced by P, denoted by $\Delta(P)$, is $w(M') - w(M)$, i.e., the total weight of the edges of P in $E - M$ minus that of the edges of P in M. The *net change* induced by C is

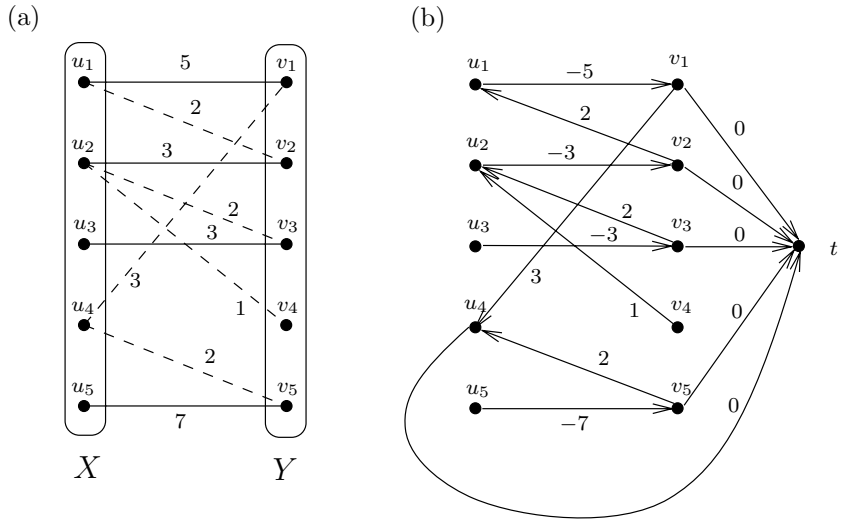


FIG. 1. (a) *a bipartite graph G;* (b) *the corresponding directed graph D.*

defined similarly.

The next lemma divides the computation of $\mathrm{mwm}(G - \{u\})$ into two cases.

LEMMA 2.1. *Let $u \in X$.*

1. *If u is not matched by M, then M is also a maximum weight matching in $G - \{u\}$ and $\mathrm{mwm}(G - \{u\}) = \mathrm{mwm}(G)$.*
2. *If u is matched by M, then G contains an alternating path P starting from u, which can transform M to a maximum weight matching in $G - \{u\}$.*

Proof. Statement 1 is straightforward. To prove statement 2, let M' be a maximum weight matching in $G - \{u\}$. Consider the edges in $M \cup M' - M \cap M'$. They form a set S of alternating paths and cycles. Since u is matched by M but not by M', u is of degree 1 in $M \cup M' - M \cap M'$. Let P be the alternating path in S with u as an endpoint. Let M'' be the matching obtained by transforming M only with P. Since u is not matched by M'', M'' is a matching in $G - \{u\}$. M' can be obtained by further transforming M'' with the remaining alternating paths and cycles in S. The net change induced by each of these alternating paths and cycles is nonpositive; otherwise, such a path or cycle can improve M and we obtain a contradiction. Therefore, $w(M'') \geq w(M')$, i.e., both M' and M'' are maximum weight matchings in $G - \{u\}$. □

By Lemma 2.1(2), we can compute $\mathrm{mwm}(G - \{u\})$ for any $u \in X$ matched by M by finding the alternating path starting from u with the largest net change. Below we construct a directed graph D, which enables us to identify such an alternating path for every node easily. The node set of D is $X \cup Y \cup \{t\}$, where t is a new node. The edge set of D is defined as follows; see Figure 1 for an example.

- If $x \in X$ is not matched by M, D has an edge from x to t with weight zero.
- If $y \in Y$ is matched by M, D has an edge from y to t with weight zero.
- If M has an edge (x, y) where $x \in X$ and $y \in Y$, D has an edge from x to y with weight $-w(x, y)$.
- If $E - M$ has an edge (x, y) where $x \in X$ and $y \in Y$, D has an edge from y to x with weight $w(x, y)$.

Note that D has $n + 1$ nodes and at most $n + m$ edges. The weight of each edge

in D is an integer in $[-N, N]$.

LEMMA 2.2.

1. *D contains no positive-weight cycle.*
2. *Each alternating path P in G that starts from $u \in X$ corresponds to a simple path Q in D from u to t, and vice versa. Also, $\Delta(P) = w(Q)$.*
3. *For each $u \in X$ matched by M, $\mathrm{mwm}(G - \{u\})$ is the sum of $\mathrm{mwm}(G)$ and the weight of the longest path in D from u to t.*

Proof. Statement 1. Consider a simple cycle $C = u_1, u_2, \ldots, u_k, u_1$ in D. Since t has no outgoing edges, no u_i equals t. By the definition of D, C is also an alternating cycle in G. Therefore, $w(C)$ is the net change induced by transforming M with C. Since M is a maximum weight matching in G, this net change is nonpositive.

Statement 2. Consider an alternating path $P = u, u_1, u_2, \ldots, u_k$ in G starting from u. In D, P is also a simple path. If $u_k \in X$, then u_k is not matched by M, and D contains the edge (u_k, t). If $u_k \in Y$, then u_k is matched by M, and D again contains the edge (u_k, t). Therefore, D contains the simple path $Q = u, u_1, u_2, \ldots, u_k, t$. The weight of Q is $\Delta(P)$. The reverse direction of the statement is straightforward.

Statement 3. This statement follows from Lemma 2.1(2) and Statement 2. □

THEOREM 2.3. *Given G, we can compute $\mathrm{mwm}(G - \{u\})$ for all nodes $u \in G$ in $O(\sqrt{n}m \log(nN))$ time.*

Proof. By symmetry and Lemmas 2.1(1) and 2.2(3), we compute $\mathrm{mwm}(G - \{u\})$ for all $u \in X$ as follows.

1. Compute a maximum weight matching M of G.
2. Construct D as above and find the weights of its longest paths to t.
3. For each $u \in X$, if u is matched by M, then $\mathrm{mwm}(G - \{u\})$ is the sum of $\mathrm{mwm}(G)$ and the weight of the longest path from u to t in D; otherwise, $\mathrm{mwm}(G - \{u\}) = \mathrm{mwm}(G)$.

Step 1 takes $O(\sqrt{n}m \log(nN))$ time. At step 2, constructing D takes $O(n + m)$ time, and the single-destination longest paths problem takes $O(\sqrt{n}m \log N)$ time [14]. Step 3 takes $O(n)$ time. Thus, the total time is $O(\sqrt{n}m \log(nN))$. □

3. The main result. This section gives a formal definition of maximum agreement subtrees and an overview of our new subtree algorithm.

3.1. Basics. Throughout the remainder of this paper, unrooted trees are denoted by U or X, and rooted trees by T, W, or R. A node of degree 0 or 1 is a *leaf*; otherwise, it is *internal*. Adopted to avoid technical trivialities, this definition is somewhat nonstandard in that if the root of a rooted tree is of degree 1, it is also a leaf.

For an unrooted tree U and a node $u \in U$, let U^u denote the rooted tree constructed by rooting U at u. For a rooted tree T and a node $v \in T$, let T^v denote the *rooted subtree* of T that comprises v and its descendents. Similarly, for a node $v \in U^u$, U^{uv} is the rooted subtree of U^u rooted at v, which is also called a *rooted subtree* of U.

An *evolutionary* tree is a tree whose leaves are labeled with distinct symbols. Let T be a rooted evolutionary tree with leaves labeled over a set L. A label subset $L' \subseteq L$ *induces* a subtree of T, denoted by $T|L'$, whose nodes are the leaves of T labeled over L' as well as the least common ancestors of such leaves in T, and whose edges preserve the ancestor-descendent relationship of T. Consider two rooted evolutionary trees T_1 and T_2 labeled over L. Let T_1' be a subtree of T_1 induced by some subset of L. We similarly define T_2' for T_2. If there exists an isomorphism between T_1' and T_2' mapping each leaf in T_1' to one in T_2' with the same label, then T_1'

and T_2' are each called *agreement subtrees* of T_1 and T_2. Note that this isomorphism is unique. Consider any nodes $u \in T_1$ and $v \in T_2$. We say that u *is mapped to* v *in* T_1' and T_2' if this isomorphism maps u to v. A *maximum* agreement subtree of T_1 and T_2 is one containing the largest possible number of labels. Let $\text{mast}(T_1, T_2)$ denote the number of labels in such a tree. A *maximum agreement subtree* of two unrooted evolutionary trees U_1 and U_2 is one with the largest number of labels among the maximum agreement subtrees of U_1^u and U_2^v over all nodes $u \in U_1$ and $v \in U_2$. Let

$$\text{mast}(U_1, U_2) = \max\{\text{mast}(U_1^u, U_2^v) \mid u \in U_1, v \in U_2\}. \tag{3.1}$$

Remark. The nodes u (or v) can be restricted to internal nodes when the trees have at least three nodes. We can also generalize the above definition to handle a pair of rooted tree and unrooted tree (T, U). That is, $\text{mast}(T, U)$ is defined to be $\max\{\text{mast}(T, U^v) \mid v \in U\}$.

3.2. Our subtree algorithm. The next theorem is our main result. The *size* $|U|$ (or $|T|$) of an unrooted tree U (or a rooted tree T) is its node count.

THEOREM 3.1. *Let U_1 and U_2 be two unrooted evolutionary trees. We can compute* $\text{mast}(U_1, U_2)$ *in* $O(N^{1.5} \log N)$ *time, where* $N = \max\{|U_1|, |U_2|\}$.

We prove Theorem 3.1 by presenting our algorithm in a top-down manner with an outline here. As in previous work, our algorithm only computes $\text{mast}(U_1, U_2)$ and can be augmented to report a corresponding subtree. It uses graph separators. A *separator* of a tree is an internal node whose removal divides the tree into connected components each containing at most half of the tree's nodes. Every tree that contains at least three nodes has a separator, which can be found in linear time.

If U_1 or U_2 has at most two nodes, $\text{mast}(U_1, U_2)$ as defined in (3.1) can easily be computed in $O(N)$ time. Otherwise, both trees have at least three nodes each, and we can find a separator x of U_1. We then consider three cases.

Case 1. In some maximum agreement subtree of U_1 and U_2, the node x is mapped to a node $y \in U_2$. In this case, $\text{mast}(U_1, U_2) = \text{mast}(U_1^x, U_2)$. To compute $\text{mast}(U_1^x, U_2)$, we might simply evaluate $\text{mast}(U_1^x, U_2^y)$ for different y in U_2. This approach involves solving the mast problem for $\Theta(N)$ different pairs of rooted trees and introduces much redundant computation. For example, consider a rooted subtree R of U_2. For all $y \in U_2 - R$, R is a common subtree of U_2^y. Hence, R is examined repeatedly in the computation of $\text{mast}(U_1^x, U_2^y)$ for these y. To speed up the computation, we devise the technique of label compression in section 4 to elicit sufficient information between U_1^x and R so that we can compute $\text{mast}(U_1^x, U_2^y)$ for all $y \in U_2 - R$ without examining R. This leads to an efficient algorithm for handling Case 1; the time complexity is stated in the following lemma.

LEMMA 3.2. *Assume that U_1 and U_2 have at least three nodes each. Given an internal node $x \in U_1$, we can compute* $\text{mast}(U_1^x, U_2)$ *in* $O(N^{1.5} \log N)$ *time.*

Proof. See section 4 to section 7. □

Case 2. In some maximum agreement subtree of U_1 and U_2, two certain nodes v_1 and v_2 of U_1 are mapped to nodes in U_2, and x is on the path in U_1 between v_1 and v_2. This case is similar to Case 1. Let $\tilde{U}_2$ be the tree constructed by adding a dummy node in the middle of every edge in U_2. Then, $\text{mast}(U_1, U_2) = \text{mast}(U_1^x, \tilde{U}_2^y)$ for some dummy node y in $\tilde{U}_2$. Thus, $\text{mast}(U_1, U_2) = \text{mast}(U_1^x, \tilde{U}_2)$. As in Case 1, $\text{mast}(U_1^x, \tilde{U}_2)$ can be computed in $O(N^{1.5} \log N)$ time.

Case 3. Neither Case 1 nor Case 2 holds. Let $U_{1,1}, U_{1,2}, \ldots, U_{1,b}$ be the evolutionary trees formed by the connected components of $U_1 - \{x\}$. Let $J_1, \ldots, J_b$ be the sets of labels in these components, respectively. Then, a maximum agreement subtree of

```
/* U_1 and U_2 are unrooted trees. */
mast(U_1, U_2)
  find a separator x of U_1;
  construct Ũ_2 by adding a dummy node w at the middle of each edge (u, v) in
  U_2;
  val = mast(U_1^x, U_2);
  val' = mast(U_1^x, Ũ_2);
  let U_{1,1}, U_{1,2}, ..., U_{1,b} be the connected components of U_1 − {x};
  for all i ∈ [1, b], let J_i be the set of labels of U_{1,i};
  for all i ∈ [1, b], set val_i = mast(U_{1,i}, U_2|J_i)};
  return max{val, val', max_{1≤i≤b} val_i};
```

FIG. 2. *Algorithm for computing* $\mathrm{mast}(U_1, U_2)$.

U_1 and U_2 is labeled over some J_i. Therefore, $\mathrm{mast}(U_1, U_2) = \max\{\mathrm{mast}(U_{1,i}, U_2|J_i) \mid i \in [1, b]\}$, and we compute each $\mathrm{mast}(U_{1,i}, U_2|J_i)$ recursively.

Figure 2 summarizes the steps for computing $\mathrm{mast}(U_1, U_2)$. Here we analyze the time complexity $T(N)$ based on Lemma 3.2. Cases 1 and 2 each take $O(N^{1.5} \log N)$ time. Let $N_i = |U_{1,i}|$. Then Case 3 takes $\sum_{i\in[1,b]} T(N_i)$ time. By recursion,

$$T(N) = O(N^{1.5} \log N) + \sum_{i\in[1,b]} T(N_i).$$

Since x is a separator of U_1, $N_i \leq \frac{N}{2}$. Then, since $\sum_{i\in[1,b]} N_i \leq N$, $T(N) = O(N^{1.5} \log N)$ [5, 19] and the time bound in Theorem 3.1 follows. To complete the proof of Theorem 3.1, we devote section 4 through section 7 to proving Lemma 3.2.

4. Label compressions. To compute a maximum agreement subtree, our algorithm recursively processes overlapping subtrees of the input trees. The technique of label compression compresses overlapping parts of such subtrees to reduce their total size. We define label compressions with respect to a rooted subtree in section 4.1 and with respect to two label-disjoint rooted subtrees in section 4.2. We do not use label compression with respect to three or more trees.

As a warm-up, let us define a concept called *subtree shrinking*, which is a primitive form of label compression. Let T be a rooted tree. Let R be a rooted subtree of T. Let $T \ominus R$ denote the rooted tree obtained by replacing R with a leaf γ. We say that γ is a *shrunk* leaf. The other leaves are *atomic* leaves. Similarly, for two label-disjoint rooted subtrees R_1 and R_2 of T, let $T \ominus (R_1, R_2)$ denote the rooted tree obtained by replacing R_1 and R_2 with *shrunk* leaves γ_1 and γ_2, respectively. We extend these notions to an unrooted tree U and define $U \ominus R$ and $U \ominus (R_1, R_2)$ similarly.

4.1. Label compression with respect to one rooted subtree. Let T be a rooted tree. Let v be a node in T and u an ancestor of v. Let P be the path of T from u to v. A node *lies* between u and v if it is in P but differs from u and v. A subtree of T is *attached* to u if it is some T^w where w is a child of u. A subtree of T *hangs* between u and v if it is attached to some node lying between u and v, but its root is not in P and is not v.

We are now ready to define the concept of label compression. Let T and R be rooted evolutionary trees labeled over L and K, respectively. The *compression* of T with respect to R, denoted by $T \otimes R$, is a tree constructed by affixing extra nodes to

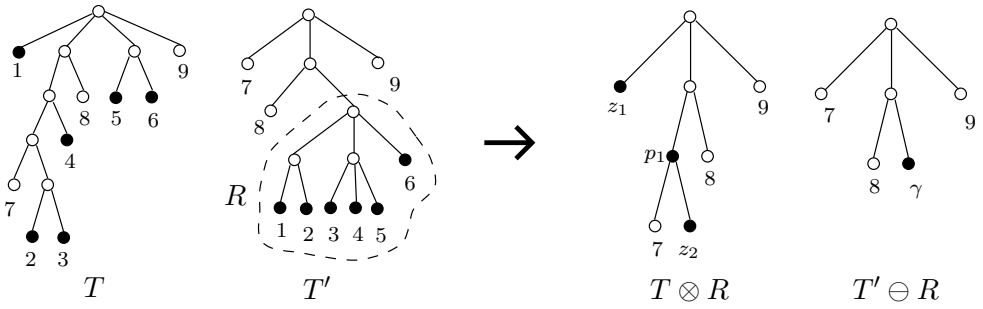


FIG. 3. *An example of label compression.*

$T|(L-K)$ with the following steps; see Figure 3 for an example. Consider each node y in $T|(L-K)$, let x be its parent in $T|(L-K)$.

- Let $\mathcal{A}(T,K,y)$ denote the set of subtrees of T that are attached to y and whose leaves are all labeled over K. If $\mathcal{A}(T,K,y)$ is nonempty, compress all the trees in $\mathcal{A}(T,K,y)$ into a single node z_1 and attach it to y.
- Let $\mathcal{H}(T,K,y)$ denote the set of subtrees of T that hang between x and y (by definition of $T|(L-K)$, these subtrees are all labeled over K). If $\mathcal{H}(T,K,y)$ is nonempty, compress the parents $p_1,\ldots,p_m$ of the roots of the trees in $\mathcal{H}(T,K,y)$ into a single node p_1, and insert it between x and y; also compress all the trees in $\mathcal{H}(T,K,y)$ into a single node z_2 and attach it to p_1.

The nodes z_1, z_2, and p_1 are called *compressed* nodes, and the leaves in $T\otimes R$ that are not compressed are *atomic* leaves.

We further store in $T\otimes R$ some auxiliary information about the relationship between T and R. For an internal node v in $T\otimes R$, let $\alpha(v) = \mathrm{mast}(T^v, R)$. For a compressed leaf v in $T\otimes R$, if it is compressed from a set of subtrees $T^{v_1},\ldots,T^{v_s}$, let $\alpha(v) = \max\{\mathrm{mast}(T^{v_1},R),\ldots,\mathrm{mast}(T^{v_s},R)\}$.

Let T_1 and T_2 be two rooted evolutionary trees. Assume T_2 contains a rooted subtree R. Given $T_1\otimes R$, we can compute $\mathrm{mast}(T_1,T_2)$ without examining R. We first construct $T_1\ominus R$ by replacing R of T_2 with a shrunk leaf and then compute $\mathrm{mast}(T_1,T_2)$ from $T_1\otimes R$ and $T_2\ominus R$. To further our discussion, we next generalize the definition of maximum agreement subtree for a pair of trees that contain compressed leaves and a shrunk leaf, respectively.

Let $W_1 = T_1\otimes R$ and $W_2 = T_2\ominus R$. Let γ be the shrunk leaf in W_2. We define an agreement subtree of W_1 and W_2 similar to that of ordinary evolutionary trees. An atomic leaf must still be mapped to an atomic leaf with the same label. However, the shrunk leaf γ of W_2 can be mapped to any internal node or compressed leaf v of W_1 as long as $\alpha(v) > 0$. The size of an agreement subtree is the number of its atomic leaves, plus $\alpha(v)$ if γ is mapped to a node $v \in W_1$. A *maximum* agreement subtree of W_1 and W_2 is one with the largest size. Let $\mathrm{mast}(W_1,W_2)$ denote the size of such a subtree. The following lemma is the cornerstone of label compression.

LEMMA 4.1. $\mathrm{mast}(T_1,T_2) = \mathrm{mast}(W_1,W_2)$.

Proof. It follows directly from the definition. □

We can compute $\mathrm{mast}(W_1,W_2)$ as if W_1 and W_2 were ordinary rooted evolutionary trees [9, 11, 20] with a special procedure on handling the shrunk leaf. The time complexity is stated in the following lemma. Let $n = \max\{|W_1|,|W_2|\}$ and $N = \max\{|T_1|,|T_2|\}$.

LEMMA 4.2. *Suppose that all the auxiliary information of W_1 has been given.*

Then $\text{mast}(W_1, W_2)$ *can be computed in* $O(n^{1.5} \log N)$ *time, and afterwards we can retrieve* $\text{mast}(W_1^v, W_2)$ *for any node* $v \in W_1$ *in* $O(1)$ *time.*

Proof. We adapt Farach and Thorup's rooted subtree algorithm [11] to compute $\text{mast}(W_1, W_2)$. Details are given in Appendix A. □

We demonstrate a scenario where label compression speeds up the computation of $\text{mast}(U_1^x, U_2)$ for Lemma 3.2. Suppose that we can identify a rooted subtree R of U_2 such that x is mapped to a node outside R, i.e., we can reduce (3.1) to

$$\text{mast}(U_1^x, U_2) = \max\{\text{mast}(U_1^x, U_2^y) \mid y \text{ is an internal node not in } R\}. \tag{4.1}$$

Note that every U_2^y contains R as a common subtree. To avoid overlapping computation on R, we construct $W = U_1^x \otimes R$ and $X = U_2 \ominus R$. Then $X^y = U_2^y \ominus R$ and from Lemma 4.1, $\text{mast}(U_1^x, U_2^y) = \text{mast}(W, X^y)$. We rewrite (4.1) as

$$\text{mast}(U_1^x, U_2) = \max\{\text{mast}(W, X^y) \mid y \text{ is an internal node of } X\}. \tag{4.2}$$

If R is large, then W and X are much smaller than U_1^x and U_2. Consequently, it is beneficial to compress U_1^x and compute $\text{mast}(U_1^x, U_2)$ according to (4.2).

4.2. Label compression with respect to two rooted subtrees. Let T, R_1, R_2 be rooted evolutionary trees labeled over L, K_1, K_2, respectively, where $K_1 \cap K_2 = \phi$. Let $K = K_1 \cup K_2$. The *compression* of T with respect to R_1 and R_2, denoted by $T \otimes (R_1, R_2)$, is a tree constructed from $T|(L - K)$ by the following two steps. For each node y and its parent x in $T|(L - K)$,

1. if $\mathcal{A}(T, K, y)$ is nonempty, compress all the trees in $\mathcal{A}(T, K, y)$ into a single leaf z and attach it to y; create and attach an auxiliary node $\bar{z}$ to y;
2. if $\mathcal{H}(T, K, y)$ is nonempty, compress the parents $p_1, \ldots, p_m$ of the roots of the subtrees in $\mathcal{H}(T, K, y)$ into a single node p_1 and insert it between x and y; compress the subtrees in $\mathcal{H}(T, K, y)$ into a single node z and attach it to p_1; create and insert an auxiliary node $\bar{p}_1$ between p_1 and y; create auxiliary nodes $\bar{z}$ and $\bar{\bar{z}}$ and attach them to p_1 and $\bar{p}_1$, respectively.

The nodes p_1 and z are *compressed* nodes of $T \otimes (R_1, R_2)$. The nodes $\bar{p}_1, \bar{z}$, and $\bar{\bar{z}}$ are *auxiliary* nodes. These nodes are added to capture the topology of T that is isomorphic with the subtrees R_1 and R_2 of T'.

We also store auxiliary information in $T \otimes (R_1, R_2)$. Let R^+ be the tree obtained by connecting R_1 and R_2 together with a node, which becomes the root of R^+.

Consider the internal nodes of $T \otimes (R_1, R_2)$. If v is an internal node inherited from $T|(L - K)$, then let $\alpha_1(v) = \text{mast}(T^v, R_1)$ and $\alpha_2(v) = \text{mast}(T^v, R_2)$. If p_1 and $\bar{p}_1$ are internal nodes compressed from some path $p_1, \ldots, p_m$ of T, then only p_1 stores the values $\alpha_1(p_1) = \text{mast}(T^{p_1}, R_1)$, $\alpha_2(p_1) = \text{mast}(T^{p_1}, R_2)$, and $\alpha_+(p_1) = \text{mast}(T^{p_1}, R^+)$.

We do not store any auxiliary information at the atomic leaves in $T \otimes (R_1, R_2)$. Consider the other leaves in $T \otimes (R_1, R_2)$ based on how they are created.

Case 1. Nodes $z, \bar{z}$ are leaves created with respect to $\mathcal{A}(T, K, y)$ for some node y in $T|(L - K)$. Let $\mathcal{A}(T, K, y) = \{T^{v_1}, \ldots, T^{v_k}\}$. We store the following values at z.

- $\alpha_1(z) = \max\{\text{mast}(T^{v_i}, R_1) \mid i \in [1, k]\}$, $\alpha_2(z) = \max\{\text{mast}(T^{v_i}, R_2) \mid i \in [1, k]\}$, $\alpha_+(z) = \max\{\text{mast}(T^{v_i}, R^+) \mid i \in [1, k]\}$;
- $\beta(z) = \max\{\text{mast}(T^{v_i}, R_1) + \text{mast}(T^{v_{i'}}, R_2) \mid T^{v_i}$ and $T^{v_{i'}}$ are distinct subtrees in $\mathcal{A}(T, K, y)\}$.

Case 2. Nodes $z, \bar{z}$, and $\bar{\bar{z}}$ are leaves created with respect to the subtrees in $\mathcal{H}(T, K, y) = \{T^{v_1}, \ldots, T^{v_k}\}$ for some node y in $T|(L - K)$. We store the following values at z:

- $\alpha_1(z)$, $\alpha_2(z)$, and $\alpha_+(z)$ as in Case 1;
- $\beta(z) = \max\{\text{mast}(T^{v_i}, R_1)+\text{mast}(T^{v_j}, R_2) \mid T^{v_i}$ and T^{v_j} are distinct subtrees in $\mathcal{H}(T, K, y)$ that are attached to the same node in $T\}$;
- $\beta_{1\succ 2}(z) = \max\{\text{mast}(T^{v_j}, R_1) + \text{mast}(T^{v_{j'}}, R_2) \mid (j, j') \in Z\}$ and $\beta_{2\succ 1}(z) = \max\{\text{mast}(T^{v_j}, R_2) + \text{mast}(T^{v_{j'}}, R_1) \mid (j, j') \in Z\}$, where $Z = \{(j, j') \mid T^{v_j}, T^{v_{j'}} \in \mathcal{H}(T, K, y)$ and the parent of v_j in T is a proper ancestor of the parent of $v_{j'}\}$.

Let T_1 and T_2 be rooted evolutionary trees. Let R_1 and R_2 be label-disjoint rooted subtrees of T_2. Let $W_1 = T \otimes (R_1, R_2)$ and $W_2 = T' \ominus (R_1, R_2)$. Below, we give the definition of a maximum agreement subtree of W_1 and W_2.

Let γ_1 and γ_2 be the two shrunk leaves in W_2 representing R_1 and R_2, respectively. Let y_c be the least common ancestor of γ_1 and γ_2 in W_2. Intuitively, in a pair of agreement subtrees (W_1', W_2') of W_1 and W_2, atomic leaves are mapped to atomic leaves, and shrunk leaves are mapped to internal nodes or leaves. Moreover, we allow W_2' to contain y_c as a leaf, which can be mapped to an internal node or leaf of W_1'. More formally, we require that there is an isomorphism between W_1' and W_2' satisfying the following conditions:

1. Every atomic leaf is mapped to an atomic leaf with the same label.
2. If W_2' contains y_c as a leaf and thus neither γ_1 nor γ_2 is found in W_2', then y_c is mapped to a node v with $\alpha_+(v) > 0$.
3. If only one of γ_1 and γ_2 exists in W_2', say γ_1, then it is mapped to a node v with $\alpha_1(v) > 0$.
4. If both γ_1 and γ_2 exist in W_2', then any of the following cases is permitted:
   - γ_1 and γ_2, respectively, are mapped to a compressed leaf z and its sibling $\bar{z}$ in W_1' with $\beta(z) > 0$.
   - γ_1 and γ_2, respectively, are mapped to a compressed leaf z and the accompanying auxiliary leaf $\bar{\bar{z}}$ in W_1' with $\beta_{1\succ 2}(z) > 0$, or the leaves $\bar{\bar{z}}$ and z in W_1' with $\beta_{2\succ 1}(z) > 0$.
   - γ_1 and γ_2, respectively, are mapped to two leaves or internal nodes v and w with $\alpha_1(v)$, $\alpha_2(w) > 0$.

The way we measure the size of W_1' and W_2' depends on their isomorphism. For example, if y_c is mapped to some node v in W_1', then the size is the total number of atomic leaves in W_1' plus $\alpha_+(v)$. More precisely, the size of W_1' and W_2' is defined to be the total number of atomic leaves in W_1' plus the corresponding α or β values depending on the isomorphism between W_1' and W_2'. A *maximum* agreement subtree of W_1 and W_2 is one with the largest possible size. Let $\text{mast}(W_1, W_2)$ denote the size of such a subtree. The following lemma, like Lemma 4.1, is also the cornerstone of label compression.

LEMMA 4.3. $\text{mast}(T_1, T_2) = \text{mast}(W_1, W_2)$.

Proof. It follows directly from the definition of $\text{mast}(W_1, W_2)$. □

Again, $\text{mast}(W_1, W_2)$ can be computed by adapting Farach and Thorup's rooted subtree algorithm [11]. The time complexity is stated in the following lemma. Let $n = \max\{|W_1|,|W_2|\}$ and $N = \max\{|T_1|, |T_2|\}$.

LEMMA 4.4. *Suppose that all the auxiliary information of W_1 has been given. Then we can compute* $\text{mast}(W_1, W_2)$ *in* $O(n^{1.5} \log N)$ *time. Afterwards we can retrieve* $\text{mast}(W_1^v, W_2)$ *for any* $v \in W$ *in* $O(1)$ *time.*

Proof. See Appendix A. □

5. Computing mast(U_1^x, U_2)—proof of Lemma 3.2. At a high level, we first apply label compression to the input instance (U_1^x, U_2). We then reduce the problem

```
/* W is a rooted tree with compressed leaves. X is unrooted with shrunk leaves. */
mast(W, X)
  let y be a separator of X;
  val = mast(W, X^y);
  if (X has at most one shrunk leaf) or (y lies between the two shrunk leaves)
  then
    new_subproblem(W, X, y);
    for each (W_i, X_i), val_i = mast(W_i, X_i);
  else
    let y' be the node on the path between the two shrunk leaves that is the closest to
    y;
    val = mast(W, X^{y'});
    new_subproblem(W, X, y');
    for each (W_i, X_i), set val_i = mast(W_i, X_i);
  return max{val, max_{i=1}^{b} val_i};


/* Generate new subproblems {(W_1, X_1), ..., (W_b, X_b)}. */
new_subproblem(W, X, y)
  let v_1, ..., v_b be the neighbors of y in X;
  for all i ∈ [1, b]
    let X_i be the unrooted tree formed by shrinking the subtree X^{v_i y} into a shrunk leaf;
    let W_i be the rooted tree formed by compressing W with respect to X^{v_i y};
  compute and store the auxiliary information in W_i for all i ∈ [1, b];
```

FIG. 4. *Algorithm for computing* $\mathrm{mast}(W, X)$.

to a number of smaller subproblems (W, X), each of which is similar to (U_1^x, U_2) and is solved recursively. For each (W, X) generated, X is a subtree of U_2 with at most two shrunk leaves, and W is a label compression of U_1^x with respect to some rooted subtrees of U_2 that are represented by the shrunk leaves of X. Also, W and X contain the same number of atomic leaves.

5.1. Recursive computation of mast(W, X). Our subtree algorithm initially sets $W = U_1^x$ and $X = U_2$. In general, $W = U_1^x \otimes R$ and $X = U_2 \ominus R$, or $W = U_1^x \otimes (R, R')$ and $X = U_2 \ominus (R, R')$ for some rooted subtrees R and R' of U_2. If W or X has at most two nodes, then $\mathrm{mast}(W, X)$ can easily be computed in linear time. Otherwise, both W and X each have at least three nodes. Let $N = \max\{|U_1|, |U_2|\}$ and $n = \max\{|W|, |X|\}$. Our algorithm first finds a separator y of X and computes $\mathrm{mast}(W, X)$ for the following two cases. The output is the larger of the two cases. Figure 4 outlines our algorithm.

Case 1. $\mathrm{mast}(W, X) = \mathrm{mast}(W, X^y)$. We root X at y and evaluate $\mathrm{mast}(W, X^y)$. By Lemma 4.4, this takes $O(n^{1.5} \log N)$ time.

Case 2. $\mathrm{mast}(W, X) = \mathrm{mast}(W, X^z)$ for some internal node $z \neq y$. We compute $\max\{\mathrm{mast}(W, X^z) \mid z \text{ is an internal node and } z \neq y\}$ by solving a set of subproblems $\{\mathrm{mast}(W_1, X_1), \ldots, \mathrm{mast}(W_b, X_b)\}$ such that their total size is n and $\max\{\mathrm{mast}(W, X^z) \mid z \text{ is an internal node and } z \neq y\} = \max\{\mathrm{mast}(W_i, X_i) \mid i \in [1, b]\}$. Moreover, our algorithm enforces the following properties:

- If X contains at most one shrunk leaf, every subproblem generated has size at most half that of X.
- If X has two shrunk leaves, at most one subproblem (W_{i_o}, X_{i_o}) has size

greater than half that of X, but X_{i_o} contains only one shrunk leaf. Thus, in the next recursion level, every subproblem spawned by (W_{i_o}, X_{i_o}) has size at most half that of X.

To summarize, whenever the recursion goes down by two levels, the size of a subproblem reduces by half.

The subproblems $\text{mast}(W_1, X_1), \ldots, \text{mast}(W_b, X_b)$ are formally defined as follows. Assume that the separator y has b neighbors in X, namely, $v_1, \ldots, v_b$. For each $i \in [1, b]$, let C_i be the connected component in $X - \{y\}$ that contains v_i. The size of C_i is at most half that of X. Intuitively, we would like to shrink the subtree $X^{v_i y}$ into a leaf, producing a smaller unrooted tree X_i. We first consider the simple case where X has at most one shrunk leaf. Then no C_i contains more than one shrunk leaf.

If C_i contains no shrunk leaf, then X_i contains only one shrunk leaf representing the subtree $X^{v_i y}$. Note that $X^{v_i y}$ corresponds to the subtree $U_2^{v_i y}$ in U_2 and $X_i = U_2 \ominus U_2^{v_i y}$. Let $W_i = U_1^x \otimes U_2^{v_i y}$.

If C_i contains one shrunk leaf γ_1, then X_i contains γ_1 as well as a new shrunk leaf representing the subtrees $X^{v_i y}$. The two subtrees are label-disjoint. Again, $X^{v_i y}$ corresponds to the subtree $U_2^{v_i y}$ in U_2. Assume that γ_1 corresponds to a subtree $U_2^{v' y'}$ in U_2. Then $X_i = U_2 \ominus (U_2^{v' y'}, U_2^{v_i y})$. Let $W_i = U_1^x \otimes (U_2^{v' y'}, U_2^{v_i y})$.

We now consider the case where X itself already has two shrunk leaves γ_1 and γ_2. If y lies on the path between γ_1 and γ_2, then no C_i contains more than one shrunk leaf and we define the smaller problem instances (W_i, X_i) as above. Otherwise, there is a C_i containing both γ_1 and γ_2. X_i as defined contains three compressed leaves, violating our requirement. In this case, we replace y with the node y' on the path between γ_1 and γ_2, which is the closest to y. Now, to compute $\text{mast}(W, X)$, we consider the two cases depending on whether the root of W is mapped to y' or not. Again, we first compute $\text{mast}(W, X^{y'})$. Then, we define the connected components C_i and the smaller problem instances (W_i, X_i) with respect to y'. Every X_i has at most two compressed leaves, but y' may not be a separator and we cannot guarantee that the size of every subproblem is reduced by half. However, there can exist only one connected component C_{i_o} with size larger than half that of X. Indeed, C_{i_o} is the component containing y. In this case, both γ_1 and γ_2 are not inside C_{i_o}, and X_{i_o} as defined contains only one compressed leaf. Thus, the subproblems that $\text{mast}(W_{i_0}, X_{i_0})$ spawns in the next recursion level each have size of at most half that of (W, X).

With respect to y or y', computing the topology of all X_i and W_i from X and W is straightforward; see section 5.2. Computing the auxiliary information in all W_i efficiently requires some intricate techniques, which are detailed in sections 6 and 7.

5.2. Computing the topology of compressed trees. The topology of X_i can be constructed from X by replacing the subtree $X^{v_i y}$ of X with a shrunk leaf. Let J and J_i be the sets of labels in X and X_i, respectively. For the trees W_i, recall that the definitions of W and the trees W_i are based on affixing some nodes to the trees $U_1^x|J$ and $U_1^x|J_i$, respectively. Observe that $W|J$ and $U_1^x|J$ have the same topology. Moreover, $W|J_i = (W|J)|J_i$ and $U_1^x|J_i = (U_1^x|J)|J_i$. Thus, $W|J_i$ and $U_1^x|J_i$ have the same topology. We can obtain $U_1^x|J_i$ by constructing $W|J_i$. Note that $J = \bigcup_{1 \le i \le b} J_i$ and all the label sets J_i are disjoint. We can construct all the trees $W|J_i$ from W in $O(n)$ time [7, 10]. Next, we show how to construct W_i from $W|J_i$ in time linear in the size of $W|J_i$. We only detail the case where X_i consists of two shrunk leaves. The case for one shrunk leaf is similar. The following procedure is derived directly from the definition of the compression of U_1^x with respect to two subtrees.

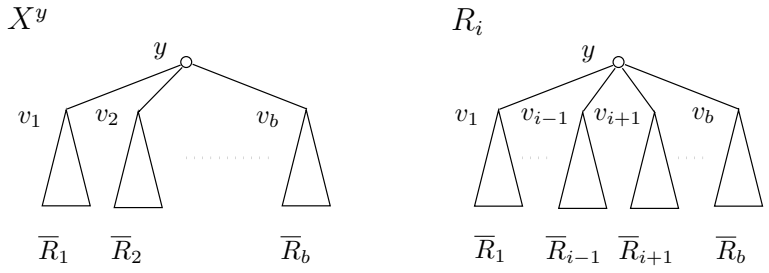


FIG. 5. *The structures of X^y and R_i.*

Let v be any node of $W|J_i$. If v is not the root, let u be the parent of v in $W|J_i$.

- If $\mathcal{A}(U_1^x, L-J_i, v)$ is nonempty or equivalently the degree of v in U_1^x is different from its degree in $W|J_i$, then attach auxiliary leaves z and $\bar{z}$ to v.
- If $\mathcal{H}(U_1^x, L-J_i, v)$ is nonempty or equivalently u is not the parent of v in U_1^x, then create a path between u and v consisting of two nodes p and $\bar{p}$, attach auxiliary leaves z and $\bar{z}$ to p, and attach $\bar{\bar{z}}$ to $\bar{p}$.

5.3. Time complexity of computing mast(W, X).

LEMMA 5.1. *We can compute* $\mathrm{mast}(W, X)$ *in* $O(n^{1.5}\log N)$ *time.*

Proof. Let $T(n)$ be the computation time of $\mathrm{mast}(W, X)$. The computation is divided into two cases. Case 1 of section 5.1 takes $O(n^{1.5}\log N)$ time. For Case 2, a set of subproblems $\{\mathrm{mast}(W_i, X_i) \mid i \in [1, b]\}$ is generated. As to be shown in sections 6 and 7, the time to prepare all these subproblems is also $O(n^{1.5}\log N)$. These subproblems, except possibly one, are each of size less than $n/2$. For the exceptional subproblem, say, $\mathrm{mast}(W_l, X_l)$, its computation is again divided into two cases. One case takes $O(n^{1.5}\log N)$ time. For the other case, another set of subproblems is generated in $O(n^{1.5}\log N)$ time. This time every such subproblem has size less than $n/2$. Let Σ be the set of all the subproblems generated in both steps. The total size of the subproblems in Σ is at most n, and

$$T(n) = O(n^{1.5}\log N) + \sum_{\mathrm{mast}(W', X') \in \Sigma} T(|X'|).$$

It follows that $T(n) = O(n^{1.5}\log N)$. ∎

By letting $W = U_1^x$ and $X = U_2$, we have proved Lemma 3.2. What remains is to show how to compute the auxiliary information stored in all W_i from (W, X) in $O(n^{1.5}\log N)$ time. Note that X contains at most two shrunk leaves. Depending on the number of shrunk leaves in X, we divide our discussion into sections 6 and 7.

6. Auxiliary information for X with no shrunk leaf. The case of X containing no shrunk leaf occurs only when the algorithm starts, i.e., $W = U_1^x$, $X = U_2$, and $N = n$. The subproblems $\mathrm{mast}(W_1, X_1), \ldots, \mathrm{mast}(W_b, X_b)$ spawned from (W, X) are defined by an internal node y in X, which is adjacent to the nodes $v_1, \ldots, v_b$. Let R_i and $\overline{R}_i$ denote the rooted subtrees $X^{v_i y}$ and $X^{y v_i}$, respectively. Note that the rooted tree X^y is composed of the subtrees $\overline{R}_1, \ldots, \overline{R}_b$. Also, $W_i = W \otimes R_i$ and $X_i = W \ominus R_i$. The total size of all $\overline{R}_i$ is at most n. Furthermore, each R_i is X^y with $\overline{R}_i$ removed; see Figure 5. This section discusses how to compute the auxiliary information required by each W_i in $O(n^{1.5}\log N)$ time.

6.1. Auxiliary information in the compressed leaves of W_i. Consider any compressed leaf v in W_i. Let S_v denote the set of subtrees from which v is compressed. Then, the auxiliary information to be stored in v is

$$\alpha(v) = \max\{\text{mast}(W^z, R_i) \mid W^z \in S_v\}. \tag{6.1}$$

Observe that for any $W^z \in S_v$, W^z contains no labels outside R_i. Thus, $\text{mast}(W^z, R_i) = \text{mast}(W^z, X^y)$ and we can rewrite (6.1) as

$$\alpha(v) = \max\{\text{mast}(W^z, X^y) \mid W^z \in S_v\}.$$

We use the rooted subtree algorithm of [11] to compute $\text{mast}(W, X^y)$ in $O(n^{1.5} \log N)$ time. Then, we can retrieve the value of $\text{mast}(W^z, X^y)$ for any node $z \in W$ in $O(1)$ time. To compute $\max\{\text{mast}(W^z, X^y) \mid W^z \in S_v\}$ efficiently, we assume that for any node $u \in W$, the subtrees attached to u are numbered consecutively, starting from 1. We consider a preprocessing for efficient retrieval of the following types of values:

- for some node $u \in W$ and some interval $[a, b]$, $\max\{\text{mast}(W^z, X^y) \mid W^z$ is a subtree attached to u and its number falls in $[a, b]\}$;
- for some path P of W, $\max\{\text{mast}(W^z, X^y) \mid W^z$ is a subtree attached to some node in $P\}$.

LEMMA 6.1. *Assume that we can retrieve* $\text{mast}(W^z, X^y)$ *for any* $z \in W$ *in* $O(1)$ *time. Then we can preprocess* W *and* X *and construct additional data structures in* $O(n \log^* n)$ *time so that any value of the above types can be retrieved in* $O(1)$ *time.*

Proof. We adapt preprocessing techniques for on-line product queries in [3]. □

With the preprocessing stated in Lemma 6.1, we can determine $\alpha(v)$ as follows. Note that S_v is either a subset of the subtrees attached to a node u in W or the set of subtrees attached to nodes on a particular path in W. In the former case, u is also a parent of v and S_v is partitioned into at most $d_u + 1$ intervals where d_u is the degree of u in W_i. From Lemma 6.1, $\alpha(v)$ can be found in $O(d_u + 1)$ time. Similarly, for the latter case, $\alpha(v)$ can be found in $O(1)$ time. Thus, the compressed leaves in W_i are processed in $O(|W_i|)$ time. Summing over all W_i, the time complexity is $O(n)$. Therefore, the overall computation time for preprocessing and finding auxiliary information in the leaves of all W_i is $O(n^{1.5} \log N)$.

6.2. Auxiliary information in the internal nodes of W_i. Consider any internal node v in W_i with $i \in [1, b]$. Our goal is to compute the auxiliary information $\alpha(v) = \text{mast}(W^v, R_i)$. Note that R_i may be of size $\Theta(n)$, and even computing one particular $\text{mast}(W^v, R_i)$ already takes $O(n^{1.5} \log N)$ time. Fortunately, these R_i are very similar. Each R_i is X^y with $\overline{R}_i$ removed. Exploiting this similarity and using the algorithm in section 2 for all-cavity matchings, we can perform an $O(n^{1.5} \log N)$-time preprocessing so that we can retrieve $\text{mast}(W^v, R_i)$ for any internal node v in W and $i \in [1, b]$ in $O(\log^2 n)$ time. Therefore, it takes $O(|W_i| \log^2 n)$ time to compute $\alpha(v)$ for all internal nodes v of one particular W_i, and $O(n \log^2 n)$ time for all W_i. The $O(n^{1.5} \log N)$-time preprocessing is detailed as follows.

First, note that if we remove y from X^y, the tree would decompose into the subtrees $\overline{R}_1, \ldots, \overline{R}_b$. Thus, the total size of all $\overline{R}_i$ is at most n. The next lemma suggests a way to retrieve efficiently $\text{mast}(W^v, \overline{R}_i)$ and $\max\{\text{mast}(W^v, \overline{R}_j) \mid j \in I\}$ for any $v \in W$ and and $I \subseteq [1, b]$.

LEMMA 6.2. *We can compute* $\text{mast}(W, \overline{R}_i)$ *for all* $i \in [1, b]$ *in* $O(n^{1.5} \log N)$ *time. Then, we can retrieve* $\text{mast}(W^v, \overline{R}_i)$ *for any node* v *in* W *and* $i \in [1, b]$ *in* $O(\log n)$ *time. Furthermore, we can build a data structure to retrieve* $\max\{\text{mast}(W^v, \overline{R}_j) \mid j \in I\}$ *for any* $v \in W$ *and* $I \subseteq [1, b]$ *in* $O(\log^2 n)$ *time.*

Proof. This lemma follows from the rooted subtree algorithm and related data structures in [11]. □

Below, we give a formula to compute $\text{mast}(W^v, R_i)$ efficiently. For any $z \in W$ and $i \in [1, b]$, let $\text{r-mast}(W^z, R_i)$ denote the maximum size among all the agreement subtrees of W^z and R_i in which z is mapped to the root of R_i.

LEMMA 6.3.

$$\text{mast}(W^v, R_i) = \max \begin{cases} \max\{\text{mast}(W^v, \overline{R}_j) \mid j \in [1, b], j \neq i\}; \\ \max\{\text{r-mast}(W^z, R_i) \mid z \in W^v\}. \end{cases}$$

Proof. Observe that $\text{mast}(W^v, R_i) = \text{mast}(W^z, R_i) = \text{r-mast}(W^z, R_i)$ if in some maximum agreement subtree of W^v and R_i, the root of R_i is mapped to some node z in W^v. On the other hand, $\text{mast}(W^v, R_i) = \text{mast}(W^v, \overline{R}_j)$ for some $j \neq i$ if in some maximum agreement subtree of W^v and R_i, the root of R_i is not mapped to any node z in W^v. □

By Lemma 6.3, we decompose the computation of $\text{mast}(W^v, R_i)$ into two parts. The value $\max\{\text{mast}(W^v, \overline{R}_j) \mid j \in [1, b], j \neq i\}$ is determined by answering two queries $\max\{\text{mast}(W^v, \overline{R}_j) \mid j \in [1, i-1]\}$ and $\max\{\text{mast}(W^v, \overline{R}_j) \mid j \in [i+1, b]\}$ in $O(\log^2 n)$ time by Lemma 6.2. The computation of $\max\{\text{r-mast}(W^z, R_i) \mid z \in W^v\}$ makes use of a maximum weight matching of some bipartite graph as follows.

Let $\text{Ch}(z)$ denote the set of children of a node z in a tree. Let $G_{z,i} \subseteq \text{Ch}(z) \times \{\overline{R}_1, \ldots, \overline{R}_{i-1}, \overline{R}_{i+1}, \ldots, \overline{R}_b\}$ be a bipartite graph where $w \in \text{Ch}(z)$ is connected to $\overline{R}_j$ if and only if $\text{mast}(W^w, \overline{R}_j) > 0$. Such an edge has weight $\text{mast}(W^w, \overline{R}_j) \leq N$.

FACT 6.4 (see [11]). *If the root of R_i is mapped to z in some maximum agreement subtree of W^z and R_i, then a maximum weight matching of $G_{z,i}$ consists of at least two edges, and* $\text{mwm}(G_{z,i}) = \text{r-mast}(W^z, R_i)$.

Note that if a maximum weight matching of $G_{z,i}$ consists of one edge, it corresponds to an agreement subtree of W^z and R_i in which the root of R_i is not mapped to any node in W^z. Thus, it is possible that $\text{mwm}(G_{z,i}) > \text{r-mast}(W^z, R_i)$. Nevertheless, in this case we are no longer interested in the exact value of $\text{r-mast}(W^z, R_i)$ since in a maximum agreement subtree of W^z and R_i, the root of R_i is not mapped to any node in W^z. In fact, Lemma 6.3 can be rewritten with the $\text{r-mast}(W^z, R_i)$ replaced by $\text{mwm}(G_{z,i})$. Furthermore, since $G_{z,1}, G_{z,2}, \ldots, G_{z,b}$ are very similar, the weights of a maximum weight matching cannot be all distinct.

LEMMA 6.5. *At least $b - d_z$ of* $\text{mwm}(G_{z,1})$, $\text{mwm}(G_{z,2}), \ldots, \text{mwm}(G_{z,b})$ *have the same value, where d_z denotes the degree of z in W.*

Proof. Consider the bipartite graph $K \subseteq \text{Ch}(z) \times \{\overline{R}_1, \ldots, \overline{R}_b\}$ in which a node $w \in \text{Ch}(z)$ is connected to $\overline{R}_i$ if and only if $\text{mast}(W^w, \overline{R}_i) > 0$. This edge is given a weight of $\text{mast}(W^w, \overline{R}_i)$. Then, every $G_{z,i}$ is a subgraph of K. Let M be a maximum weight matching of K. Observe that if an R_i is not adjacent to any edge in M, then M is also a maximum weight matching of $G_{z,i}$. Since M contains at most d_z edges, there are at least $b - d_z$ trees R_i not adjacent to any edge in M and the corresponding $\text{mwm}(G_{z,i})$ have the same value. □

We next use $O(n^{1.5} \log N)$ time to find for all z in W, $\text{mwm}(G_{z,1}), \ldots, \text{mwm}(G_{z,b})$. The results are to be stored in an array A_z of dimension b for each node z, i.e., $A_z[i] = \text{mwm}(G_{z,i})$. Note that if we represent each A_z as an ordinary array, then filling these arrays entry by entry for all $z \in W$ would cost $\Omega(bn)$ time. Nevertheless, by Lemma 6.5, most of the weights $\text{mwm}(G_{z,i})$ have the same value. Thus, we store these values in sparse arrays. Like an ordinary array, any entry in a *sparse array* A can be read and modified in $O(1)$ time. In addition, we require that all the entries

in A can be initialized to a fixed value in $O(1)$ time and that all the distinct values stored in A can be retrieved in $O(m)$ time, where m denotes the number of distinct values in A. For an implementation of sparse array, see Exercise 2.12, page 71 of [2].

Before showing how to build these sparse arrays, we illustrate how they support the computation of

$$\max\{\mathrm{mwm}(G_{z,i}) \mid z \in W^v\} = \max\{A_z[i] \mid z \in W^v\}. \tag{6.2}$$

An efficient data structure for answering such a query is given in Appendix B. Let m_z be the number of distinct values in A_z, and $m = \sum_{z \in W}(m_z + 1)$. Let $\alpha(n)$ denote the inverse Ackermann function. Appendix B shows how to construct a data structure on top of the sparse arrays A_z in $O(m\alpha(|W|))$ time such that we can retrieve for any $v \in W$ and $i \in [1, b]$ the value of $\max\{A_z[i] \mid z \in W^v\}$ in $O(\log |W|)$ time. From Lemma 6.5, $m_z \leq d_z + 1$ for all $z \in W$; thus, $m = O(|W|)$. Therefore, the data structure can be built in time $O(m\alpha(|W|)) = O(|W|\alpha(|W|)) = O(n \log n)$ and the retrieval time of (6.2) is $O(\log |W|) = O(\log n)$.

To summarize, after building all the necessary data structures, we can retrieve $\max\{\mathrm{mast}(W^v, \overline{R}_j) \mid j \in [1, b], j \neq i\}$ in $O(\log^2 n)$ time and $\max\{\text{r-mast}(W^z, R_i) \mid z \in W^v\}$ in $O(\log n)$ time. Hence, for any $v \in W$ and $i \in [1, b]$, $\mathrm{mast}(W^v, R_i)$ can be computed in $O(\log^2 n)$ time.

To complete our discussion, we show below how to construct a sparse array A_z or equivalently compute the weights $\{\mathrm{mwm}(G_{z,i}) \mid i \in [1, b]\}$ efficiently. We cannot afford to examine every $G_{z,i}$ and compute $\mathrm{mwm}(G_{z,i})$ separately. Instead we build only one weighted graph $G_z \subseteq \mathrm{Ch}(z) \times \{\overline{R}_1, \ldots, \overline{R}_b\}$ as follows.

For a node z in W, the *max-child* z' of z is a child of z such that the subtree rooted at z' contains the maximum number of atomic leaves among all the subtrees attached to z. Let $\kappa(z)$ denote the total number of atomic leaves that are in W^z but not in $W^{z'}$. The edges of G_z are specified as follows.

- For any non-max-child u of z, G_z contains an edge between u and some $\overline{R}_i$ if and only if $\mathrm{mast}(W^u, \overline{R}_i) > 0$. There are at most $\kappa(z)$ such edges.
- Regarding the max-child z' of z, we put into G_z only a limited number of edges between z' and $\{\overline{R}_1, \ldots, \overline{R}_b\}$. For each $\overline{R}_i$ already connected to some non-max-child of z, G_z has an edge between z' and $\overline{R}_i$ if $\mathrm{mast}(W^{z'}, \overline{R}_i) > 0$. Among all other $\overline{R}_i$, we pick $\overline{R}_{i'}$ and $\overline{R}_{i''}$ such that $\mathrm{mast}(W^{z'}, \overline{R}_{i'})$ and $\mathrm{mast}(W^{z'}, \overline{R}_{i''})$ are the first and second largest.
- Every edge $(u, \overline{R}_i)$ in G_z is given a weight of $\mathrm{mast}(W^u, \overline{R}_i)$.

LEMMA 6.6. *For all* $i \in [1, b]$, $\mathrm{mwm}(G_z - \{\overline{R}_i\}) = \mathrm{mwm}(G_{z,i})$. *Furthermore,* G_z *can be built in* $O((\kappa(z) + 1) \log^2 n)$ *time.*

Proof. The fact that $\mathrm{mwm}(G_z - \{\overline{R}_i\}) = \mathrm{mwm}(G_{z,i})$ follows from the construction of G_z. Note that G_z contains $O(\kappa(z) + 1)$ edges. All edges in G_z, except $(z', \overline{R}_{i'})$ and $(z', \overline{R}_{i''})$, can be found using $O(\kappa(z))$ time. The weight of these edges can be found in $O(\kappa(z) \log n)$ time using Lemma 6.2. To identify $(z', \overline{R}_{i'})$ and $(z', \overline{R}_{i''})$, note that at most $\kappa(z)$ instances of $\overline{R}_i$ are connected to some non-max-child of z. All other $\overline{R}_i$ are partitioned into at most $\kappa(z) + 1$ intervals. For each interval, say $I \subseteq [1, b]$, by Lemma 6.2, the corresponding $\mathrm{mast}(W^{z'}, \overline{R}_i)$ which attains the maximum in the set $\{\mathrm{mast}(W^{z'}, \overline{R}_j) \mid j \in I\}$ can be found in $O(\log^2 n)$ time. Thus, by scanning all the $\kappa(z) + 1$ intervals, $\overline{R}_{i'}$ can be found in $O((\kappa(z) + 1) \log^2 n)$ time. $\overline{R}_{i''}$ can be found similarly. ∎

Since G_z contains $O(\kappa(z) + 1)$ edges, and each edge has weight at most N, we use the Gabow–Tarjan algorithm [13] to compute $\mathrm{mwm}(G_z)$ in $O(\sqrt{\kappa(z) + 1}(\kappa(z) +$

$1)\log N)$ time. Then, using our algorithm for all-cavity maximum weight matching, we can compute $\mathrm{mwm}(G_z - \{\overline{R}_i\})$ for all $i \in [1, b]$, and store the results in a sparse array A_z in the same amount of time.

Thus, all G_z with $z \in W$ can be constructed in time $\sum_{z\in W} O((\kappa(z)+1)\log^2 n)$, which is $O(n^{1.5}\log N)$ as $\sum_{z\in W}\kappa(z) = O(n\log n)$ [9]. Given all G_z, the time for computing A_z for all $z \in W$ is $O(\sum_{z\in W}(\kappa(z)+1)^{1.5}\log N)$.

LEMMA 6.7. $\sum_{z\in W}(\kappa(z)+1)^{1.5}\log N = O(n^{1.5}\log N)$.

Proof. Let $T(W) = \sum_{z\in W}(\kappa(z)+1)^{1.5}\log N$. Let P be a path starting from the root of W such that every next node is the max-child of its predecessor. Then $\sum_{z\in P}\kappa(z) \le |W| \le n$. Let $\chi(P)$ denote the set of subtrees attached to some node on P. The subtrees in $\chi(P)$ are label-disjoint and each has size at most $n/2$. Thus,

$$\begin{aligned} T(W) &\le \sum_{z\in P}(\kappa(z)+1)^{1.5}\log N + \sum_{W'\in\chi(P)} T(W') \\ &\le n^{1.5}\log N + \sum_{W'\in\chi(P)} T(W') \\ &= O(n^{1.5}\log N). \qquad \square \end{aligned}$$

7. Auxiliary information for X with one or two shrunk leaves.

7.1. X has one shrunk leaf. Consider the computation of $\mathrm{mast}(W, X)$. According to the algorithm, $\mathrm{mast}(W, X)$ will spawn b subproblems $\mathrm{mast}(W_1, X_1), \ldots, \mathrm{mast}(W_b, X_b)$, which are defined by an internal node y in X adjacent to the nodes $v_1, \ldots, v_b$. Also, for every $i \in [1, b]$, R_i and $\overline{R}_i$ denote the subtrees $X^{v_i y}$ and $X^{y v_i}$, respectively. Suppose that X has one shrunk leaf and without loss of generality, assume that the shrunk leaf of X is in $\overline{R}_b$, i.e., X_b has two shrunk leaves and all the other X_i have one shrunk leaf each. This section shows how to find the auxiliary information required by $W_1, \ldots, W_b$ in $O(n^{1.5}\log N)$ time.

LEMMA 7.1. *The auxiliary information required by $W_1, \ldots, W_{b-1}$ can be computed in $O(n^{1.5}\log N)$ time.*

Proof. Note that $\mathrm{mast}(W_1, X_1), \ldots, \mathrm{mast}(W_{b-1}, X_{b-1})$ are almost identical to the subproblems considered in section 6 in that all the X_i have exactly one shrunk leaf each. Using exactly the same approach, we can compute the auxiliary information in $W_1, \ldots, W_{b-1}$. $\square$

The remaining section focuses on computing the auxiliary information in W_b. Let γ_1 and γ_2 be the two shrunk leaves of X_b. Assume that γ_1 is also a shrunk leaf in X, and γ_2 represents R_b. Let Q^+ be the subtree obtained by connecting γ_1 and R_b together with a node. To compute the auxiliary information in W_b, we require the values $\mathrm{mast}(W^v, \gamma_1)$, $\mathrm{mast}(W^v, R_b)$, and $\mathrm{mast}(W^v, Q^+)$ for all nodes $v \in W$. These values are computed based on the following lemma.

LEMMA 7.2. $\mathrm{mast}(W^v, \gamma_1)$, $\mathrm{mast}(W^v, R_b)$, *and* $\mathrm{mast}(W^v, Q^+)$ *for all nodes $v \in W$ can be computed in $O(n^{1.5}\log N)$ time.*

Proof. By Lemma 4.2, $\mathrm{mast}(W, R_b)$ and $\mathrm{mast}(W, Q^+)$ can be computed in time $O(n^{1.5}\log N)$ and afterwards, for each node $v \in W$, $\mathrm{mast}(W^v, R_b)$ and $\mathrm{mast}(W^v, Q^+)$ can be retrieved in $O(1)$ time. For each node $v \in W$, $\mathrm{mast}(W^v, \gamma_1)$ is the auxiliary information stored at v in W and can be retrieved in $O(1)$ time. $\square$

Now, we are ready to compute the auxiliary information stored at each node $v \in W_b$. No auxiliary information is required for atomic leaves. Below, Lemmas 7.3 and 7.4 show that using $O(n)$ additional time, we can compute the auxiliary information

in internal nodes and in compressed leaves, respectively. In summary, the auxiliary information in $W_1, \ldots, W_b$ can be computed in $O(n^{1.5} \log N)$ time.

LEMMA 7.3. *Given* $\text{mast}(W^v, \gamma_1)$, $\text{mast}(W^v, R_b)$, *and* $\text{mast}(W^v, Q^+)$ *for all nodes* $v \in W$, *the auxiliary information stored at the internal nodes in* W_b *can be found in* $O(n)$ *time.*

Proof. Let J_b be the set of labels of the atomic leaves of W_b. An internal node v can be either an auxiliary node, a compressed node, or a node of $W|J_b$. If $v \in W|J_b$, then $v \in W$. Thus, $\alpha_1(v) = \text{mast}(W^v, \gamma_1)$ and $\alpha_2(v) = \text{mast}(W^v, R_b)$.

If v is a compressed node, then we need to compute $\alpha_1(v), \alpha_2(v)$ and $\alpha_+(v)$. Recall that v represents some tree path $\sigma = v_1, \ldots, v_k$ of W, where v_1 is the closest to the root, i.e., $v = v_1$. Thus, $\alpha_1(v) = \text{mast}(W^{v_1}, \gamma_1)$, $\alpha_2(v) = \text{mast}(W^{v_1}, R_b)$, and $\alpha_+(v) = \text{mast}(W^{v_1}, Q^+)$.

Thus, $O(n)$ time is sufficient for finding the auxiliary information stored at every internal node of W_b. □

LEMMA 7.4. *Given* $\text{mast}(W^v, \gamma_1)$, $\text{mast}(W^v, R_b)$, *and* $\text{mast}(W^v, Q^+)$ *for all nodes* $v \in W$, *the auxiliary information stored at the compressed leaves in* W_b *can be found in* $O(n)$ *time.*

Proof. If v is a compressed leaf in W, v's parent u must not be an auxiliary node. Depending on whether u is a compressed node, we have two cases.

Case A. u is not a compressed node. We need to compute $\alpha_1(v), \alpha_2(v), \alpha_+(v), \beta(v)$. Note that u is also in W. When W_b is constructed from W, some of the subtrees of W attached to u are replaced by v and no longer exist in W_b. Let $W^{p_1}, \ldots, W^{p_k}$ be these subtrees. Observe that both v and $W^{p_1}, \ldots, W^{p_k}$ represent the same set of subtrees in T_1. Thus,

- $\alpha_1(v) = \max\{\text{mast}(W^{p_i}, \gamma_1) \mid 1 \le i \le k\}$;
- $\alpha_2(v) = \max\{\text{mast}(W^{p_i}, R_b) \mid 1 \le i \le k\}$;
- $\alpha_+(v) = \max\{\text{mast}(W^{p_i}, Q^+) \mid 1 \le i \le k\}$;
- $\beta(v) = \max\{\text{mast}(W^{p_i}, \gamma_1) + \text{mast}(W^{p_j}, R_b) \mid 1 \le i \ne j \le k\}$.

These four values can be found in $O(k)$ time. Since $W^{p_1}, \ldots, W^{p_k}$ are subtrees attached to u in W, k is at most the degree of u in W. Moreover, the sum of the degrees of all internal nodes of W is $O(n)$. Therefore, $O(n)$ time suffices to compute the auxiliary information for all the compressed leaves in W_b whose parents are not compressed node.

Case B. u is a compressed node. We need to compute $\alpha_1(v)$, $\alpha_2(v)$, $\alpha_+(v)$, $\beta(v)$, $\beta_{1\succ 2}(v)$, and $\beta_{1\succ 2}(v)$. Note that u is compressed from a tree path $p_1, \ldots, p_k$ in W, where p_1 is the closest to the root. Moreover, v is compressed from the subtrees hanging between p_1 and p_k. For every $i \in [1, k]$, let $\mathcal{T}_i$ be the set of subtrees of W attached to p_i that are compressed into v. Both v and the subtrees in $\cup_{1 \le i \le k} \mathcal{T}_i$ represent the same set of subtrees in T_1. The auxiliary information stored at v can be expressed as follows.

- $\alpha_1(v) = \max\{\text{mast}(W^q, \gamma_1) \mid W^q \in \mathcal{T}_i \text{ for some } i \in [1, k]\}$.
- $\alpha_2(v) = \max\{\text{mast}(W^q, R_b) \mid W^q \in \mathcal{T}_i \text{ for some } i \in [1, k]\}$.
- $\alpha_+(v) = \max\{\text{mast}(W^q, Q^+) \mid W^q \in \mathcal{T}_i \text{ for some } i \in [1, k]\}$.
- $\beta(v) = \max_{1 \le i \le k}[\max\{\text{mast}(W^q, \gamma_1) + \text{mast}(W^{q'}, R_b) \mid W^q, W^{q'} \in \mathcal{T}_i\}]$.
- $\beta_{1\succ 2}(v) = \max_{1 \le j < i \le k}[\max\{\text{mast}(W^q, \gamma_1) \mid W^q \in \mathcal{T}_i\} + \max\{\text{mast}(W^{q'}, R_b) \mid W^{q'} \in \mathcal{T}_j\}]$.
- $\beta_{2\succ 1}(v) = \max_{1 \le j < i \le k}[\max\{\text{mast}(W^q, R_b) \mid W^q \in \mathcal{T}_i\} + \max\{\text{mast}(W^{q'}, \gamma_1) \mid W^{q'} \in \mathcal{T}_j\}]$.

These values can be found in $O(\sum_{1 \le i \le k} d_{p_i})$ time, where d_{p_i} is the degree of p_i in W.

Thus, the auxiliary information for every compressed leaf of W_b, whose parents are compressed nodes, can be computed in $O(n)$ time. □

7.2. X has two shrunk leaves. Recall that the subproblems $\text{mast}(W_1, X_1)$, $\ldots, \text{mast}(W_b, X_b)$ are spawned from $\text{mast}(W, X)$. This section considers the case where X has two shrunk leaves. Without loss of generality, assume that the two shrunk leaves are in $\overline{R}_{b-1}$ and $\overline{R}_b$, respectively. Then, $X_1, \ldots, X_{b-2}$ each have one shrunk leaf. X_{b-1} and X_b each have two shrunk leaves. Below, we show how to compute the auxiliary informations of $W_1, \ldots, W_b$ in $O(n^{1.5} \log N)$ time.

LEMMA 7.5. *The auxiliary information required by $W_1, \ldots, W_{b-2}$ can be computed in $O(n^{1.5} \log N)$ time.*

Proof. The proof of this lemma is the same as that of Lemma 7.1. □

For the remaining subproblems $\text{mast}(W_{b-1}, X_{b-1})$ and $\text{mast}(W_b, X_b)$, both X_{b-1} and X_b have two shrunk leaves. By symmetry, it suffices to discuss the computation of $\text{mast}(W_b, X_b)$ only. Lemma 7.6 shows that the auxiliary information in W_b can be computed in $O(n^{1.5} \log N)$ time. Therefore, the auxiliary information in $W_1, \ldots, W_b$ can be computed in $O(n^{1.5} \log N)$ time.

LEMMA 7.6. *The auxiliary information in W_b can be computed in $O(n^{1.5} \log N)$ time.*

Proof. Let γ_1 and γ_2 be the two shrunk leaves of X_b. Assume that γ_1 is also a shrunk leaf in X and γ_2 represents R_b, i.e., γ_2 represents the subtree $U_2^{v_b y}$ of T_1. Let Q^+ be the subtree obtained by connecting γ_1 and R_b. By the same argument as in Lemmas 7.3 and 7.4, the auxiliary information in W_b can be computed based on the values $\text{mast}(W^v, \gamma_1)$, $\text{mast}(W^v, R_b)$, and $\text{mast}(W^v, Q^+)$ for all $v \in W$. The value $\text{mast}(W^v, \gamma_1)$ can be found in W. The values $\text{mast}(W^v, R_b)$ and $\text{mast}(W^v, Q^+)$ for all $v \in W$ can be retrieved in $O(1)$ time after $\text{mast}(W, R_b)$ and $\text{mast}(W, Q^+)$ are computed in $O(n^{1.5} \log N)$ time based on Lemma 4.4. Then the auxiliary information in W_b can be computed in $O(n)$ time. □

8. Extension. We have presented an $O(N^{1.5} \log N)$-time algorithm for computing a maximum agreement subtree of two unrooted evolutionary trees of at most N nodes each. This algorithm can be modified slightly to compute a maximum agreement subtree for two mixed trees M_1 and M_2.

For a mixed tree M, a node ℓ is *consistent* with a node u if the directed edges on the path between u and ℓ all point away from u. Let M^u be the rooted tree constructed by assigning u in M as the root and removing the nodes of M inconsistent with u. Given two mixed tree M_1 and M_2, we define a maximum agreement subtree of M_1 and M_2 to be the one with the largest number of labels among the maximum agreement subtree of M_1^u and M_2^v over all nodes $u \in M_1$ and $v \in M_2$. That is,

$$\text{mast}(M_1, M_2) = \max\{\text{mast}(M_1^u, M_2^v) \mid u \in M_1, v \in M_2\}.$$

As in the unrooted case, to compute $\text{mast}(M_1, M_2)$, we find a separator y of M_1 and compute $\text{mast}(M_1^y, M_2)$. However, we need to delete the nodes of M_1 not in M_1^y. When computing $\text{mast}(M_1^y, M_2)$, we construct some rooted subtrees of M_2. Again, we delete the nodes of M_2 not in these rooted subtrees. Such deletions are straightforward and do not increase the time complexity of computing $\text{mast}(M_1, M_2)$. Thus, $\text{mast}(M_1, M_2)$ can be computed in $O(N^{1.5} \log N)$ time.

Appendix A. Computing $\text{mast}(W_1, W_2)$. Let T_1 and T_2 be rooted evolutionary trees. Let R_1 and R_2 be two label-disjoint rooted subtrees of T_2. Let $W_1 = T_1 \otimes (R_1, R_2)$ and $W_2 = T_2 \ominus (R_1, R_2)$. This section shows that $\text{mast}(W_1, W_2)$

can be computed as if W_1 and W_2 were ordinary rooted evolutionary trees [9, 11, 20] with some special procedures on handling compressed and shrunk leaves. Note that the case where W_1 and W_2 are compressed and shrunk with respect to a subtree can be treated as the special case where R_1 is empty.

LEMMA A.1. *We can compute* $\text{mast}(W_1, W_2)$ *in* $O(n^{1.5} \log N)$ *time, where* $n = \max\{|W_1|, |W_2|\}$ *and* $N = \max\{|T_1|, |T_2|\}$. *Afterwards, we can retrieve* $\text{mast}(W_1^u, W_2)$ *for any node* u *of* W_1 *in* $O(1)$ *time.*

Proof. We adopt the framework of Farach and Thorup's algorithm [11], which is essentially a sparsified dynamic programming based on the following formula. For any internal nodes u of W_1 and v of W_2,

$$\text{mast}(W_1^u, W_2^v) = \max \begin{cases} \max\{\text{mast}(W_1^x, W_2^v) \mid x \text{ is a child of } u\}; \\ \max\{\text{mast}(W_1^u, W_2^y) \mid y \text{ is a child of } v\}; \\ \text{r-mast}(W_1^u, W_2^v), \end{cases} \tag{A.1}$$

where $\text{r-mast}(W_1^u, W_2^v)$ denotes the maximum size of all the agreement subtrees of W_1^u and W_2^v in which u is mapped to v.

Our algorithm differs from Farach and Thorup's algorithm in the way how each individual $\text{mast}(W_1^u, W_2^v)$ is computed. When W_1 and W_2 are ordinary evolutionary trees, each $\text{mast}(W_1^u, W_2^v)$ is found by computing a maximum weight matching of some bipartite graph, and it takes $O(n^{1.5} \log n)$ time to compute $\text{mast}(W_1, W_2)$. Below, we show that when W_1 and W_2 have compressed and shrunk leaves, each $\text{mast}(W_1^u, W_2^v)$ can be found either in constant time or by computing at most two maximum weight bipartite matchings of similar graphs but with edge weights bounded by N instead of n. Thus, we can compute $\text{mast}(W_1, W_2)$ using the same sparsified dynamic programming in [11]; as a by-product, we can afterwards retrieve $\text{mast}(W_1^u, W_2)$ for any node u of W_1 in $O(1)$ time. The enlarged upper bound of edge weights increases the time complexity to $O(n^{1.5} \log N)$, though.

In the rest of this section, we show how each $\text{mast}(W_1^u, W_2^v)$ is computed. First, we consider the case when u is a leaf. The following case analysis shows that $O(1)$ time suffices to compute $\text{mast}(W_1^u, W_2^v)$.

Case 1. u is an atomic leaf. If W_2^v contains a leaf with the same label as that of u, then $\text{mast}(W_1^u, W_2^v) = 1$; otherwise it equals zero.

Case 2. u is an auxiliary leaf. Then, $\text{mast}(W_1^u, W_2^v) = 0$.

Case 3. u is a compressed leaf. By definition, u can only be mapped to γ_1, γ_2, or the least common ancestor y_c of γ_1 and γ_2. If W_2^v has no shrunk leaves, then $\text{mast}(W_1^u, W_2^v) = 0$. If W_2^v has only one shrunk leaf, say γ_1, then $\text{mast}(W_1^u, W_2^v) = \alpha_1(u)$. If W_2^v has two shrunk leaves, then W_2^v must also contain y_c and $\text{mast}(W_1^u, W_2^v) = \max\{\alpha_1(u), \alpha_2(u), \alpha_+(u)\}$.

Next, we consider the case when u is an internal node. Assume that v is an atomic leaf. Then $\text{mast}(W_1^u, W_2^v) = 1$ if W_1^u contains a leaf with the same label as that of v, and zero otherwise. If v is a shrunk leaf, say γ_1, then $\text{mast}(W_1^u, W_2^v) = \alpha_1(u)$. It remains to consider the case when v is an internal node. Due to the nature of dynamic programming, we only need to compute $\text{r-mast}(W_1^u, W_2^v)$, then we can apply (A.1) to compute $\text{mast}(W_1^u, W_2^v)$. We further divide our discussion into the following three cases.

Case 1. u is an auxiliary internal node. In such case, u has only two children; one of them is an auxiliary leaf. By definition, an auxiliary leaf will not be mapped to any node in any agreement subtree of W_1^u and W_2^v; thus, there is no agreement subtree in which u is mapped to v and $\text{r-mast}(W_1^u, W_2^v) = 0$.

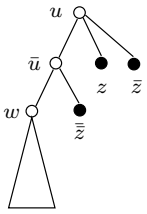


FIG. 6. *Structure of* W_1^u.

Case 2. u is an ordinary internal node. As in [11], we first construct the bipartite graph defined as follows: Let A and B be the set of children of u and v, respectively; define $G[A, B]$ to be the bipartite graph formed by the edges $(x, y) \in A \times B$ with $\text{mast}(W_1^x, W_2^y) > 0$, and (x, y) is given a weight $\text{mast}(W_1^x, W_2^y)$.

If none of u's children is an auxiliary leaf, then $\text{r-mast}(W_1^u, W_2^v) = \text{mwm}(G[A, B])$. Otherwise, let $\bar{z}$ be the child of u which is an auxiliary. In this case, u also has a compressed child z. Other than z and $\bar{z}$, no other child of u is a compressed and auxiliary leaf. On the other hand, consider the rooted subtrees W_2^x rooted at the children x of v. If the shrunk leaves appear together in one of such subtrees, then by definition, $\bar{z}$ cannot be mapped to any shrunk leaf in any agreement subtree of W_1^u and W_2^v, and $\text{r-mast}(W_1^u, W_2^v) = \text{mwm}(G[A, B] - \{\bar{z}\})$. If the shrunk leaves appear in two different subtrees rooted at two children y_1 and y_2 of v, then

$$\text{r-mast}(W_1^u, W_2^v) = \max\Big\{\text{mwm}(G[A, B]-\{\bar{z}\}), \text{mwm}(G[A, B]-\{z, \bar{z}, y_1, y_2\})+\beta(z)\Big\}.$$

Case 3. u is a compressed internal node. By definition of a compressed node, the structure of W_1^u is very restrictive—u has exactly three children z, $\bar{z}$, and an auxiliary internal node $\bar{u}$; $\bar{u}$ has two children, an auxiliary leaf $\bar{\bar{z}}$ and an uncompressed internal node w; see Figure 6. To find $\text{r-mast}(W_1^u, W_2^v)$, we note that there are only six possible ways for how $z, \bar{z}, \bar{\bar{z}}$ are mapped to γ_1 and γ_2. We consider each of these cases and $\text{r-mast}(W_1^u, W_2^v)$ is the maximum of the values found. We discuss only the case where γ_1 and γ_2 are mapped to z and $\bar{\bar{z}}$, respectively. The other cases can be handled similarly. Let P be the path between γ_2 and y_c. Let $S(P)$ denote the set of subtrees hanged on P. The size of the largest agreement subtrees of W_1^u and W_2^v in which γ_1 and γ_2 are mapped to z and $\bar{z}$, respectively, equals

$$\beta_{1\succ 2}(z) + \max\{\text{mast}(W_1^w, \tau) \mid \tau \in S(P)\}. \tag{A.2}$$

Note that using the technique in [11], we can precompute $\max\{\text{mast}(W_1^x, \tau) \mid \tau \in S(P)\}$ for all $x \in W_1$ in $O(n^{1.5} \log N)$ time. Afterwards, (A.2) can be found in constant time. □

Appendix B. Preprocessing for finding $\max\{A_z[i] | z \in W^v\}$.

Let h be the number of nodes in W. Consider the h arrays A_z of dimension b where $z \in W$. Recall that m_z is the number of distinct values in A_z, and $m = \sum_{z \in W}(m_z + 1)$. This section describes an $O(m\alpha(h))$-time preprocessing, which supports finding $\max\{A_z[i] \mid z \in W^v\}$, for any $i \in [1, b]$ and any node v of W, in $O(\log h)$ time.

By definition, each A_z has at least $b - m_z$ entries storing some common value c_z. For every $i \in [1, b]$, let Γ_i be the set of nodes z where $A_z[i]$ stores a value different from c_z. Note that $\sum m_z = \sum_{1 \le i \le b} |\Gamma_i|$. We assume that each node of W is identified

uniquely by an integer in $[1, h]$ assigned by a preorder tree traversal [8]. For any node $v \in W$, let $\beta(v)$ be the number of proper descendents of v in W.

Based on Lemma B.1, $\max\{A_z[i] \mid z \in W^v\} = \max\{A_z[i] \mid z \in [v, v + \beta(v)]\}$ for any $v \in W$, $i \in [1, b]$. Therefore, to solve our problem, it is sufficient to give an $O(m\alpha(h))$-time preprocessing to support finding $\max\{A_z[i] \mid z \in H\}$ for any $i \in [1, b]$ and any interval $H \subseteq [1, h]$ in $O(\log h)$ time.

LEMMA B.1. *For any $v \in W$ and $i \in [1, b]$, $\max\{A_z[i] \mid z \in W^v\} = \max\{A_z[i] \mid z \in [v, v + \beta(v)]\}$.*

Proof. It is straightforward. □

Our preprocessing does not work on each sequence $A_1[i], A_2[i], \ldots, A_h[i]$ directly. Instead, it first draws out useful information about the common values c_z stored in the sequences and applies a contraction technique to shorten each sequence. Then, it executes Fact B.2 on these shortened sequences.

FACT B.2 (see [3]). *Given any sequence $a_1, \ldots, a_h$ of real numbers, we can preprocess these h numbers in $O(h)$ time so that we can find the maximum of any subsequence $a_x, a_{x+1}, \ldots, a_y$ in $O(\alpha(h))$ time.*

Our preprocessing is detailed as follows. Its time complexity is $O(m\alpha(h))$ as shown in Lemma B.3.

1. For each $i \in [1, b]$, find Γ_i and arrange the integers in Γ_i in ascending order.
2. Apply Fact B.2 to the sequence $c_1, \ldots, c_h$.
3. For every nonempty $\Gamma_i = \{x_1 < \cdots < x_d\}$, compute $\beta_\ell = \max\{A_x[i] \mid x \in (x_\ell, x_{\ell+1})\}$ for every $\ell \in [1, d-1]$, and then apply Fact B.2 to the sequence $A_{x_1}[i], \beta_1, A_{x_2}[i], \ldots, \beta_{d-1}, A_{x_d}[i]$.

LEMMA B.3. *The preprocessing requires $O(m\alpha(h))$ time.*

Proof. We can examine all the entries of A_z whose values differ from c_z in $O(m_z)$ time. By examining all such entries of $A_1, \ldots, A_h$, we can construct Γ_i and arrange the integers in Γ_i in ascending order. Thus, step 1 takes $O(m)$ time. Step 2 takes $O(h) = O(m)$ time. For step 3, we first analyze the time required to process one nonempty $\Gamma_i = \{x_1 < \cdots < x_d\}$. Note that $(x_\ell, x_{\ell+1}) \cap \Gamma_i = \phi$ for every $\ell \in [1, d-1]$. Thus, $\beta_\ell = \max\{c_x \mid x \in (x_\ell, x_{\ell+1})\}$ can be computed in $O(\alpha(n))$ time using the result of step 2. Summing over all $\ell \in [1, d-1]$, computing all β_ℓ takes $O(|\Gamma_i|\alpha(h))$ time. Applying Fact B.2 to the sequence $A_{x_1}[i], \beta_1, A_{x_2}[i], \ldots, \beta_{d-1}, A_{x_d}[i]$ takes $O(|\Gamma_i|)$ time. In total, it takes $O(|\Gamma_i|\alpha(h))$ time to process one Γ_i, and step 3 takes $O(\sum |\Gamma_i|\alpha(h)) = O(m\alpha(h))$ time. Thus, the total time of our preprocessing is $O(m\alpha(h))$. □

After the preprocessing, each query can be answered in $O(\log h)$ time as stated in the following lemma.

LEMMA B.4. *After the preprocessing, $\max\{A_z[i] \mid z \in H\}$ can be found in $O(\log n)$ time for any $i \in [1, b]$ and any interval $H \subseteq [1, h]$.*

Proof. Let $H = [p, q]$. A crucial step is to find $[p, q] \cap \Gamma_i$. Without loss of generality, assume $\Gamma_i \neq \phi$. To find $[p, q] \cap \Gamma_i$, we first find the smallest integer x_s in Γ_i that is greater than p, and the largest integer x_t in Γ_i that is smaller than q. Since Γ_i is sorted, we can find x_s and x_t in $O(\log |\Gamma_i|) = O(\log h)$ time. If $x_s > x_t$, then $[p, q] \cap \Gamma_i = \phi$; otherwise, $[p, q] \cap \Gamma_i$ is the set of integers between x_s and x_t in Γ_j.

If $[p, q] \cap \Gamma_i = \phi$, then $\max\{A_x[i] \mid x \in [p, q]\} = \max\{c_x \mid x \in [p, q]\}$. Because of step 1 of our preprocessing, we can find $\max\{c_x \mid x \in [p, q]\}$ in $O(\alpha(h))$ time.

If $[p, q] \cap \Gamma_i = \{x_s < x_{s+1} < \cdots < x_t\}$, then $[p, q] = [p, x_s - 1] \cup \{x_s\} \cup (x_s, x_{s+1}) \cup \cdots \cup \{x_t\} \cup [x_t + 1, q]$ and $\max\{A_z[i] \mid z \in [p, q]\}$ equals the maximum of

1. $\max\{A_x[i] \mid x \in [p, x_s - 1]\}$,

2. $\max\{A_x[i] \mid x \in \{x_s\} \cup (x_s, x_{s+1}) \cup \cdots \cup (x_{t-1}, x_t) \cup \{x_t\}\}$,
3. $\max\{A_x[i] \mid x \in [x_t + 1, q]\}$.

Note that item 2 equals the maximum of $A_{x_s}[i], \beta_s, \ldots, \beta_{t-1}, A_{x_t}[i]$, which can be computed in $O(\alpha(h))$ time after step 3 of our preprocessing. Since $\Gamma_i \cap [p, x_s - 1] = \phi$ and $\Gamma_i \cap [x_t + 1, q] = \phi$, step 2 enables us to compute items 1 and 3 in $O(\alpha(h))$ time. As a result, $\max\{A_z[i] \mid z \in [p, q]\}$ can be answered in $O(\log h)$ time. □

Acknowledgments. The authors thank the referees for helpful comments.

REFERENCES

[1] R. Agarwala and D. Fernández-Baca, *A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed*, SIAM J. Comput., 23 (1994), pp. 1216–1224.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[3] N. Alon and B. Schieber, *Optimal Preprocessing for Answering On-Line Product Queries*, Tech. Rep. 71, The Moise and Frida Eskenasy Institute of Computer Science, Tel Aviv University, Tel Aviv, Israel, 1987.

[4] A. Amir and D. Keselman, *Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms*, SIAM J. Comput., 26 (1997), pp. 1656–1669.

[5] J. L. Bentley, D. Haken, and J. B. Saxe, *A general method for solving divide-and-conquer recurrences*, SIGACT News, 12 (1980), pp. 36–44.

[6] H. L. Bodlaender, M. R. Fellows, and T. J. Warnow, *Two strikes against perfect phylogeny*, in Proceedings of the 19th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 623, Springer-Verlag, New York, 1992, pp. 273–283.

[7] R. Cole and R. Hariharan, *An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees*, in Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 323–332.

[8] T. H. Cormen, C. L. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[9] M. Farach, T. Przytycka, and M. Thorup, *Computing the agreement of trees with bounded degrees*, in Proceedings of the 3rd Annual European Symposium on Algorithms, Lecture Notes in Comput. Sci. 979, P. Spirakis, ed., Springer-Verlag, New York, 1995, pp. 381–393.

[10] M. Farach and M. Thorup, *Fast comparison of evolutionary trees*, Inform. and Computation, 123 (1995), pp. 29–37.

[11] M. Farach and M. Thorup, *Sparse dynamic programming for evolutionary-tree comparison*, SIAM J. Comput., 26 (1997), pp. 210–230.

[12] C. R. Finden and A. D. Gordon, *Obtaining common pruned trees*, J. Classification, 2 (1985), pp. 255–276.

[13] H. N. Gabow and R. E. Tarjan, *Faster scaling algorithms for network problems*, SIAM J. Comput., 18 (1989), pp. 1013–1036.

[14] A. V. Goldberg, *Scaling algorithms for the shortest paths problem*, SIAM J. Comput., 24 (1995), pp. 494–504.

[15] D. Gusfield, *Optimal mixed graph augmentation*, SIAM J. Comput., 16 (1987), pp. 599–612.

[16] D. Gusfield, *Efficient algorithms for inferring evolutionary trees*, Networks, 21 (1991), pp. 19–28.

[17] S. K. Kannan, E. L. Lawler, and T. J. Warnow, *Determining the evolutionary tree using experiments*, J. Algorithms, 21 (1996), pp. 26–50.

[18] M. Y. Kao, *Data security equals graph connectivity*, SIAM J. Discrete Math., 9 (1996), pp. 87–100.

[19] M. Y. Kao, *Multiple-size divide-and-conquer recurrences*, in Proceedings of the International Conference on Algorithms, the 1996 International Computer Symposium, National Sun Yat-Sen University, Kaohsiung, Taiwan, Republic of China, 1996, pp. 159–161. Reprinted in SIGACT News, 28 (1997), pp. 67–69.

[20] M. Y. Kao, *Tree contractions and evolutionary trees*, SIAM J. Comput., 27 (1998), pp. 1592–1616.

[21] M. Y. Kao, T. W. Lam, W. K. Sung, and H. F. Ting, *All-cavity maximum matchings*, in Proceedings of the 8th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 1350, H. Imai and H. W. Leong, eds., Springer-Verlag, New York, 1997, pp. 364–373.

[22] T. W. Lam, W. K. Sung, and H. F. Ting, *Computing the unrooted maximum agreement subtree in sub-quadratic time*, Nordic J. Comput. 3 (1996), pp. 295–322.

[23] T. M. Przytycka, *Transforming rooted agreement into unrooted agreement*, J. Comput. Bio., 5 (1998), pp. 333–348.

[24] M. Steel and T. Warnow, *Kaikoura tree theorems: Computing the maximum agreement subtree*, Informat. Process. Lett., 48 (1994), pp. 77–82.

[25] L. Wang, T. Jiang, and E. Lawler, *Approximation algorithms for tree alignment with a given phylogeny*, Algorithmica, 16 (1996), pp. 302–315.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 625–648

CONSTRUCTIVE, DETERMINISTIC IMPLEMENTATION OF SHARED MEMORY ON MESHES*

ANDREA PIETRACAPRINA†, GEPPINO PUCCI†, AND JOP F. SIBEYN‡

Abstract. This paper describes a scheme to implement a shared address space of size m on an n-node mesh, with m polynomial in n, where each mesh node hosts a processor and a memory module. At the core of the simulation is a hierarchical memory organization scheme (HMOS), which governs the distribution of the shared variables, each replicated into multiple copies, among the memory modules, through a cascade of bipartite graphs. Based on the expansion properties of such graphs, we devise a protocol that accesses any n-tuple of shared variables in worst-case time $O(n^{1/2+\eta})$, for any constant $\eta > 0$, using $O(1/\eta^{1.59})$ copies per variable, or in worst-case time $O(n^{1/2}\log n)$, using $O(\log^{1.59} n)$ copies per variable. In both cases the access time is close to the natural $O(\sqrt{n})$ lower bound imposed by the network diameter. A key feature of the scheme is that it can be made fully constructive when m is not too large, thus providing in this case the first efficient, constructive, deterministic scheme in the literature for bounded-degree processor networks. For larger memory sizes, the scheme relies solely on a nonconstructive graph of weak expansion. Finally, the scheme can be efficiently ported to other architectures, as long as they exhibit certain structural properties. In the paper we discuss the porting to multidimensional meshes and to the pruned butterfly, an area-universal network which is a variant of the fat-tree.

Key words. PRAM simulation, parallel computation, shared memory machines, networks of processors, meshes, expander graphs

AMS subject classification. 68Q10

PII. S009753979732712X

1. Introduction.

A desirable feature of a parallel computer is the provision of a *shared address space* that can be accessed concurrently by all the processors of the machine. Indeed, the manipulation of shared data provides a powerful and uniform mechanism for interprocessor communication and constitutes a valuable tool for the development of simple and portable parallel software. Unfortunately, when the number of processors exceeds a certain (modest) threshold, any efficient hardware realization of shared memory is either prohibitively expensive or out of reach of current technology. Therefore, a shared address space must be provided *virtually* on hardware platforms consisting of a set of processor/memory module pairs which are connected through a network of point-to-point communication links.

*Received by the editors September 10, 1997; accepted for publication (in revised form) June 24, 1999; published electronically June 27, 2000. This research was supported in part through the Leonardo Fibonacci Institute by the Istituto Trentino di Cultura. The results in this paper appeared in preliminary conference form in A. Pietracaprina, G. Pucci, and J. F. Sibeyn, *Constructive deterministic PRAM simulation on a mesh-connected computer*, in Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, NJ, 1994, pp. 248–256, and A. Pietracaprina and G. Pucci, *Improved deterministic PRAM simulation on the mesh*, in Proceedings of the 22nd International Colloquium on Automata, Languages and Programming, Szeged, Hungary, 1995, pp. 372–383.

http://www.siam.org/journals/sicomp/30-2/32712.html

†Dipartimento di Elettronica e Informatica, Università di Padova, Via Gradenigo 6/a, 35131 Padova, Italy (andrea@artemide.dei.unipd.it, geppo@artemide.dei.unipd.it). The research of these authors was supported in part by CNR and MURST of Italy.

‡Max-Planck Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany (jopsi@mpi-sb.mpg.de). The research of this author was supported in part the EC Cooperative Action IC-1000 (project ALTEC: Algorithms for Future Technologies).

This problem has received considerable attention over the past two decades and has been the target of a large number of investigations, both theoretical and applied. In the theoretical community, the problem is best known as the *PRAM simulation* problem. An (n, m)-PRAM is an abstraction of a shared-memory machine consisting of n synchronous RAM processors that have direct access to m shared *variables*. In a PRAM step, executed in unit time, any set of n variables can be read or written in parallel by the processors. A solution to the PRAM simulation problem is a scheme to perform *any* computation of an (n, m)-PRAM on a target machine consisting of a network of n processor/memory pairs. A typical PRAM simulation scheme distributes the PRAM shared variables among the n modules local to the machine processors and recasts a parallel access to the shared memory into the routing of messages from the processors requesting the variables to the processors storing such variables.

Several randomized PRAM simulation schemes have been proposed in the literature. In all these schemes, the shared variables are distributed among the memory modules via one (or more) hash functions randomly drawn from a suitable universal class. Among the most relevant results, we recall that a PRAM step can be simulated, with high probability, in $O(\log\log\log n \log^* n)$ time on the complete network [CMS95], in $O(\log n)$ time on the butterfly [Ran91], and in $O(\sqrt{n})$ time on the mesh [LMRR94].

In contrast, the development of efficient deterministic schemes, that is, schemes that guarantee a fast worst-case simulation time for any PRAM step, appears to be much harder. A simple argument shows that in order to avoid trivial worst-case scenarios, where all the variables requested in the PRAM step are stored in a small region of the network, one has to use several copies for each variable, so that only a subset of "convenient" copies needs to be reached by each operation. The number of copies used for each variable is called the *redundancy* of the scheme.

The idea of replicating each variable into multiple copies dates back to the pioneering work of Mehlhorn and Vishkin [MV84]. In their approach, a read operation need access only one (the most convenient) copy. For $m = O(n^R)$, the authors obtain a scheme for the complete interconnection which uses R copies per variable and allows any set of n reads to be satisfied in time $O(n^{1-1/R})$. However, the execution of n write operations, where all copies of the variables must be accessed, is penalized and requires $O(Rn)$ time in the worst case.

Later, Upfal and Wigderson [UW87] proposed a more balanced protocol requiring that, in order to read or write a variable, only a majority of its copies be accessed. They also represent the allocation of the copies to the modules by means of a *memory organization scheme* (MOS). An MOS is a bipartite graph $G = (V, U)$, where V is the set of shared variables, U is the set of memory modules of the underlying machine, and R edges connect each variable to the modules storing its copies. For m polynomial in n and $R = \Theta(\log n)$, the authors show that there exist suitable expanding graphs that guarantee a worst-case $O(\log n\,(\log\log n)^2)$ time to access any n variables on the complete interconnection. This bound was later improved to $O(\log n)$ in [AHMP87]. Several authors pursued the ideas in [UW87] to develop simulation schemes for *bounded-degree networks* of various topologies. In particular, schemes have been devised to simulate an arbitrary step of an (n, m)-PRAM, with m polynomial in n, in time $O(\log^2 n/\log\log n)$ on a mesh of trees (MoT) with n processors and $\Theta(n^2)$ switching elements [LPP90], or in time $O(\log n \log m/\log\log n)$ on an n-processor expander-based network [HB94], or in time $O(\log n \log\log n \log\log(m/n))$ on a suitably augmented MoT [Her96].

All of the aforementioned deterministic schemes (except for the one in [MV84] which, however, is not general since write accesses are heavily penalized) suffer from two major limitations.

1. The MOS graphs must exhibit maximum expansion relatively to the m/n ratio. Although the existence of such graphs can be proved through standard counting arguments, no efficient constructions are yet available. In addition, it is unlikely that the (few) constructions known for expanders may be of use when m is much larger than n.

2. The expansion properties of the MOS are exclusively used to curb memory contention. Network congestion issues are either ignored, as in the case of simulations on the complete network, or solved by means of separate mechanisms tailored to the specific network's topology.[1]

Recently, constructive deterministic schemes exhibiting nontrivial performance have been developed for the complete interconnection. In [PP97] three schemes are presented for $m = O(n^{3/2})$, $m = O(n^2)$, and $m = O(n^3)$ variables, which attain $O(n^{1/3})$, $O(n^{1/2})$, and $O(n^{2/3})$ access time, respectively, for any n-tuple of variables using constant redundancy. These schemes rely on MOS graphs that admit efficient explicit constructions but exhibit weak expansion. In this paper we will exploit the same constructions in a more complex framework to achieve efficient implementations of shared data on realistic, low-bandwidth machines. Specifically, we will develop a novel approach where the inefficiencies caused by the weak expansion of the memory map are absorbed into the inherent bandwidth limitations of the interconnection, and where both memory contention and network congestion are controlled through a single mechanism.

1.1. Overview of results. This paper presents a deterministic scheme for implementing a shared address space of size m on an n-node square mesh, with m polynomial in n, where each node consists of a processor with direct access to a local memory module. The scheme provides a protocol to access an arbitrary set of n shared variables in nearly optimal time for all values of m. The scheme is fully constructive for $m = O(n^{3/2})$, whereas for larger values of m it embodies only a nonconstructive component graph of constant degree whose expansion properties, however, are much weaker than those required of the graphs used in previous works. Full constructiveness can also be attained for memory sizes up to $m = O(n^{9/2})$, at the expense of a progressive degradation in performance when m gets closer to the upper bound.

The scheme adopts a novel redundant representation of the shared variables and is centered around the *hierarchical memory organization scheme* (HMOS), which provides a structured distribution of the copies of the variables among the memory modules. The HMOS consists of $k+1$ levels of logical modules built upon the set of shared variables. The modules of the first level (level 0) store copies of variables, whereas modules of level $i > 0$ store replicas of modules of level $i - 1$. The HMOS is represented by a cascade of bipartite graphs, where the first graph governs the distribution of the copies of the variables to the modules of level 0, and the other graphs govern the distribution of the replicas of modules at higher levels. Each level of the HMOS corresponds to a tessellation of the mesh into submeshes of appropriate size, with each module of that level assigned to a distinct submesh.

[1] In fact, in [HB94], an MOS with slightly less than maximum expansion is employed in order to reduce the redundancy and, consequently, network congestion, at the expense of an increase in memory contention. However, such an MOS does not embody any specific mechanism to explicitly control network congestion.

TABLE 1.1
Results in this paper.

$m = n^\tau$	Access time	Constructive	Redundancy
$\tau > \frac{3}{2}$	$O(n^{\frac{1}{2}+\eta})$ $\forall$ constant $\eta > 0$	no	$O(1/\eta^{1.59})$
$\tau > \frac{3}{2}$	$O(\sqrt{n}\log n)$	no	$O(\log^{1.59} n)$
$1 \le \tau \le \frac{3}{2}$	$O(n^{\frac{1}{2}+\eta})$ $\forall$ constant $\eta > 0$	yes	$O(1/\eta^{1.59})$
$1 \le \tau \le \frac{3}{2}$	$O(\sqrt{n}\log n)$	yes	$O(\log^{1.59} n)$
$\frac{3}{2} < \tau \le \frac{13}{6}$	$O(n^{\frac{2\tau+1}{8}})$	yes	$\Theta(1)$
$\frac{13}{6} \le \tau \le \frac{5}{2}$	$O(n^{\frac{2}{3}})$	yes	$\Theta(1)$
$\frac{5}{2} \le \tau \le \frac{9}{2}$	$O(n^{\frac{2\tau+3}{12}})$	yes	$\Theta(1)$

We devise an access protocol to satisfy n arbitrary read/write requests issued by the n processors, which takes advantage of the hierarchical structure of the HMOS. As customary in any multicopy approach, an access to a variable is executed on a selected subset of its copies. A suitable copy selection mechanism is developed to limit the number of copies to be accessed in each submesh, and, ultimately, in each individual module. In this sense, the HMOS provides a single mechanism to cope with both memory contention and network congestion, which represents a novelty with respect to previous works, where the two issues were dealt with separately.

In order to guarantee low memory contention and network congestion, the HMOS component graphs must exhibit certain expansion properties. Compared to those employed in previous schemes, our graphs have much weaker expansion, attainable using only constant (rather than logarithmic) input degree. This makes the HMOS more amenable to explicit construction. Indeed, all HMOS graphs but the first one are taken as subgraphs of a well-known combinatorial structure, the BIBD, for which an explicit and simple construction is known. As for the first graph, an explicit construction can be provided when m is not too large, thus making the HMOS fully constructive, while for large values of m, the graph can be shown to exist through a standard counting argument. Our results are reported in detail below and summarized in Table 1.1.

THEOREM 1.1. *For any constant $\tau \ge 1$, there exists a scheme to distribute $m = n^\tau$ shared variables among the local memory modules of an n-node mesh with redundancy R so that any n variables can be read/written in time*

$$T = O(n^{\frac{1}{2}+\eta})$$

for any constant $\eta > 0$, with $R = O(1/\eta^{1.59})$, or in time

$$T = O(n^{\frac{1}{2}}\log n)$$

with $R = O(\log^{1.59} n)$.

As mentioned before, for arbitrary values of m, the HMOS embodies one nonconstructive graph. Full constructiveness can be achieved when $m = O(n^{9/2})$, as reported below.

THEOREM 1.2. *For any constant τ, with $1 \le \tau < 9/2$, there exists a fully constructive scheme to distribute $m = n^\tau$ shared variables among the local memory*

modules of an n-node mesh, which, for $\tau \leq 3/2$*, achieves the same performances as those stated in Theorem* 1.1 *while, for* $\tau > 3/2$*, achieves the following access times:*

$$\begin{aligned} T &= O(n^{\frac{2\tau+1}{8}}) \quad \text{for } \tfrac{3}{2} < \tau \leq \tfrac{13}{6}, \\ T &= O(n^{\frac{2}{3}}) \quad \text{for } \tfrac{13}{6} \leq \tau \leq \tfrac{5}{2}, \\ T &= O(n^{\frac{2\tau+3}{12}}) \quad \text{for } \tfrac{5}{2} \leq \tau \leq 9/2, \end{aligned}$$

with redundancy $R = \Theta(1)$.

Compared to the natural $\Omega(\sqrt{n})$ lower bound imposed by the network diameter, our fastest access time is only a logarithmic factor away from optimal.

Prior to the present work, no efficient deterministic schemes for implementing shared memory explicitly designed for the mesh topology were known in the literature. However, the schemes designed for the complete interconnection can be implemented on the mesh through sorting and routing. In particular, $O(\sqrt{n}\log n)$ access time can be obtained by implementing the nonconstructive scheme in [AHMP87], and $O(n^{1/2+t})$ access time can be obtained by porting the constructive schemes of [PP97] to the mesh, with $t = 1/6$ for $m = O(n^{3/2})$, $t = 1/4$ for $m = O(n^2)$, and $t = 1/3$ for $m = O(n^3)$.

Our results improve upon previously published ones in the following ways. First, we attain $O(\sqrt{n}\log n)$ access time, as in [AHMP87], with a memory organization which is fully constructive when $m = O(n^{3/2})$, while, for all other values of m polynomial in n, it embodies a nonconstructive graph exhibiting much weaker expansion than that required in [AHMP87]. The recent results of [PP97] suggest that our memory map is more amenable to explicit construction. Note also that our constructive results outperform the ones obtainable by the straightforward porting to the mesh of the schemes in [PP97] discussed above.

Finally, it is important to observe that our scheme is not specifically tailored to the mesh topology but can be ported, with minor adjustments, to other topologies. In particular, the same access times reported in Theorems 1.1 and 1.2 can be attained on an n-leaf pruned butterfly, an area-universal network which is a variant of the fat-tree, and Theorem 1.1 can be generalized to hold for d-dimensional meshes, with constant d, by substituting $n^{1/d}$ for $n^{1/2}$ in the formulas.

The rest of this paper is organized as follows. Section 2 defines the machine model and introduces the routing and sorting primitives used by the access protocol. Section 3 describes the HMOS (section 3.1) and its implementation on the mesh (section 3.2). A suitable construction for the BIBDs used in the HMOS is given in an appendix to the paper. Section 4 presents the protocol for accessing an arbitrary n-tuple of shared variables. This section is subdivided into two subsections that describe the selection of the copies and the routing protocol, respectively. In section 5, suitable values for the design parameters of the HMOS are selected, and Theorems 1.1 and 1.2 are proved. Section 6 shows how the scheme can be generalized to other architectures, such as the pruned butterfly and multidimensional meshes. Section 7 closes with some final remarks.

2. Machine model. We present our shared memory implementation on a *mesh*, consisting of an array of $\sqrt{n} \times \sqrt{n}$ processor-memory pairs, connected through a two-dimensional grid of communication links. The machine operates in lock-step, where in each step a processor can perform a constant amount of local computation (including accesses to its local memory) and can exchange a constant number of words with one of its direct neighbors. Our objective is to devise a distributed representation of

$m \geq n$ shared variables on the mesh so that any n-tuple of read/write accesses to these variables can be served efficiently. The approach will be generalized to other architectures in section 6.

The access protocol will make use of the following primitives, for which optimal algorithms are known in the literature. We call *ℓ-sorting* a sorting instance in which at most ℓ keys are initially assigned to each processor and are to be redistributed so that the ℓ smallest keys will be held by the first processor, the next ℓ smallest ones by the second processor, and so on, with the processors numbered in row major order. We have the following fact.

FACT 2.1 (see [Kun93]). *Any ℓ-sorting can be performed on an n-node mesh in $O(\ell\sqrt{n})$ time.*

We call *(ℓ_1, ℓ_2)-routing* a routing problem in which each mesh processor is the source of at most ℓ_1 packets and the destination of at most ℓ_2 packets. We have the following fact.

FACT 2.2 (see [SK94]). *Any (ℓ_1, ℓ_2)-routing can be performed on an n-node mesh in $O(\sqrt{\ell_1 \ell_2 n})$ time.*

A simple bisection-based argument shows that this result is optimal in the general case. However, for a special class of (ℓ_1, ℓ_2)-routings, a better performance can be achieved as follows. Fix a tessellation of the mesh into n/s submeshes of s nodes each, and consider an (ℓ_1, ℓ_2)-routing where at most δs packets are destined for each submesh. We first use ℓ_1-sorting and (ℓ_1, δ)-routing to spread the packets evenly among the nodes of their destination submeshes, and then complete the routing by running n/s independent instances of (δ, ℓ_2)-routing within each submesh. The overall routing time becomes

$$O(\ell_1\sqrt{n} + \sqrt{\ell_1 \delta n} + \sqrt{\delta \ell_2 s}).$$

Comparing the $O(\sqrt{\ell_1 \ell_2 n})$ complexity of the general (ℓ_1, ℓ_2)-routing algorithm with the above routing time, we see that the new algorithm is profitable when $\delta, \ell_1 = o(\ell_2)$ and $\delta s = o(\ell_1 n)$. This fact will be exploited in our access protocol, where packet routing is used to access selected copies of the variables. In particular, we will employ several nested tessellations of the mesh and provide strong bounds on the congestion within the submeshes of each tessellation, so that the above strategy can be applied. The packets will then be routed gradually to their destinations through a sequence of smaller and smaller submeshes.

3. The HMOS. This section introduces the HMOS, a mechanism through which m shared variables are distributed among the n memory modules of a processor network. The section is organized in two subsections: section 3.1 presents the logical structure of HMOS, while section 3.2 describes its actual implementation on the mesh.

3.1. Logical structure of the HMOS. The HMOS is structured as $k+1$ levels of logical modules built upon the shared variables, where $k = O(\log \log n)$ is a nonnegative integer function of n to be specified by the analysis. More specifically, starting from $m = n^\tau$ shared variables, for a fixed constant $\tau > 1$, the HMOS comprises m_i modules at level i, called *i-modules* for $0 \leq i \leq k$, where the m_i's are strictly decreasing values that will be specified later. Modules are nested collections of variables, obtained as follows. First, each variable is replicated into $r = \Theta(1)$ copies, which are assigned to distinct 0-modules. The contents of each 0-module, viewed as an indivisible unit, are in turn replicated into 3 copies, which are assigned to distinct 1-modules. In general, the contents of each $(i-1)$-module, viewed as an indivisible unit, are replicated into 3 copies, which are assigned to distinct i-modules, for

$0 < i \leq k$. It is easy to see that the above process will eventually create 3^{k-i} replicas of each i-module and $r3^k$ copies per variable. In the rest of this paper, we will reserve the term *copy* to denote the replica of a variable, and *i-block* to denote the replica of an i-module.

The difference between an i-module and one of its i-blocks is akin to the difference between a variable and one of its copies. Namely, an i-module represents an abstract entity, of which several physical replicas, its i-blocks, exist. Since the contents of k-modules are not replicated, the terms k-module and k-block will be used interchangeably. Note that a k-block is made of $(k-1)$-blocks, which in turn are made of $(k-2)$-blocks, and so on until 0-blocks are reached. The latter contain copies of variables.

Let V denote the set of shared variables, and U_i the set of i-modules for $0 \leq i \leq k$. The HMOS is represented as a leveled *direct acyclic graph* (dag) $\mathcal{H}$, which is defined by a cascade of $k+1$ bipartite graphs, namely (V, U_0) and (U_{i-1}, U_i), $0 < i \leq k$, whose edges are directed left to right. In (V, U_0), each variable $v \in V$ is adjacent to r 0-modules, denoted by $\gamma_0(v, j)$, $1 \leq j \leq r$. For $0 < i \leq k$ in (U_{i-1}, U_i), each $(i-1)$-module u is adjacent to three i-modules denoted by $\gamma_i(u, j)$, $1 \leq j \leq 3$. For notational convenience, we will number the levels in the HMOS starting from -1, the level of the variables. As a consequence, nodes at level i, $0 \leq i \leq k$, are i-modules.

In the HMOS, each variable v uniquely identifies a single-source subdag $\mathcal{H}_v$ induced by all nodes reachable from $v \in V$. A straightforward property of $\mathcal{H}_v$ is that it contains $r3^k$ distinct source-sink paths which are in one-to-one correspondence with the $r3^k$ copies of v. Each source-sink path in $\mathcal{H}_v$ (hence, each copy of v) is uniquely identified by the string of nodes traversed by the path. Moreover, for $0 \leq i \leq k$, the suffix of any such string starting with a node u at level i of $\mathcal{H}_v$, identifies a specific i-block storing a copy of v. Note that several source-sink paths in $\mathcal{H}_v$ may correspond to strings with a common suffix starting from level i. In this case, the i-block corresponding to the common suffix will store several distinct copies of v. A small HMOS for 8 shared variables is shown in Figure 3.1.

The component bipartite graphs of the HMOS must be carefully chosen in order to guarantee a good distribution of the copies of the variables, once the HMOS is mapped onto the processors' memory modules. More specifically, we require that the graphs exhibit good *expansion*, according to the following definition.

DEFINITION 3.1. *Let $G = (X, Y)$ be a bipartite graph, where each input node in X has degree d. For $0 < \alpha \leq 1$, $0 < \epsilon < 1$, and $1 \leq \mu \leq d$, G is said to have* (α, ϵ, μ)-expansion *if for any subset $S \subseteq X$, $|S| \leq \alpha|X|$, and for any set E of $\mu|S|$ edges, μ outgoing edges for each node in S, the set $\Gamma^E(S) \subseteq Y$ reached by the chosen edges has size*

$$|\Gamma^E(S)| = \Omega(|S|^{1-\epsilon}).$$

We let $|U_0| = n^\rho$, where $0 < \rho < 3/2$ is a parameter to be fixed by the analysis. We require that (V, U_0) has input degree r, for a fixed odd constant $r > 0$, output degree $|V|r/|U_0|$, and exhibits (α, ϵ, μ)-expansion, where $\alpha = n/m$, ϵ is a positive constant less than 1, and $\mu = (r+1)/2$. Clearly, a necessary condition for the existence of such a graph is $(\alpha|V|)^{1-\epsilon} = n^{1-\epsilon} \leq n^\rho$, which implies $\rho + \epsilon - 1 \geq 0$. The analysis will determine suitable values for r, ϵ, and ρ that guarantee the existence of (V, U_0). Moreover, an explicit construction for such graph will be available when $m = O(n^{9/2})$.

The graphs (U_{i-1}, U_i) for $0 < i \leq k$ are derived from instances of varying size of the same combinatorial structure, the *balanced incomplete block design* (BIBD), defined below.

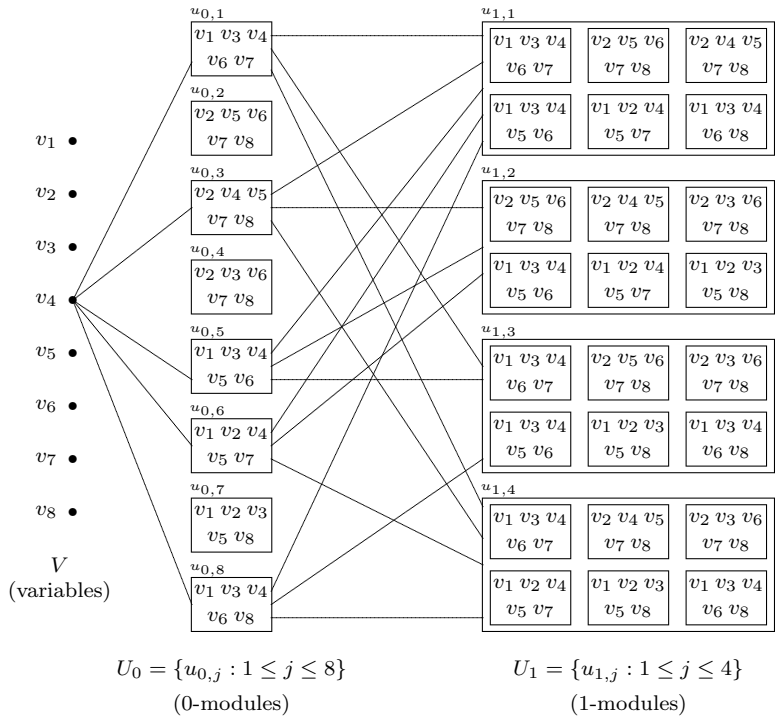


FIG. 3.1. *An HMOS $\mathcal{H}$ built upon 8 shared variables, with $r = 5$ and $k = 1$. There are 8 0-modules and 4 1-modules/1-blocks. Each 1-block contains 6 0-blocks (physical copies of 0-modules), each of which in turn contains 5 copies of the variables. The edges shown in the figure are those of the subdag $\mathcal{H}_{v_4}$. Note that there are 15 edge-disjoint, source-sink paths in $\mathcal{H}_{v_4}$, each path corresponding to one of the 15 copies of v_4.*

DEFINITION 3.2 (see [Hal86]). *A* BIBD *with parameters w and q, or (w, q)-*BIBD*, is a bipartite graph (X, Y) such that*

- *$|Y| = w$;*
- *the degree of each node in X is q;*
- *for any two nodes $y_1, y_2 \in Y$ there is exactly one node $x \in X$ adjacent to both.*

From the definition, it immediately follows that $|X| = w(w-1)/(q(q-1))$ and that the degree of each node in Y is $(w-1)/(q-1)$. One important property of the BIBD, which we will heavily exploit, is stated in the following lemma.

LEMMA 3.3. *Let $G = (X, Y)$ be a (w, q)-BIBD. Consider a node $y \in Y$ and a subset $S \subseteq X$ such that any node in S is adjacent to y. Let E be a set of $\mu|S|$ edges, with $\mu \le q$, containing μ outgoing edges from each x in S, and let $\Gamma^E(S)$ denote the set of nodes of U reached by the chosen edges. Then*

$$|\Gamma^E(S)| \ge (\mu - 1)|S| + 1.$$

Proof. The definition of BIBD implies that no two nodes in S share a neighbor other than y. If $y \notin \Gamma^E(S)$, then $|\Gamma^E(S)| = \mu|S|$. Otherwise, at most $|S|$ of the selected edges reach y; the other $(\mu - 1)|S|$ reach distinct nodes. ∎

COROLLARY 3.4. *A (w, q)-BIBD has $(1, 1/2, \mu)$-expansion for every $2 \le \mu \le q$.*

Proof. Let $G = (X, Y)$ be a (w, q)-BIBD, and let S be an arbitrary subset of input nodes. We now show that $|\Gamma^E(S)| > \sqrt{(\mu - 1)\mu|S|}$ for an arbitrary set E containing

μ outgoing edges from each node in S. Assume that $|\Gamma^E(S)| \leq \sqrt{(\mu-1)\mu|S|}$. Then, there must be an output node in $\Gamma^E(S)$ that is adjacent to at least $\sqrt{|S|\mu/(\mu-1)}$ nodes in S. According to Lemma 3.3, this implies that $|\Gamma^E(S)| \geq (\mu-1)\sqrt{|S|\mu/(\mu-1)}+1 > \sqrt{(\mu-1)\mu|S|}$, which contradicts our assumption. □

For convenience, we assume that both n and $3^k n^\rho$ are even powers of three. Consider the sequence of integers $d_0, d_1, \ldots, d_k$ defined as

$$\begin{cases} d_0 &= \log_3 n^\rho, \\ d_i &= 2\left\lceil \frac{1}{2}\left(\frac{d_{i-1}}{2}+1+k-i\right)\right\rceil - k + i \quad \text{for } 1 \leq i \leq k. \end{cases}$$

For $0 \leq i \leq k$, set the number of i-modules to be $m_i = 3^{d_i}$. The following two properties are easily established.

(i) $d_i + k - i$ is even, for $0 \leq i \leq k$.

(ii) $3\sqrt{m_{i-1}} \leq m_i \leq 16\sqrt{m_{i-1}}$, which implies that $m_{i-1} \leq m_i(m_i-1)/6$ for $1 \leq i \leq k$, and $m_i = \Theta(n^{\rho/2^i})$ for $0 \leq i \leq k$.

We choose (U_{i-1}, U_i) as a subgraph of a $(m_i, 3)$-BIBD, where $m_i(m_i-1)/6 - m_{i-1}$ inputs are removed along with their incident edges. The inputs to be removed are chosen in such a way that the remaining edges are evenly distributed among the outputs, so that each node of U_i becomes adjacent to

$$n_i = \frac{3m_{i-1}}{m_i}$$

nodes of U_{i-1}. An efficient construction of such subgraphs is described in the appendix.

3.2. Mapping the HMOS onto the mesh. The HMOS is physically mapped onto the mesh by storing each i-block in a distinct submesh of appropriate size. For some values of the parameter ρ, the number of 0-blocks exceeds the mesh nodes, hence a single mesh node must store more than one 0-block. There are $3^{k-i}m_i = \Theta(3^{k-i}n^{\rho/2^i})$ i-blocks, $0 \leq i \leq k$, and each i-block contains exactly n_i $(i-1)$-blocks, $1 \leq i \leq k$. We define k nested tessellations of the mesh into submeshes as follows. The outermost tessellation is a subdivision of the mesh into m_k submeshes (*k-submeshes*), each storing a distinct k-block. Each k-submesh is in turn tessellated into n_k $(k-1)$-*submeshes* storing the component $(k-1)$-blocks of the k-block assigned to the k-submesh. In general, for $2 \leq i \leq k$, each *i-submesh*, storing a given i-block, is tessellated into n_i $(i-1)$-*submeshes* storing its component $(i-1)$-blocks. Thus, for $1 \leq i \leq k$, we have a total of

$$m_k n_k n_{k-1} \cdots n_{i+1} = 3^{k-i}m_i = \Theta(3^{k-i}n^{\rho/2^i})$$

i-submeshes, each of size

$$t_i = \frac{n}{3^{k-i}m_i} = \frac{n}{3^{d_i+k-i}} = \Theta(3^{i-k}n^{1-\rho/2^i}).$$

Note that the assumption $k = O(\log\log n)$ ensures $t_i \geq 1$ for $i \geq 1$ and n sufficiently large. Moreover, since both $\log_3 n$ and $d_i + k - i$ are even, we have that t_i is an even power of three, hence $\sqrt{t_i}$ is integral and $\sqrt{t_{i-1}}$ divides $\sqrt{t_i}$. As a consequence, the $(i-1)$-submeshes storing the n_i component $(i-1)$-blocks of an i-block are all contained within the i-submesh storing the i-block. Finally, the organization of the

$3^k n^\rho$ 0-blocks depends on the parameter ρ. When $3^k n^\rho < n$, we assign each 0-block to a submesh of $t_0 = n^{1-\rho}/3^k$ nodes and evenly partition the contents of the block among these nodes. Otherwise, when $3^k n^\rho > n$ there are more 0-blocks than processors, so we assign $3^k n^{\rho-1}$ 0-blocks to each processor. In either case, each processor stores $r3^k m/n$ copies of variables.

4. The access protocol. Suppose that m shared variables are distributed among the n mesh nodes according to the HMOS. In this section, we present the protocol that realizes a parallel access to any n-tuple of variables, where each processor issues a read/write request for a distinct variable. (The case of concurrent accesses can be reduced to the case of exclusive accesses in time $O(\sqrt{n})$ by means of standard sorting-based techniques for leader election and data distribution [Lei92].)

Let S denote the set of variables to be accessed. The access protocol consists of a *copy selection phase* followed by a *routing phase*. In the first phase, a suitable set of copies for the variables in S is chosen, so that accessing these copies will enforce data consistency and generate low memory contention and network congestion. In the subsequent phase, the selected copies are effectively accessed through an appropriate routing strategy. In case of read operations, the accessed data are returned to the requesting processors along the reverse routing paths.

4.1. Copy selection phase. Copy selection achieves the double objective of controlling both memory contention and network congestion by means of a single mechanism. The hierarchical structure of the HMOS provides a geographical distribution of the copies into nested regions of the network. By carefully limiting the number of copies that have to be accessed in any block at any level, we reduce the number of packets that will ever be routed to any such region, which allows us to adopt the efficient routing strategy illustrated in section 2.

4.1.1. A new consistency rule. Recall from section 3.1 that the $r3^k$ copies of a variable v are associated with the source-sink paths of $\mathcal{H}_v$, the subdag of $\mathcal{H}$ induced by v and by all of its descendants. Suppose that we want to read/write v. In order to guarantee consistency, the copies of v to be accessed are selected according to a new rule, which extends the majority protocol of [UW87] to fit the structure of the HMOS. Specifically, we require that the selected copies form a *target set*, which is defined as follows. Let C_v be a set of copies of v, and let $\mathcal{N}(C_v)$ be the set of nodes of $\mathcal{H}_v$ belonging to the source-sink paths associated with these copies. Recall that r is odd, and let $\mu = (r+1)/2$.

DEFINITION 4.1. *C_v is a* target set *for v if $|C_v| = \mu 2^k$ and the following condition holds: a majority (μ) of the nodes at level 0 of $\mathcal{H}_v$ belong to $\mathcal{N}(C_v)$, and, for each node at level i belonging to $\mathcal{N}(C_v)$, a majority (two) of its successors at level $i+1$ belong to $\mathcal{N}(C_v)$ for $0 \le i < k$.*

Figure 4.1 depicts the source-sink paths corresponding to a target set C_{v_4} for variable v_4 in the HMOS of Figure 3.1.

An easy inductive argument shows that any two target sets for the same variable have nonempty intersection. Based on such a property we can guarantee consistency, that is, ensure that a read always returns the most updated value, as follows. As customary in any multicopy approach, we equip each copy with a time-stamp, which is set to the current step whenever the copy is written. A read or write operation on a variable v is simulated by accessing a target set of its copies. By the intersection property of target sets, the copies accessed for reading a variable v must include at least one of the most recently written copies for v, which can be identified by looking

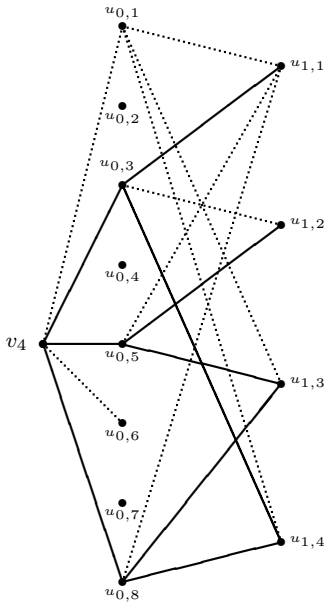


FIG. 4.1. *The source-sink paths (solid lines) in $\mathcal{H}_{v_4}$ corresponding to a target set C_{v_4} of* 6 *copies for variable v_4 in the HMOS $\mathcal{H}$ of Figure* 3.1.

for the most recent time-stamp. It should be noted that a target set contains only $\mu 2^k$ copies out of the $r3^k$ total copies of a variable. Therefore, unlike previous protocols, we maintain consistency by accessing much less than a majority of the copies.

4.1.2. The selection procedure. The copy selection phase determines a target set C_v for each $v \in S$. This is accomplished in $k+1$ iterations, numbered from 0 to k, during which the nodes of the $\mathcal{H}_v$'s are marked in a top-down fashion from the sources to the sinks. More specifically, for every $\mathcal{H}_v$, with $v \in S$, Iteration 0 marks the source v and μ of its successors (0-modules); Iteration i, $0 < i \leq k$, marks two successors (i-modules) of each marked node at level $i-1$. In this fashion, at the end of Iteration i, $0 \leq i \leq k$, the marked nodes in each $\mathcal{H}_v$ form exactly $\mu 2^i$ distinct paths from the source to nodes at level i (in what follows we refer to such paths as *marked paths*). We choose C_v as the set of copies of v corresponding to the $\mu 2^k$ source-sink marked paths in $\mathcal{H}_v$ at the end of the last iteration.

It is important to notice that a node of $\mathcal{H}$, say an i-module u, may belong to several subdags $\mathcal{H}_v$ corresponding to variables in S. During the copy selection procedure, we keep track of u independently in each such subdag; hence, u may result marked in some of the subdags and unmarked in the others. Suppose that, at the end of Iteration i, u is marked in some subdags and that a total of h marked paths in these subdags reach u. This implies that at the end of copy selection there will be $h2^{k-i}$ marked paths that pass through u, that is, $h2^{k-i}$ copies in $\bigcup_{v \in S} C_v$ stored in i-blocks of u. In Iteration $i+1$, the two successors of u to be marked are chosen to be the same for all subdags in which u is marked; hence, for each chosen successor node, u will contribute h paths to the total number of marked paths that will pass through that node at the end of the iteration. The main idea behind the copy selection phase is to control congestion in $(i+1)$-blocks by choosing the nodes to be marked in Iteration $i+1$ in such a way as to keep the number of marked paths passing through each such

node under some reasonable bound.

The following notations will be needed to describe the copy selection procedure.

DEFINITION 4.2. *For $0 \leq i \leq k$, $A_i \subseteq U_i$ denotes the set of i-modules that are marked in some $\mathcal{H}_v$, with $v \in S$, during Iteration i. The modules in A_i are called* active modules.

DEFINITION 4.3. *The* weight $w(u)$ *of an active module $u \in A_i$ is the sum, over all variables $v \in S$, of the number of marked paths from v to u in $\mathcal{H}_v$.*

From the previous discussion we conclude that exactly $w(u)$ copies in $\bigcup_{v \in S} C_v$ will reside in *each* of the selected 2^{k-i} i-blocks of u, while the other i-blocks of u will not contain any copy in $\bigcup_{v \in S} C_v$. Using the expansion properties of the HMOS, we will be able to guarantee suitably low values for the $w(u)$'s.

The actions performed by the copy selection procedure are reported below. Since Iteration 0 is different from the others, it is described separately. In order to understand the parameters used in Iteration 0, recall that we chose (V, U_0) to have $(n/m, \epsilon, \mu)$-expansion, where $0 < \epsilon < 1$ and $\mu = (r+1)/2$. In other words, the graph guarantees that for any set S of at most $|S| \leq n$ variables and any choice of μ 0-modules adjacent to each variable, the overall number of chosen 0-modules is at least $\beta |S|^{1-\epsilon}$ for some constant $\beta > 0$. Finally, recall that each processor is in charge of a distinct variable.

Iteration 0.

1. For $v \in S$, let p_v denote the processor in charge of v. Each p_v creates r *copy-packets* denoted by the tuples $[p_v, v, u_j = \gamma_0(v, j), h_{v,u_j} = 1]$ for $1 \leq j \leq r$. Upon creation, all copy-packets are regarded as *unmarked*.

2. Let $\mathcal{CP}_0$ denote an initially empty set. The following three substeps are executed until μ copy-packets for each variable are put in $\mathcal{CP}_0$.

(i) *Sorting*: Sort all unmarked copy-packets by their third component.

(ii) *Selection*: For each $u \in U_0$, let c_u be the number of copy-packets with third component u in the sorted sequence. If $c_u \leq (2r/\beta)n^{\epsilon}$, then all such packets are marked. Otherwise, none of them is marked. Subsequently, all the packets are sent back to their originating processors.

(iii) *Counting*: For each $v \in S$, p_v counts the total number of its copy-packets marked so far. If these are at least μ, then exactly μ of them are put in $\mathcal{CP}_0$ while the remaining $r - \mu$ copy-packets are discarded. Otherwise, the marked copy-packets are locally buffered.

3. For each $v \in S$, p_v marks the source of $\mathcal{H}_v$ and μ of its successors corresponding to the copy-packets for v included in $\mathcal{CP}_0$.

The analysis will show that the set $\mathcal{CP}_0$ is determined in at most $\log n + 1$ iterations of step 2.

Iteration i $(1 \leq i \leq k)$. At the beginning of Iteration i, the mesh processors store a set $\mathcal{CP}_{i-1}$ of copy-packets. Specifically, each processor p_v stores copy-packets of type $[p_v, v, u, h_{v,u}]$, where $u \in A_{i-1}$ and $\sum_u h_{v,u} = \mu 2^{i-1}$. The value $h_{v,u}$ reflects the *multiplicity* of u with respect to v, that is, the number of distinct marked paths in $\mathcal{H}_v$ from v to u. Iteration i consists of the following steps.

1. The copy-packets in $\mathcal{CP}_{i-1}$ are sorted by their third component.

2. For each group of packets with the same third component $u \in A_{i-1}$, a *leader* processor p_u is elected. Each p_u computes $w(u)$ as the sum of the multiplicities carried by the packets in its group, and creates the three *module-packets* $[p_u, u, \gamma(u, j), w(u)]$ for $1 \leq j \leq 3$.

3. The $3|A_{i-1}|$ module-packets are sorted lexicographically by their third and

fourth components.

4. For each $x \in U_i$, a maximal subset $\mathcal{P}_x$ of module-packets with third component x is chosen, such that

$$\sum_{[p_u,u,x,w(u)]\in\mathcal{P}_x} w(u) \le c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}}\right).$$

5. The chosen packets are sent back to their originating leader processors.

6. Assume that a leader processor p_u receives $h \le 3$ module-packets back. If $h < 2$, then p_u selects $2-h$ extra module-packets from the $3-h$ that were not chosen in the previous step. Otherwise, p_u selects two module-packets among those received.

7. Let $[p_u, u, \gamma(u, j_1), w(u)]$ and $[p_u, u, \gamma(u, j_2), w(u)]$ be the two module-packets selected by p_u. Processor p_u sends the names $\gamma(u, j_1)$ and $\gamma(u, j_2)$ to the processors storing the copy-packets in its group.

8. Each copy-packet $[p_v, v, u, h_{v,u}] \in \mathcal{CP}_{i-1}$ is augmented with two extra components containing $\gamma(u, j_1)$ and $\gamma(u, j_2)$.

9. The augmented copy-packets are routed back to the processors in charge of their respective variables.

10. For any augmented copy-packet $[p_v, v, u, h_{v,u}, \gamma(u, j_1), \gamma(u, j_2)]$ received at processor p_v, the nodes $\gamma(u, j_1)$ and $\gamma(u, j_2)$ at level i of $\mathcal{H}_v$ are marked (note that the same node may be redundantly marked more than once). Moreover for each newly marked node u' at level i, a copy-packet $[p_v, v, u', h_{v,u'}]$ is created, where $h_{v,u'}$ is obtained by summing up the multiplicities of the received (augmented) copy-packets carrying u' in one of the two extra components. These new copy-packets form the set $\mathcal{CP}_i$, while all other packets are discarded.

When Iteration k terminates, for each $v \in S$, p_v determines the set C_v of copies to be accessed as those corresponding to the $\mu 2^k$ source-sink marked paths in $\mathcal{H}_v$. It is easily seen that for each $v \in S$, the set C_v computed by the copy selection procedure is indeed a target set for v.

4.1.3. Analysis of the selection procedure. We now determine the running time of the selection procedure described above. Let us first consider Iteration 0. By Fact 2.1 and since $r = O(1)$, steps 1 and 3 require altogether $O(\sqrt{n})$ time. The sorting, selection and counting substeps of step 2 can be implemented in terms of sorting and prefix operations in $O(\sqrt{n})$ time. It will be shown in Lemma 4.4 that $\log n + 1$ executions of such substeps are sufficient. Therefore, Iteration 0 requires $O(\sqrt{n}\log n)$ time altogether. For $i \ge 1$, Iteration i can be implemented in terms of a constant number of sorting and prefix operations on a set of $O(\mu 2^{i-1}n)$ packets, yielding a running time of $O(\sqrt{n}2^{i-1})$. Therefore, copy selection is completed in time

$$O\left(\sqrt{n}\log n + \sqrt{n}\sum_{i=1}^{k} 2^{i-1}\right) = O\left(\sqrt{n}\left(\log n + 2^k\right)\right). \tag{4.1}$$

LEMMA 4.4. *After* $\log n + 1$ *executions of step* 2 *in Iteration* 0, *the set* $\mathcal{CP}_0$ *contains exactly* μn *copy-packets. Moreover, for each active* 0*-module* $u \in A_0$, $w(u) \le (2r/\beta)n^{\epsilon}$.

Proof. Let S_j be the number of variables for which fewer than μ copy packets have been selected by the end of the jth execution of step 2. For the sake of convenience, set $S_0 = |S| = n$. We now show by induction that

$$S_j \le \frac{n}{2^j}$$

for any $j \geq 0$, and this will imply that for $T = \log n + 1$, $S_T = 0$. The inequality for S_0 is immediate, establishing the basis. Assume that the inequality holds for $j-1$, and suppose, for a contradiction, that $S_j > n/2^j$. By using the expansion properties of (V, U_0), it is easy to see that in the jth execution of the selection substep, at least $\beta S_j^{1-\epsilon}$ 0-modules are addressed by unmarked copy-packets. All such 0-modules must have been congested in the previous iteration (i.e., addressed by more than $(2r/\beta)n^\epsilon$ copy-packets), which accounts for a total of at least

$$\frac{2r}{\beta} n^\epsilon \beta S_j^{1-\epsilon} > r S_{j-1}$$

unmarked copy-packets involved in that iteration. However, this is impossible since S_{j-1} variables account for at most rS_{j-1} copy packets.

The bound on $w(u)$ is easily established by observing that for each 0-module u, all copy-packets with third component u that are added to $\mathcal{CP}_0$ are marked during the same iteration of step 2. Therefore

$$w(u) \leq \frac{2r}{\beta} n^\epsilon. \qquad \square$$

It must be remarked that the copy selection phase can be improved in a number of ways to obtain a faster running time at the expense of a more complex implementation. However, to avoid further complications to the presentation, we chose to describe a simpler yet slightly less efficient implementation, since, as shown in section 5, its complexity does not influence the overall running time of the access protocol.

To complete the analysis, it remains to establish the bound on the weight $w(u)$ of any $u \in A_i$, at the end of Iteration i. Recall that the sum of the multiplicities of the copy-packets in $\mathcal{CP}_i$ with third component u yields $w(u)$. Therefore, for $0 \leq i \leq k$,

$$\sum_{u \in A_i} w(u) = \mu 2^i n. \tag{4.2}$$

LEMMA 4.5. *There is a suitable constant $c \geq 3$ such that, at the end of Iteration i, for each $u \in A_i$, $0 \leq i \leq k$,*

$$w(u) \leq c\mu 2^i n^{1-\frac{1-\epsilon}{2^i}}.$$

Proof. The proof proceeds by induction on i. The basis ($i = 0$) is established by Lemma 4.4. Suppose that the inequality holds for $i-1$, and let x be an i-module. The weight of x is determined by steps 4 and 6 of Iteration i. More precisely, recall that $\mathcal{P}_x$ is the set of module-packets of kind $[p_u, u, x, w(u)]$ selected in step 4, and let $\mathcal{P}'_x$ be the set of additional module-packets (still with third component x) not in $\mathcal{P}_x$, selected in step 6. It is easy to see that

$$w(x) \leq \sum_{[p_u, u, x, w(u)] \in \mathcal{P}_x \cup \mathcal{P}'_x} w(u).$$

Because of the way the module-packets are selected in step 4, we already know that

$$\sum_{[p_u, u, x, w(u)] \in \mathcal{P}_x} w(u) \leq c\mu 2^{i-1} \left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}} \right).$$

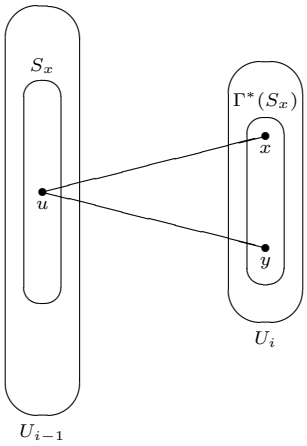


FIG. 4.2. *Critical modules in the proof of Lemma* 4.5.

We must only show that the contribution of $\mathcal{P}'_x$ to $w(x)$ is not too large. In order to derive a contradiction, we suppose that $w(x) > c\mu 2^i n^{1-\frac{1-\epsilon}{2^i}}$. This implies

$$\sum_{[p_u,u,x,w(u)]\in\mathcal{P}'_x} w(u) > c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} - n^{1-\frac{1-\epsilon}{2^{i-1}}}\right).$$

Define $S_x = \{u \in A_{i-1} \,:\, [p_u, u, x, w(u)] \in \mathcal{P}'_x\}$, so that

$$\sum_{[p_u,u,x,w(u)]\in\mathcal{P}'_x} w(u) = \sum_{u\in S_x} w(u).$$

By the inductive hypothesis, the weight of each $u \in S_x$ is at most $c\mu 2^{i-1} n^{1-\frac{1-\epsilon}{2^{i-1}}}$; therefore

$$|S_x| \geq \frac{\sum_{u\in S_x} w(u)}{c\mu 2^{i-1} n^{1-\frac{1-\epsilon}{2^{i-1}}}} > n^{\frac{1-\epsilon}{2^i}} - 1.$$

Note that for each $u \in S_x$, at least two module-packets, including $[p_u, u, x, w(u)]$, have not been selected in step 4. Let

$$\Gamma^*(S_x) = \{y \in U_i \,:\, \exists\, u \in S_x \text{ s.t. } [p_u, u, y, w(u)] \text{ has not been selected in step 4}\}.$$

Note that in the graph (U_{i-1}, U_i) each $u \in S_x$ is adjacent to either two or three nodes in $\Gamma^*(S_x)$, one of which is x (see Figure 4.2).

Since (U_{i-1}, U_i) is a BIBD, we can apply Lemma 3.3 and conclude that $|\Gamma^*(S_x)| \geq |S_x|+1 > n^{\frac{1-\epsilon}{2^i}}$. We now show that the global weight assigned in step 4 to all the nodes in $\Gamma^*(S_x)$ exceeds the total weight carried by all the module-packets, thereby leading to a contradiction. Let $y \in \Gamma^*(S_x)$, and let $[p_{u'}, u', y, w(u')]$ be a module-packet which has not been selected in step 4, with $u' \in S_x$. Then, we must have

$$w(u') + \sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) > c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}} + n^{1-\frac{1-\epsilon}{2^{i-1}}}\right);$$

that is,

$$\sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) > c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}}+n^{1-\frac{1-\epsilon}{2^{i-1}}}\right)-c\mu 2^{i-1}n^{1-\frac{1-\epsilon}{2^{i-1}}}=c\mu 2^{i-1}n^{1-\frac{1-\epsilon}{2^i}}.$$

Adding up the contributions of all nodes in $\Gamma^*(S_x)$, we get

$$\begin{aligned}\sum_{y\in\Gamma^*(S_x)}\sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u) &> |\Gamma^*(S_x)|c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}}\right)\\ &> n^{\frac{1-\epsilon}{2^i}}c\mu 2^{i-1}\left(n^{1-\frac{1-\epsilon}{2^i}}\right)=c\mu 2^{i-1}n.\end{aligned}$$

Since $c\geq 3$, the above inequality leads to a contradiction because

$$\sum_{y\in\Gamma^*(S_x)}\sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u)\leq\sum_{y\in U_i}\sum_{[p_u,u,y,w(u)]\in\mathcal{P}_y} w(u)\leq 3\sum_{u\in A_{i-1}} w(u)=3\mu 2^{i-1}n,$$

where the last equality follows from (4.2). □

4.2. Routing phase. After copy selection is completed, the copies in $\bigcup_{v\in S}C_v$ have to be accessed. Each request is encapsulated in a distinct packet, routed from the requesting processor (*origin*) to the processor storing the copy (*destination*), and back to the origin. The idea is to route the packets in stages so that they are moved gradually closer to their destinations through smaller and smaller submeshes, in accordance with the tessellations defined on the mesh. As argued in section 2, when the number of packets destined for any submesh is not too large, such a strategy yields more profitable results than sending the packets directly to their destinations.

The origin-destination part of a packet's journey consists of $k+2$ routing stages, numbered from $k+1$ down to 0. Stage i, with $k+1\geq i\geq 1$, is executed in parallel and independently in every i-submesh (here, the whole mesh is viewed as a $(k+1)$-submesh). In this stage the packets are routed to arbitrary positions in the $(i-1)$-submeshes hosting their destination $(i-1)$-blocks, in such a way that the processors of each submesh receive approximately the same number of packets. This can be achieved by first sorting the packets according to their destination submeshes and then ranking the packets destined to the same submesh. Observe that when $3^k n^\rho < n$, a 0-block is assigned to a 0-submesh of $t_0 = n^{1-\rho}/3^k$ nodes. By the end of stage 1, each packet reaches a processor within its destination 0-submesh, and in stage 0 is sent to its final destination. Instead, when $3^k n^\rho \geq n$, there are $n^{\rho-1}3^k$ 0-blocks stored within a single processor, hence each packet is at its final destination by the end of stage 1, and stage 0 is not needed. In either case, once the packet reaches its final destination, the request it carries is satisfied.

In order to estimate the time complexity of the above protocol, we need to determine the maximum number of packets sent and received by any processor in each stage. More formally, let δ_i, for $k+1\geq i\geq 0$, denote the maximum number of packets held by any processor at the beginning of stage i. Let also δ_{-1} be the maximum number of packets received by a processor at the end of stage 0, when such stage is needed (i.e., when $3^k n^\rho < n$). We have the following lemma.

LEMMA 4.6. *Let $k\geq 0$. Then*

$$\begin{aligned}\delta_{k+1} &= \mu 2^k,\\ \delta_i &= O\left(\mu 2^i 3^{k-i} n^{\frac{\rho+\epsilon-1}{2^i}}\right) \qquad \text{for } k\geq i\geq 0.\end{aligned}$$

When $3^k n^\rho < n$, we also have $\delta_{-1} = O(\mu n^\epsilon)$.

Proof. The statement is immediately evident for δ_{k+1}, since every target set contains $\mu 2^k$ copies. By Lemma 4.5, an i-block is addressed by at most $c\mu 2^i n^{1-(1-\epsilon)/2^i}$ packets for $k \geq i \geq 1$. Since there are $t_i = \Theta(3^{i-k} n^{1-\rho/2^i})$ processors storing an i-block, we have

$$\delta_i \leq \frac{c\mu 2^i n^{1-\frac{1-\epsilon}{2^i}}}{t_i} = O\left(\mu 2^i 3^{k-i} n^{\frac{\rho+\epsilon-1}{2^i}}\right).$$

In order to establish the bound for δ_0, we distinguish between two cases. If $3^k n^\rho < n$, each 0-block is assigned to a submesh of $t_0 = n^{1-\rho}/3^k$ nodes and, by Lemma 4.5, is addressed by at most $c\mu n^\epsilon$ packets, whence $\delta_0 = c\mu n^\epsilon / t_0 = O(\mu 3^k n^{\rho+\epsilon-1})$. In this case, stage 0 is needed to bring the packets to their final destinations. At the end of this stage, each processor receives at most $\delta_{-1} \leq c\mu n^\epsilon$ packets. Otherwise, if $3^k n^\rho \geq n$, there are $3^k n^{\rho-1}$ 0-blocks stored within a single processor, from which $\delta_0 = c\mu n^\epsilon 3^k n^{\rho-1} = O(\mu 3^k n^{\rho+\epsilon-1})$ as before, but the routing terminates with stage 1. ∎

Set $t_{k+1} = n$, the size of the entire mesh, and let T_i be the time complexity of stage i for $k+1 \geq i \geq 0$.

LEMMA 4.7. *We have*

$$T_{k+1} = O(2^k n^{\frac{1}{2}+\frac{\rho+\epsilon-1}{2^{k+1}}}),$$
$$T_i = O(2^i 3^{\frac{k-i}{2}} n^{\frac{1}{2}+\frac{2\rho+3\epsilon-3}{2^{i+1}}}) \qquad \text{for } k \geq i \geq 1.$$

When $3^k n^\rho < n$, we also have $T_0 = O(n^\epsilon)$.

Proof. Recall that for $k+1 \geq i \geq 1$, stage i is executed in parallel and independently in each i-submesh. The initial sorting and ranking are accomplished in $O(\delta_i \sqrt{t_i})$ time. By Fact 2.2, the subsequent (δ_i, δ_{i-1})-routing requires $O(\sqrt{\delta_i \delta_{i-1} t_i})$ time. Since $\delta_i \leq \delta_{i-1}$, we get $T_i = O(\sqrt{\delta_i \delta_{i-1} t_i})$. When $3^k n^\rho < n$, stage 0 consists of a (δ_0, δ_{-1})-routing in each submesh of size t_0, requiring $O(\sqrt{\delta_0 \delta_{-1} t_0})$ time. The lemma follows by plugging in the values for the δ_i's and the t_i's and by recalling that μ is a constant. ∎

After reaching their destinations, the packets relative to read operations must return to their origins carrying the accessed data. This second part of the routing can be accomplished by running the above protocol backwards, thus maintaining the same time complexity.

THEOREM 4.8. *The access protocol requires overall time*

$$O\left(n^\epsilon + n^{\frac{1}{2}}\left(\log n + 2^k n^{\frac{\rho+\epsilon-1}{2^{k+1}}} + \sum_{i=1}^{k} 2^i 3^{\frac{k-i}{2}} n^{\frac{2\rho+3\epsilon-3}{2^{i+1}}}\right)\right).$$

Proof. The running time of the access protocol is obtained by adding the contributions of the copy selection and routing phases together. The complexity of copy selection is given by (4.1), while the routing time is obtained by the summing of the T_i's given in Lemma 4.7. Note that $\rho + \epsilon - 1 \geq 0$; therefore the term $n^{1/2} 2^k n^{(\rho+\epsilon-1)/2^{k+1}}$ dominates the term $n^{1/2} 2^k$ coming from copy selection. Note also that the term n^ϵ, which accounts for the complexity of stage 0, does not dominate when $3^k n^\rho \geq n$. ∎

5. Tuning of the parameters. The complexity of the access protocol established by Theorem 4.8 is a function of the following design parameters:

(i) k: there are $k+1$ levels in the HMOS;
(ii) ρ: there are n^ρ 0-modules in U_0;
(iii) ϵ: the first graph of the HMOS (V, U_0) has $(n/m, \epsilon, \mu)$-expansion;
(iv) r: the (odd) input degree of (V, U_0).

Furthermore, recall that $m = |V| = n^\tau$ for some constant $\tau > 1$ and that $\mu = (r+1)/2$.

The goal of this section is to determine suitable values of the above parameters that guarantee the existence of graph (V, U_0) and yield a good performance of the access protocol. Such performance is closely related to the redundancy of the HMOS, that is, the number of copies ($r3^k$) used per variable. On the one hand, using many copies per variable yields better access times, while, on the other, lower redundancy yields simpler and space-efficient schemes. We will consider two scenarios: in the first scenario, we optimize the parameters under the assumption that the number of copies for each variable $r3^k$ can grow arbitrarily large. In the second scenario, we optimize under the restriction that the scheme uses no more than a constant number of copies for each variable.

We need the following technical result, which is a straightforward adaptation of Lemma 4 in [PP97a].

LEMMA 5.1. *Let $m = n^\tau$, with constant $\tau > 1$. There is a suitable constant $c > 0$ such that for any odd constant $r \geq c\tau \log \tau$, a random bipartite graph $G = (V, U_0)$ with $|V| = m$, $|U_0| = n$, input degree r, and output degree mr/n has $(n/m, \epsilon, \mu)$-expansion with $\mu = (r+1)/2$ and $\epsilon = (\tau - 1)/\mu$, with high probability.*

We are now ready to prove one of the main results of this paper, which was stated in section 1.1.

Proof of Theorem 1.1. We fix $\rho = 1$ and choose r to be the smallest odd integer greater than $\max\{c\tau \log \tau, 6(\tau - 1)\}$. For such values, Lemma 5.1 ensures the existence of (V, U_0) with $(n/m, \epsilon, (r+1)/2)$-expansion, where $\epsilon \leq 1/3$. Since $2\rho + 3\epsilon - 3 \leq 0$, the complexity of the access protocol given in Theorem 4.8 becomes

$$T = O\left(2^k n^{\frac{1}{2} + \frac{\epsilon}{2^{k+1}}}\right). \tag{5.1}$$

By fixing $k = \max\{0, \lfloor \log_2(\epsilon/\eta) \rfloor\}$, we have $2^{k+1} \geq \epsilon/\eta$, whence $T = O(n^{1/2+\eta})$ and $R = r3^k = O(1/\eta^{\log_2 3}) = O(1/\eta^{1.59})$. By instead fixing $k = \log_2 \log_2 n + O(1)$, so that $2^{k+1} \geq \epsilon \log_2 n$, we have $T = O(n^{\frac{1}{2}} \log n)$ and $R = O(\log^{1.59} n)$. □

As already noted before, the HMOS underlying the above result is fully constructive, except for the first graph (V, U_0), for which Lemma 5.1 only guarantees existence. In practice, one can resort to a random graph for (V, U_0), which, as the lemma shows, will exhibit the required expansion property with high probability. Although no explicit construction for (V, U_0) is known in the general case, this graph needs only weak expansion, which makes it more amenable to explicit constructions than the graphs employed in previous schemes (e.g., [UW87, AHMP87]).

In fact, an explicit construction for (V, U_0) can be obtained when the shared memory size m is within certain ranges. For example, [PP97] shows how to construct a bipartite graph with $m = \Theta(n^{3/2})$ inputs, n outputs, and input degree $r = 3$, which has $(n/m, 1/3, 2)$-expansion. This graph can be efficiently represented using constant storage per node. Thus, using this graph as (V, U_0) when $m = \Theta(n^{3/2})$, the result of Theorem 1.1 still holds, and the HMOS becomes fully constructive.

A larger range of values for m for which the HMOS can be made fully constructive, still yielding nontrivial performance, can be obtained by employing other graphs for

(V, U_0). This is shown below, thus proving Theorem 1.2, which was stated in the introduction.

Proof of Theorem 1.2. Let us consider first the case $\tau \leq 3/2$. We assume $m = x(x-1)/6$, where x is an even power of three. The argument for different values of m requires only trivial modifications. Fix $n^\rho = x = \Theta(n^{\tau/2})$, and choose (V, U_0) as an $(n^\rho, 3)$-BIBD. By Corollary 3.4, such graph has $(1, \epsilon, 2)$-expansion, with $\epsilon = 1/2$. Since $2\rho + 3\epsilon - 3 = O(1/\log n)$, the complexity of the access protocol is still given by (5.1), and the same argument used to prove Theorem 1.1 carries through. Consider now the range $3/2 < \tau \leq 13/6$, and choose n^ρ and (V, U_0) as before. By plugging $\epsilon = 1/2$ and $n^\rho = \Theta(n^{\tau/2})$ in the complexity formula given in Theorem 4.8 and choosing $k = O(1)$ large enough and even, the complexity of the access protocol becomes

$$T = O(n^{\frac{1}{2} + \frac{2\tau - 3}{8}}) = O(n^{\frac{2\tau+1}{8}}).$$

For $\tau \geq 13/6$, it is convenient to choose (V, U_0) as a $3-(n^\rho, 5, 3)$ *design*, a graph with the following properties: $|U_0| = n^\rho$, each node of V has degree 5, and for every three distinct nodes u_1, u_2, u_3 of U_0 there are exactly 3 nodes of V adjacent to all three of them. This implies that $m = |V| = \Theta(n^{3\rho})$. (See [Hal86] for a formal definition of the graph). In [PP97], an explicit construction for the graph is provided, and it is shown that it has $(1, 2/3, 3)$-expansion. With this choice for (V, U_0), we can plug $\epsilon = 2/3$ and $n^\rho = \Theta(n^{\tau/3})$ in the formula of Theorem 4.8, and by choosing $k = O(1)$ large enough, we get access time

$$T = O\left(n^{\frac{2}{3}} + n^{\frac{1}{2} + \frac{2\tau/3 - 1}{4}}\right) = O\left(n^{\frac{2}{3}} + n^{\frac{2\tau+3}{12}}\right).$$

This yields $T = O(n^{2/3})$ for $13/6 \leq \tau \leq 5/2$ and $T = O(n^{(2\tau+3)/12})$ for $5/2 \leq \tau \leq 9/2$, which completes the proof. □

Note that the access time of the constructive scheme tends to $O(n)$ as m approaches $n^{9/2}$, a performance that can be obtained through a straightforward scheme.

6. Extension to other architectures. A closer look at the access protocol developed in the previous sections for the mesh reveals that it solely relies upon a recursive decomposition of the network into subnetworks of the same type, and upon ℓ-sorting and (ℓ_1, ℓ_2)-routing primitives. As a consequence, our scheme can be ported to any network topology that exhibits a suitable decomposition into subnetworks, and for which an efficient implementation of the above primitives is available. In this section we briefly discuss the porting of the scheme to the pruned butterfly and to multidimensional meshes.

An n-leaf *pruned butterfly*, introduced in [BB95], is a variant of Leiserson's fat-tree [Lei85]. Its coarse structure may be interpreted as an n-leaf complete binary tree, where the leaves represent the processor-memory nodes of the machine, the internal nodes represent clusters of routing switches, and the edges represent channels whose bandwidth doubles every other level from the leaves to the root. More precisely, each subtree of n' leaves is connected to its parent through a channel of capacity $\Theta(\sqrt{n'})$. The pruned butterfly is an important interconnection since it is *area-universal* in the sense that it can route any set of messages almost as efficiently as any circuit of similar area.

It follows from the definition that an n-leaf pruned butterfly can be decomposed into 4^i $(n/4^i)$-leaf pruned butterflies connected through channels of capacity $\sqrt{n/4^i}$, a decomposition similar to the one of the mesh employed in our scheme. Moreover,

it is shown in [HPP95] that ℓ-sorting and (ℓ_1, ℓ_2)-routing can be performed on the pruned butterfly in the same running time as on the mesh. This immediately implies that both Theorem 1.1 and Theorem 1.2 also hold for the pruned butterfly.

We now consider the extension of the scheme to d-dimensional meshes, with d constant. For $d \geq 3$, a decomposition of an n-node d-dimensional mesh into submeshes is obtained as an immediate generalization of the two-dimensional case. As for the primitives, ℓ-sorting and (ℓ_1, ℓ_2)-routing, with $\ell_1 < \ell_2$, require time $\Theta(\ell n^{1/d})$ and $\Theta(\ell_2^{1-1/d}(n\ell_1)^{1/d})$, respectively [SK94]. Then, the same argument presented in section 4.2 shows that the access protocol can be executed on a d-dimensional mesh in time

$$T = O\left(n^{\epsilon} + n^{\frac{1}{d}}\left(\log n + 2^k n^{\frac{(d-1)(\rho+\epsilon-1)}{d2^k}} + \sum_{i=1}^{k} 2^i 3^{\frac{(d-1)(k-i)}{d}} n^{\frac{2(d-1)(\rho+\epsilon-1)+\epsilon-1}{d2^i}}\right)\right).$$

Let us fix $\rho = 1$ and $\epsilon < 1/(2d-1)$, which, based on Lemma 5.1, requires $r = \Omega(d\tau)$ in order to guarantee the existence of the first graph (V, U_0) of the HMOS. Straightforward calculations show that the above formula becomes

$$T = O\left(2^k n^{\frac{1}{d} + \frac{(d-1)\epsilon}{d2^k}}\right).$$

Arguing as in the proof of Theorem 1.1, we can prove the following result.

THEOREM 6.1. *For any constant $\tau \geq 1$, there exists a scheme to distribute $m = n^{\tau}$ shared variables among the local memory modules of an n-node d-dimensional mesh (d constant) with redundancy R so that any n variables can be read/written in time*

$$T = O\left(n^{\frac{1}{d}+\eta}\right)$$

for any constant $\eta > 0$, with $R = O(1/\eta^{1.59})$, or in time

$$T = O\left(n^{\frac{1}{d}} \log n\right)$$

with $R = O(\log^{1.59} n)$.

It has to be remarked that the bandwidth of a d-dimensional mesh increases with d; hence, in order to achieve access time close to the natural $\Omega(n^{1/d})$ lower bound, the expansion required of (V, U_0) must also increase with d. For this reason, the graphs for which an explicit construction is currently available do not exhibit sufficient expansion to grant a generalization of Theorem 1.2; however, they can still be used to yield fully constructive schemes with nontrivial $O(n^{1/d+\xi_d})$ access time for suitable constants $\xi_d < (d-1)/d$. The details follow from tedious yet trivial arithmetic manipulations, which are omitted for the sake of brevity.

7. Conclusions. In this paper, we devised a scheme for implementing a shared address space on a mesh of processor/memory pairs. The scheme enables the processors to read/write any n-tuple of shared variables concurrently and yields a quasi-optimal access time in the worst case. One of the most relevant novelties of our implementation is represented by the hierarchical memory organization scheme, the HMOS, which provides a structured distribution of copies of the shared variables among the memory modules. In particular, the HMOS succeeds in the following objectives, which were not attained by the memory organizations known in the literature: (i) it provides

a single mechanism to cope with both memory contention and network congestion. In this fashion, copy selection can be employed to reduce both; (ii) it yields fast access time by using a cascade of bipartite graphs with weak expansion, rather than using one graph of maximum expansion, which greatly simplifies the implementation. Indeed, the HMOS is fully constructive and yields quasi-optimal performance for any memory size $m = O(n^{3/2})$, which is sufficient, for example, to run any NC algorithm. For large memory sizes, the HMOS embodies only one nonconstructive graph of weak expansion.

The design of the HMOS is not specifically cast for the mesh topology. We showed that it can be implemented on the pruned butterfly and on d-dimensional meshes yielding good performance. More generally, our scheme is efficiently portable to any low-bandwidth interconnection where routing takes advantage of partitions of the processors into subnetworks, in the sense that it achieves higher performance by moving messages gradually closer to their destinations through smaller and smaller subnetworks, rather than by sending them directly to their destinations.

A challenging and long-standing open problem remains the construction of bipartite graphs that exhibit good expansion. The availability of explicit constructions and concise representations for such graphs is crucial for attaining simple and efficient deterministic shared memory implementations for all memory sizes. Recent developments in this area [PP97] seem to indicate that the construction of graphs with a linear number of edges and moderate expansion, such as those required in our scheme, are easier than the construction of the highly expanding graphs used in previous schemes. If this is true, our scheme could become a general and constructive tool for the implementation of shared memory on distributed memory machines based on low-bandwidth interconnections.

Finally, we wish to point out that in a recent paper [HPP95], which appeared after the results in the present paper were first presented [PPS94, PP95], a shared memory implementation scheme for the mesh is devised that, through a novel and complex protocol, achieves $O(\sqrt{n \log n})$ access time. However, this scheme relies on a nonconstructive graph of maximum expansion; hence it suffers from the same limitations affecting other schemes in the literature, as discussed in the introduction. The paper also proves an $\Omega(\sqrt{n \log(m/n^2)/\log\log(m/n^2)})$ lower bound on the access time of any deterministic scheme for implementing $m = \Omega(n^2)$ shared variables. The lower bound assumes that variables are accessed through a point-to-point protocol, which requires that a processor dispatch a separate message for each copy it wants to update. The assumption is satisfied by the scheme presented in this paper, which implies that our access time is only a sublogarithmic factor away from optimal.

Appendix. In this appendix, we show how to construct a bipartite graph $G = (X, Y)$ which is a subgraph of a (q^d, q)-BIBD with the same number of output nodes, i.e., $|Y| = q^d$, fewer input nodes, say $|X| = m$, $1 \leq m < q^{d-1}(q^d - 1)/(q - 1)$, and such that each input $x \in X$ has degree q, as in the original BIBD, and each output $y \in Y$ has degree ρ, with

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \leq \rho \leq \left\lceil \frac{qm}{q^d} \right\rceil .$$

As explained in section 3.1, these subgraphs (with $q = 3$ and m a power of three) govern the assignment of replicas of $(i - 1)$ modules to i-modules in the HMOS, for $1 \leq i \leq k$. The construction is obtained by modifying the one for a (q^d, q)-BIBD given in [PP93].

Let q be a prime power and let $\mathbb{F}_q$ be the finite field with q elements, with its elements represented by the integers $0, 1, \ldots, q-1$. The q^d output nodes of the BIBD are associated with the set of d-dimensional vectors over $\mathbb{F}_q$, and the inputs with the $q^{d-1}(q^d-1)/(q-1)$ pairs of vectors of kind

$$\begin{array}{llllllll} (a_{d-2}, & \ldots, & a_h, & 0, & a_{h-1}, & \ldots, & a_1, & a_0) \\ (0, & \ldots, & 0, & 1, & b_{h-1}, & \ldots, & b_1, & b_0), \end{array}$$

where the a_i's and b_i's are elements of the field, and h ranges between 0 and $d-1$. For convenience, each such pair will be denoted by $\chi(h, A, B)$, where A is the integer in $[0, q^{d-1})$ whose representation in base q is $(a_{d-2} \ldots a_h a_{h-1} \ldots a_1 a_0)$, and B is the integer in $[0, q^h)$ whose representation in base q is $(b_{h-1} \ldots b_1 b_0)$. The subgraph G is obtained from this BIBD by taking the same output set and selecting a subset of m inputs as follows. Let $\ell < d$ be the index such that

$$q^{d-1}\frac{q^\ell - 1}{q-1} \le m < q^{d-1}\frac{q^{\ell+1}-1}{q-1},$$

so that

$$m = q^{d-1}\left(\frac{q^\ell - 1}{q-1} + w\right) + z \tag{A.1}$$

for some w, $0 \le w < q^\ell$ and z, $0 \le z < q^{d-1}$. The m pairs $\chi(h, A, B)$ that we select to represent the nodes of X consist of the union of the three sets X_1, X_2, and X_3 defined below:

$$\begin{aligned} X_1 &= \left\{\chi(h, A, B) : 0 \le h < \ell,\ 0 \le A < q^{d-1},\ 0 \le B < q^h\right\}; \\ X_2 &= \left\{\chi(h, A, B) : h = \ell,\ 0 \le A < q^{d-1},\ 0 \le B < w\right\}; \\ X_3 &= \left\{\chi(h, A, B) : h = \ell,\ 0 \le A < z,\ B = w\right\}. \end{aligned}$$

It is easy to verify that $|X_1| + |X_2| + |X_3| = m$.

The edges are defined as follows: the input node

$$\begin{array}{llllllll} (a_{d-2}, & \ldots, & a_h, & 0, & a_{h-1}, & \ldots, & a_1, & a_0) \\ (0, & \ldots, & 0, & 1, & b_{h-1}, & \ldots, & b_1, & b_0) \end{array}$$

is adjacent to the q outputs

$$(a_{d-2}, \ldots, a_h, x, a_{h-1} + x \cdot b_{h-1}, \ldots, a_1 + x \cdot b_1, a_0 + x \cdot b_0)$$

for every $x \in \mathbb{F}_q$, where $+$ and $\cdot$ denote the field operations. We now show that the edges in G are evenly distributed among the outputs.

THEOREM A.1. *Any node $u \in Y$ is connected to ρ nodes of X, where*

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \le \rho \le \left\lceil \frac{qm}{q^d} \right\rceil.$$

Proof. Let u be associated with the vector $(a_{d-1}, \ldots, a_0)$. We determine the value of ρ by separately counting the contributions of the nodes in the three subsets X_1, X_2, and X_3. Consider X_1 and fix $h < \ell$. Using the properties of field operations, one can easily show that for any B, $0 \le B < q^h$, there exists exactly one value

A such that the node $\chi(h, A, B)$ is connected to u. Therefore, there are exactly $\sum_{h=0}^{\ell-1} q^h = (q^\ell - 1)/(q-1)$ nodes of X_1 connected to u. A similar argument shows that exactly w nodes of X_2 are connected to u. Finally, it can be seen that the z nodes of X_3 are connected to qz distinct output nodes; therefore, according to whether u is one of such nodes or not, we know that either $\rho = (q^\ell - 1)/(q-1) + w$ or $\rho = (q^\ell - 1)/(q-1) + w + 1$. By (A.1) we conclude that

$$\left\lfloor \frac{qm}{q^d} \right\rfloor \leq \rho \leq \left\lceil \frac{qm}{q^d} \right\rceil.$$

Note that when m is a power of q, it must be $z = 0$, and therefore $\rho = qm/q^d$ for every output node. □

Let $X = U_{i-1}$ be the set of $(i-1)$-modules, and $Y = U_i$ the set of i-modules. Thus, $q = 3$, $d = d_i$, and $m = 3^{d_{i-1}}$. Each $(i-1)$-module is adjacent to the 3 i-modules that contain its $(i-1)$-blocks, and, accordingly, each i-module u is adjacent to the ρ $(i-1)$-modules, each of which has one of its $(i-1)$-blocks stored in u. For each $(i-1)$-module, we must be able to efficiently determine the i-modules that store its $(i-1)$-blocks and the location of each block within the module.

It is easy to establish a bijection between the $(i-1)$-modules and the pairs $\chi(h, A, B)$ in X so that given an index s, $1 \leq s \leq m$, the pair associated with the sth module is determined in $O(d)$ time. Similarly, a bijection between the i-modules and the d-dimensional vectors over $\mathbb{F}_3$ is easily established. Consider the $(i-1)$-module associated with the pair

$$\chi(h, A, B) = \begin{array}{cccccccc} (a_{d-2}, & \ldots, & a_h, & 0, & a_{h-1}, & \ldots, & a_1, & a_0) \\ (0, & \ldots, & 0, & 1, & b_{h-1}, & \ldots, & b_1, & b_0). \end{array}$$

We adopt the convention that, for $0 \leq j < 3$, the jth $(i-1)$-block of this module is the ℓth item stored in the i-module u, where

$$u = (a_{d-2}, \ldots, a_h, j, a_{h-1} + jb_{h-1}, \ldots, a_1 + jb_1, a_0 + jb_0).$$

and

$$\ell = \frac{3^h - 1}{2} + B.$$

In [PP93] it is proved that the above rule is correct, i.e., no two $(i-1)$-blocks of $(i-1)$-modules are assigned the same location within the same i-module. Moreover, it is not difficult to show that $0 \leq \ell < \rho$.

Observe that the structure of any (U_{i-1}, U_i) is completely determined by the parameter d_i. Since each d_i can be derived from n, we conclude that, in order to represent (U_{i-1}, U_i), a processor needs to know only n. From this parameter, the processor can determine the exact location of any copy of any $(i-1)$-module performing $O(\log n)$ operations (arithmetic or in $\mathbb{F}_3$).

Acknowledgments. This paper benefited from discussions with Matteo Frigo, Tim Harris, and Franco Preparata. The authors wish to thank the anonymous referees for their valuable comments that helped improve both the presentation and the quality of the paper.

REFERENCES

[AHMP87] H. Alt, T. Hagerup, K. Mehlhorn, and F. P. Preparata, *Deterministic simulation of idealized parallel computers on more realistic ones*, SIAM J. Comput., 16 (1987), pp. 808–835.

[BB95] P. Bay and G. Bilardi, *Deterministic on-line routing on area-universal networks*, J. ACM, 42 (1995), pp. 614–640.

[CMS95] A. Czumaj, F. Meyer auf der Heide, and V. Stemann, *Shared memory simulations with triple-logarithmic delay*, in Proceedings of the 3rd European Symposium on Algorithms, Corfu, Greece, 1995, pp. 46–59.

[Hal86] M. Hall Jr., *Combinatorial Theory*, John Wiley & Sons, New York, 1986.

[Her96] K. T. Herley, *Representing shared data on distributed-memory parallel computers*, Math. Systems Theory, 29 (1996), pp. 111–156.

[HB94] K. T. Herley and G. Bilardi, *Deterministic simulations of PRAMs on bounded-degree networks*, SIAM J. Comput., 23 (1994), pp. 276–292.

[HPP95] K. T. Herley, A. Pietracaprina, and G. Pucci, *Implementing shared memory on multi-dimensional meshes and on the fat-tree*, in Proceedings of the 3rd European Symposium on Algorithms, Corfu, Greece, 1995, pp. 60–74.

[Kun93] M. Kunde, *Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound*, in Proceedings of the 1st European Symposium on Algorithms, Bad Hönnef, Germany, 1993, pp. 272–283.

[Lei92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[LMRR94] F. T. Leighton, B. Maggs, A. Ranade, and S. Rao, *Randomized routing and sorting on fixed-connection networks*, J. Algorithms, 17 (1994), pp. 157–205.

[Lei85] C. E. Leiserson, *Fat-trees: universal networks for hardware-efficient supercomputing*, IEEE Trans. Comput., C-34 (1985), pp. 892–901.

[LPP90] F. Luccio, A. Pietracaprina, and G. Pucci, *A new scheme for the deterministic simulation of PRAMs in VLSI*, Algorithmica, 5 (1990), pp. 529–544.

[MV84] K. Mehlhorn and U. Vishkin, *Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories*, Acta Inform., 21 (1984), pp. 339–374.

[PP93] A. Pietracaprina and F. P. Preparata, *An $O(\sqrt{n})$-worst-case-time solution to the granularity problem*, in Proceedings of the 10th Symposium on Theoretical Aspects of Computer Science, Würzburg, Germany, 1993, pp. 110–119.

[PP97] A. Pietracaprina and F. P. Preparata, *Practical constructive schemes for deterministic shared-memory access*, Theory Comput. Syst., 30 (1997), pp. 3–37.

[PP95] A. Pietracaprina and G. Pucci, *Improved deterministic PRAM simulation on the mesh*, in Proceedings of the 22nd International Colloquium on Automata, Languages and Programming, Szeged, Hungary, 1995, pp. 372–383.

[PP97a] A. Pietracaprina and G. Pucci, *The complexity of deterministic PRAM simulation on distributed memory machines*, Theory Comput. Syst., 30 (1997), pp. 231–247.

[PPS94] A. Pietracaprina, G. Pucci, and J. F. Sibeyn, *Constructive deterministic PRAM simulation on a mesh-connected computer*, in Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures, Cape May, NJ, 1994, pp. 248–256.

[Ran91] A. G. Ranade, *How to emulate shared memory*, J. Comput. System Sci., 42 (1991), pp. 307–326.

[SK94] J. F. Sibeyn and M. Kaufmann, *Deterministic 1-k routing on meshes with application to hot-potato worm-hole routing*, in Proceedings of the 11th Symposium on Theoretical Aspects of Computer Science, Caen, France, 1994, pp. 237–248.

[UW87] E. Upfal and A. Wigderson, *How to share memory in a distributed system*, J. ACM, 34 (1987), pp. 116–127.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 649–680

HOW TO MAKE A SQUARE GRID FRAMEWORK WITH CABLES RIGID*

HAROLD N. GABOW† AND TIBOR JORDÁN‡

Abstract. This paper solves the problem of making a bipartite digraph strongly connected by adding the smallest number of new edges that preserve bipartiteness. A result of Baglivo and Graver shows that this corresponds to making a two-dimensional square grid framework with cables rigid by adding the smallest number of new cables. We prove a min-max formula for the smallest number of new edges in the digraph problem and give a corresponding linear-time algorithm. We generalize these results to the problem of making an arbitrary digraph strongly connected by adding the smallest number of new edges, each of which joins vertices in distinct blocks of a given partition of the vertex set.

Key words. graph algorithms, strong connectivity, connectivity augmentation, min-max theorems, rigidity, square grid framework

AMS subject classifications. 05C85, 05C40, 52C25, 90C27, 90C47

PII. S0097539798347189

1. Introduction.

In the *connectivity augmentation problem* we are given a graph and a positive integer k. We wish to find a smallest set of edges whose addition makes the graph k-connected. Here "k-connected" can refer to vertex- or edge-connectivity. Practical applications include the design of reliable networks [9], [16]. Much of the work on connectivity augmentation is discussed in the survey article [7]. The result most relevant to this paper is one of the first augmentation algorithms, due to Eswaran and Tarjan, which makes a digraph strongly connected in linear time [6].

Restricted versions of the augmentation problem may be even more interesting in terms of both theory and practical applications. Many restrictions can be modeled by the minimum cost augmentation problem. However, even special cases of this problem are NP-hard, e.g., making a digraph strongly connected by adding the smallest number of edges from a given set [6]. Other restrictions that have been investigated include the following: augmenting to achieve k-edge-connectivity while preserving simplicity of the given graph [2], [12]; augmenting while preserving planarity [13], [14]; and augmenting an undirected bipartite graph to achieve 2-vertex-connectivity [10] or k-edge-connectivity [3] while preserving bipartiteness. Augmenting a biconnected graph to achieve both k-edge-connectivity and 3-vertex-connectivity is studied in [11].

This paper investigates the problem of making a bipartite digraph strongly connected while preserving bipartiteness. We prove a min-max formula for the smallest possible number of new edges. We give a corresponding linear-time algorithm. In addition we generalize these results to the problem of strong-connectivity augmentation with partition constraints (precisely defined below). We now give some motivation for the problem.

A basic two-dimensional structure in statics is the *square-grid framework* (see Figure 1.1). It consists of horizontal and vertical *rods*. Each rod has the same length,

*Received by the editors November 19, 1998; accepted for publication (in revised form) December 10, 1999; published electronically June 27, 2000.
http://www.siam.org/journals/sicomp/30-2/34718.html
†Department of Computer Science, University of Colorado at Boulder, Boulder, CO 80309-0430 (hal@cs.colorado.edu).
‡BRICS, Department of Computer Science, University of Aarhus, Ny Munkegade, Building 540, DK-8000 Aarhus, Denmark (jordan@daimi.au.dk).

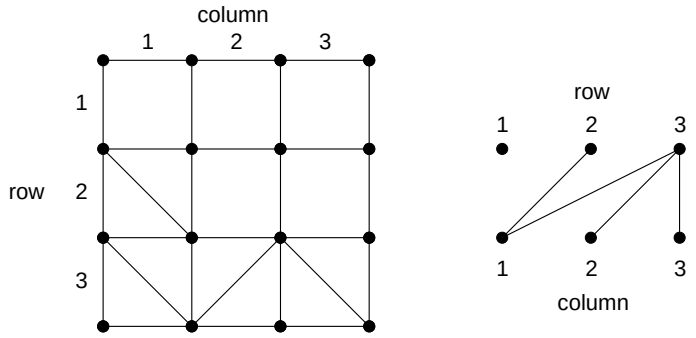


FIG. 1.1. *Row* 1 *can pivot in this square grid framework. But if we add a rod in row* 1*, column* 3*, the framework becomes rigid and the bipartite graph becomes connected.*

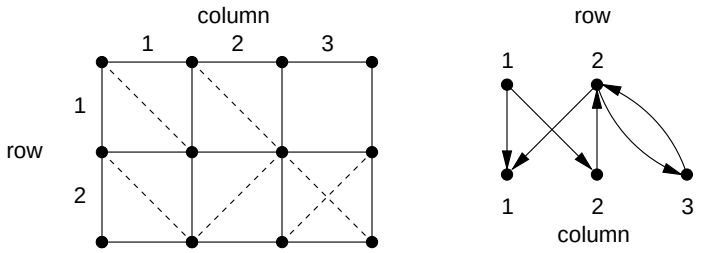


FIG. 1.2. *Here dashed lines represent cables. The joint at the upper left corner can move inwards, compressing the cables in row* 1*, columns* 1 *and* 2*, and row* 2*, column* 1*. But if we rotate the cable in row* 1*, column* 1*, by* 90°*, the framework becomes rigid and the bipartite digraph becomes strongly connected.*

and is rigid. Incident rods are connected by a *joint* which allows the rods to pivot. The rods collectively join all grid-points that are in a rectangular region and are adjacent horizontally or vertically. In addition there may be a number of "diagonal rods" that join two diagonally opposite points of a grid square. Such a framework is *rigid* if it has no nontrivial deformations, i.e., fixing the position of one rod in the plane, the positions of all other rods are uniquely determined. (These concepts are described more precisely in [15].) Bolker and Crapo [4] show that any square-grid framework has a natural representation as a bipartite graph, and the framework is rigid if and only if its bipartite graph is connected (Figure 1.1). Because of this relation the results of [3] solve problems such as, find a smallest set of new rods whose addition results in a framework that is rigid, even after an arbitrary set of k diagonal rods fail.

A potentially cheaper way to make a framework rigid is to use pliable material along the diagonals. A *cable* can be compressed but not stretched and can join two diagonally opposite points of a grid square (see Figure 1.2). Baglivo and Graver [1] show that a framework with cables has a natural representation as a bipartite digraph, such that the framework is rigid if and only if its bipartite digraph is strongly connected (Figure 1.2). The *cable-framework rigidity problem* is to make a given framework with diagonal cables rigid by adding the smallest number of new cables. Our solution to the graph augmentation problem solves the cable-framework rigidity problem, by the theorem of Baglivo and Graver.

The main part of our solution to the problem of bipartite strong-connectivity augmentation is a min-max formula for the optimum number of new edges (Lemma 5.3).

This formula extends to the general problem of strong-connectivity augmentation with partition constraints (Theorem 7.8). Roughly put, the formula states that the optimum number of new edges either equals a natural lower bound or is one more than that. The latter holds exactly when the graph belongs to one of four infinite families.

This result has similar overall structure to the min-max formula of [3] for bipartite undirected graph k-edge-connectivity augmentation: Both formulas have an infinite number of exceptional graphs that require one extra edge. The formula of [3] has a single family of exceptional graphs. Our problem, just for the $k = 1$ case, seems to have more complicated structure, with four distinct families. Together these two results give evidence that min-max formulas with exceptions may be a general phenomenon.

Our result corresponds to the $k = 1$ case of [3]. This case is the problem of augmenting a bipartite graph to make it connected and bipartite. This simple problem has no exceptional families. It is equivalent to the rod-framework rigidity problem, which was solved in [1] and [15]. Until now the cable-framework rigidity problem has been left open, perhaps because the phenomenon of exceptional families has been ignored.

Section 2 of this paper gives basic facts about our augmentation problem, including lower and upper bounds on the solution size. Sections 3–5 prove the min-max formula for bipartite augmentation. Sections 6 and 7 prove the formula in general (reducing to the case covered in section 3). Section 8 presents efficient algorithms for the augmentation problems. Section 9 discusses some augmentation problems related to ours.

The material in section 5 is not logically necessary for proving the general formula. However, it gives a simpler proof of the special case needed for the cable-framework rigidity problem. Furthermore this proof is the basis of the algorithm in section 8 that is more efficient than the general algorithm. The reader should also be aware at the outset that many details of the algorithms in section 8 are omitted because they directly follow the proofs in preceding sections.

This section closes with some terminology. In the problem of *(strong-connectivity) augmentation with partition constraints* we are given a digraph $D = (V, E)$ together with a partition $\mathcal{P}$ of V into disjoint sets called *color classes*. We wish to make D strongly connected, if possible, by adding the smallest number of new directed edges, where no new edge joins vertices in the same color class. The *bipartite strong-connectivity augmentation problem* is the special case where D is a bipartite digraph and the color classes are the two sets of the given bipartition of V.

We use B, W ("black," "white") as typical colors. A vertex v is *colored* B if $v \in B \in \mathcal{P}$. A *singleton vertex* v has $\{v\} \in \mathcal{P}$, i.e., no other vertex has its color. There is no restriction on a new edge incident with a singleton vertex. Note that in our general augmentation problem there is no restriction on an original edge of E: it can join vertices of the same color. We will refer to the augmentation problem for a graph D without mentioning $\mathcal{P}$ when $\mathcal{P}$ is clear from context.

The symbols $\subseteq$ and $\subset$ denote set containment and proper set containment, respectively. An over-bar denotes set complement (with the universe understood), e.g., $\overline{B}$ denotes the set of nonblack vertices $V - B$. A set consisting of a single element x is usually denoted by x.

Consider a digraph. If S is a set of vertices containing y but not x, then an edge (x, y) *enters* S and an edge (y, x) *leaves* S. A vertex x *precedes* a vertex y (alternatively, x *can reach* y) if there is a path from x to y.

A digraph with no cycles is a *dag* (directed acyclic graph). A *source* (*sink*) is

a vertex with no entering (leaving) edge. An *extreme* vertex is a source or a sink. The *strong component graph* of a digraph is formed by contracting every strongly connected component. It is a dag.

2. Basic facts. This section presents basic concepts and notation. In particular it defines the quantity Φ and proves that an optimum solution to our augmentation problem uses either Φ or $\Phi+1$ new edges. It also presents several graph families that require exactly Φ new edges. These graph families provide the foundation of our general solution.

The special case of strong-connectivity augmentation with partition constraints when D is a dag is called the *dag problem*. The general problem reduces to the dag problem, as follows. Let D be a given digraph. Let H be its strong component graph. To assign colors to the vertices of H let a vertex x of H correspond to the strong component X of D. If every vertex of X is colored B, then x is colored B. If X contains vertices of two or more colors, then x is a singleton vertex. Solutions to the augmentation problem with partition constraints on D and H correspond because of the following observation: For two vertices x_i, $i=1,2$, of H corresponding to strong components X_i, $i=1,2$, of D, x_1 and x_2 have different colors if and only if some vertex of X_1 and some vertex of X_2 have different colors.

From now on we will work on the dag problem for a dag $H=(V,E)$. We use the following notation. OPT is the smallest number of new edges needed to solve the given dag problem. By definition $OPT=0$ if $|V|=1$ and $OPT=\infty$ (i.e., there is no solution) if $|V|>1$ and all vertices have the same color.

Let X be a set of vertices of H. X^+ denotes the set of all sources in X and X^- the set of all sinks. For instance V^+ is the set of all sources of H. An isolated vertex belongs to $V^+\cap V^-$.

$R^-(X)$ denotes the set of all sinks that can be reached from some source of X. Similarly $R^+(X)$ denotes the set of all sources that can reach some sink of X. For $X,Y\subseteq V$, the predicate $r(X,Y)$ is true when some source of X precedes some sink of Y. This is equivalent to $R^-(X)\cap Y\neq\emptyset$. Note as a special case that if x is a source and y is a sink, $r(x,y)$ means x precedes y. We abbreviate $r(X,X)$ to $r(X)$. Negation is denoted by $\neg$, e.g., $\neg r(X,Y)$.

We will characterize OPT in terms of the following quantity:

$$\Phi=\begin{cases}0 & \text{if } |V|=1,\\ \infty & \text{if } |V|>1 \text{ and } |\mathcal{P}|=1,\\ \max\{|V^+|,\,|V^-|,\,|B^+|+|B^-| : B\in\mathcal{P}\} & \text{otherwise.}\end{cases}$$

Clearly $OPT=\Phi$ in the first two cases. In the third case $OPT\geq\Phi$: A solution must contain an edge entering every source and an edge leaving every sink. This implies $OPT\geq|V^+|,|V^-|$. One new edge cannot leave a sink and enter a source of the same color. This implies $OPT\geq|B^+|+|B^-|$ for any color B.

There are simple graphs with $OPT>\Phi$. For instance, the graph on four vertices $b^+\in B^+$, $b^-\in B^-$, $w^+\in W^+$, $w^-\in W^-$ with edges (b^+,w^-) and (w^+,b^-) has $\Phi=2$ and $OPT=3$.

Any instance of the dag problem has a *directionally symmetric* instance, obtained by reversing the orientations of all edges. Solutions to these two problems correspond, and OPT and Φ are the same. We invoke directional symmetry to simplify the proofs.

It is convenient to work with graphs that have no isolated vertices. If $|V|>1$ we eliminate an isolated vertex x by replacing it with vertices x_1,x_2 having the same color as x and an edge (x_1,x_2). (There is no edge (x_2,x_1), so the transformed graph

is a dag.) A solution on the transformed graph gives a solution on the original graph by contracting x_1, x_2 to a vertex we call x.

PROPOSITION 2.1. *The transformation to eliminate an isolated vertex does not change the value of* $|V^+|$, $|V^-|$, $|B^+| + |B^-|$ *for* $B \in \mathcal{P}$, Φ *or* OPT.

Proof. We show OPT does not change. Let H be the original graph and H' the transformed graph. Clearly a solution on H' gives a solution on H that is no larger. For the converse let F be a set of new edges in H and let $F' = \{(y, x_1) : (y, x) \in F\} \cup \{(x_2, z) : (x, z) \in F\}$. A path in $H + F$ corresponds to a path in $H' + F'$ if the vertex x is replaced by x_1, x_2. This implies $H' + F'$ is strongly connected if $H + F$ is. □

From now on assume the given dag H has no isolated vertices. We drop this assumption in the statement of our main results (Lemma 5.3 and Theorem 7.8).

In a *unicolored dag* the extreme vertices all have the same color. In a *bicolored (tricolored) dag* the extreme vertices are contained in at most two (three) color classes, which we take as B and W (and G).

LEMMA 2.2. *A unicolored dag has* $OPT = \Phi$.

Proof. Assume $|\mathcal{P}| > 1$, else, obviously, $OPT = \Phi = \infty$. Thus $\Phi = |V^+| + |V^-|$. Choose any vertex y of a color different from the extreme vertices. Add edges (y, x^+) for every $x^+ \in V^+$ and (x^-, y) for every $x^- \in V^-$. Thus y can reach every source and every sink can reach y. This implies y can reach every vertex and every vertex can reach y. □

A *rooted dag* is a dag with exactly one source or exactly one sink.

LEMMA 2.3. *Any rooted dag has* $OPT = \Phi$.

Proof. By symmetry assume the given dag H has exactly one source x^+. Assume H is not unicolored (Lemma 2.2). Hence $\Phi = |V^-|$ and some sink y^- is differently colored from x^+. Add edge (y^-, x^+). The new strong component graph K has a unique source that is a singleton vertex. Also Φ has decreased by 1. Now add an edge from each sink of K to the source to get a strongly connected graph. This shows $OPT = \Phi$. □

A set of extreme vertices is *consistently colored* unless it contains a source and sink of the same color. The dag is *consistently colored* if $V^+ \cup V^-$ is consistently colored. (Equivalently, every color B has B^+ or B^- empty.)

DEFINITION 2.4. *A pair of vertices* $x^- \in V^-$ *and* $y^+ \in V^+$ *is* compatible *if the pair is consistently colored and there are vertices* $x^+ \in V^+ - y^+$, $y^- \in V^- - x^-$ *with* $r(x^+, x^-)$ *and* $r(y^+, y^-)$.

LEMMA 2.5. *Any consistently colored set of two sources and two sinks contains a compatible pair.*

Proof. Let the consistently colored set consist of sources y_i^+, $i = 1, 2$, and sinks x_i^-, $i = 1, 2$. For $i = 1, 2$, if x_i^-, y_i^+ is not compatible, then $r(y_i^+, x_i^-)$. This makes x_1^-, y_2^+ compatible. □

If vertices $x^- \in V^-$, $y^+ \in V^+$ form a compatible pair, *augmenting* x^-, y^+ means adding a new edge (x^-, y^+) to the graph and passing to the new strong component graph. Thus if $H + (x^-, y^+)$ has a strong component containing x^- and y^+, we contract it. The augment operation also adds the new edge (x^-, y^+) to the solution set of the augmentation problem.

LEMMA 2.6. *Augmenting a compatible pair* x^-, y^+ *decreases* $|V^+|$, $|V^-|$, *and* $|B^+| + |B^-|$ *for* B *the color of* x^- *or* y^+, *each by one. Furthermore it does not create an isolated vertex.*

Proof. Let H be the given graph. Let K be the strong component graph of

$H + (x^-, y^+)$. Any vertex of $V^+ - y^+$ is a source of K since it has no entering edge. Any other vertex of H is not a source of K since it is preceded by a vertex of $V^+ - y^+$. (In particular y^+ is preceded by x^+ of Definition 2.4.) This proves $|V^+|$ decreases by one. The other parts are similar. □

LEMMA 2.7. *Any consistently colored dag has $OPT = \Phi$.*

Proof. A consistently colored dag has $\Phi = \max\{|V^+|, |V^-|\}$. Without loss of generality let $\Phi = |V^-|$. If the given graph H is rooted, we are done by Lemma 2.3. If not, H has a compatible pair by Lemma 2.5. Augment it. Lemma 2.6 shows the graph remains consistently colored and Φ decreases by 1. Repeating this procedure eventually makes H rooted. □

LEMMA 2.8. *Let H be a dag with no compatible pair of vertices. Then H is either rooted, unicolored, or bicolored. In the third alternative if H is bicolored by B and W, then H contains two subgraphs that are rooted dags, one spanning $B^+ \cup W^-$, the other spanning $W^+ \cup B^-$, and $B^+, B^-, W^+, W^- \neq \emptyset$.*

Proof. Suppose H is neither rooted nor unicolored. We show the third alternative of the lemma holds. If H is consistently colored, then Lemma 2.5 gives a compatible pair. Hence assume some color B has $B^+, B^- \neq \emptyset$.

Claim. $\overline{B}^- \neq \emptyset$ implies H contains a rooted dag spanning $B^+ \cup \overline{B}^-$ and $\overline{B}^+ \neq \emptyset$.

Proof. The hypothesis and Lemma 2.5 imply at least one of the sets B^+, $\overline{B}^-$ has exactly one vertex. First suppose $B^+ = \{b^+\}$. Since H is not rooted, $\overline{B}^+ \neq \emptyset$. Since any sink $\bar{b}^- \in \overline{B}^-$ is incompatible with b^+, $r(b^+, \bar{b}^-)$.

Next suppose $\overline{B}^- = \{\bar{b}^-\}$ and $|B^+| > 1$. Since any source b^+ is incompatible with $\bar{b}^-$, $r(b^+, \bar{b}^-)$. Furthermore b^+ does not precede any other sink, which with $B^- \neq \emptyset$ implies $\overline{B}^+ \neq \emptyset$. This proves the claim. □

Since H is not unicolored, $\overline{B}^+ \cup \overline{B}^- \neq \emptyset$. The claim and its directionally symmetric version show H contains two rooted dags, spanning $B^+ \cup \overline{B}^-$ and $\overline{B}^+ \cup B^-$, respectively. If two vertices of $\overline{B}^+$ and $\overline{B}^-$ have different colors, they are compatible. Hence all extremes of $\overline{B}$ have the same color. □

LEMMA 2.9. *Any dag has $OPT \leq \Phi + 1$.*

Proof. First assume the given dag H is bicolored. If H is unicolored or rooted, Lemmas 2.2 and 2.3 show $OPT = \Phi$. If H has a compatible pair, augment it. Lemma 2.6 shows this reduces Φ by one, so by induction we are done. Now Lemma 2.8 shows we can assume H contains two subgraphs that are rooted dags, one spanning $B^+ \cup W^-$ and the other spanning $W^+ \cup B^-$. We consider two cases that, using directional symmetry, cover all possibilities.

Case 1. $|W^+| = |W^-| = 1$.

Lemma 2.8 shows $B^+, B^- \neq \emptyset$. Hence $\Phi = |B^+| + |B^-|$. Add an edge from each black sink to the unique white sink. The resulting graph has a unique sink, $1 + |B^+|$ sources, and $\Phi = 1 + |B^+|$. Hence Lemma 2.3 implies H has $OPT \leq \Phi + 1$.

Case 2. $|W^+| = |B^+| = 1$ and $|W^-|, |B^-| > 1$.

We show $OPT = \Phi$ in this case. Observe that $\Phi = |V^-|$. Take distinct vertices $w^+ \in W^+$, $b^+ \in B^+$, $w_1^-, w_2^- \in W^-$, $b_1^-, b_2^- \in B^-$. Add edges (w_1^-, b_1^-), (b_1^-, w^+) and let K be the new strong component graph. The two new edges decrease $|V^-|$ by two (since w^+ precedes the sink b_2^-). K is a rooted dag with unique source b^+. Since K has a white sink w_2^-, K has $\Phi = |V^-|$. Lemma 2.3 implies $OPT = \Phi$ for both K and H.

Now we show that a dag H that is not bicolored reduces to the bicolored case. First suppose $\Phi > |B^+| + |B^-|$ for every $B \in \mathcal{P}$. Lemma 2.8 shows a compatible pair exists, so augment it. Lemma 2.6 and the assumption on Φ show this decreases Φ by

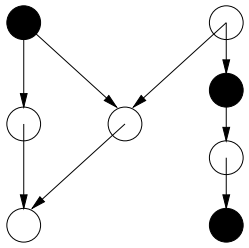


FIG. 3.1. *A* 1*-blocker for* B*. Black and white vertices are in* B *and* W*, respectively.*

1. Repeat this as many times as possible until the graph becomes bicolored, rooted, or $\Phi = |B^+| + |B^-|$ for some $B \in \mathcal{P}$. It remains only to analyze this last case.

Suppose $\Phi = |B^+| + |B^-|$. Let K be the same dag using just two colors, B and $\overline{B}$. This does not change Φ, i.e., $\Phi \geq |\overline{B}^+| + |\overline{B}^-|$, since $2\Phi \geq |V^+| + |V^-| = |B^+|+|B^-|+|\overline{B}^+|+|\overline{B}^-| = \Phi+|\overline{B}^+|+|\overline{B}^-|$. Since K is bicolored it has $OPT \leq \Phi+1$. Any solution to the augmentation problem on K is a valid solution on H. Hence $OPT \leq \Phi + 1$ for H. □

COROLLARY 2.10. *Any bicolored dag with* $\Phi > |B^+| + |B^-|$ *for every* $B \in \mathcal{P}$ *has* $OPT = \Phi$*.*

Proof. Apply the above argument. An augment decreases all four quantities that define Φ, by Lemma 2.6. Hence it preserves the inequality of the lemma, and the given dag eventually becomes rooted or satisfies Case 2 (or its directionally symmetric version). □

This result is generalized by Lemma 3.5.

3. Bicolored dags. Our solution to the augmentation problem works by reducing the graph to one of the simple families analyzed in section 2 or to a bicolored dag. This section gives the key part of the argument for bicolored dags. We have already seen that for such graphs $OPT = \Phi$ unless Φ is the number of extreme vertices in some color (Corollary 2.10). Hence we will complete the analysis of bicolored dags by considering two cases,

$$\Phi = |B^+| + |B^-| > |W^+| + |W^-|;$$

$$\Phi = |B^+| + |B^-| = |W^+| + |W^-|.$$

In both cases it is possible to have $OPT = \Phi + 1$. This occurs precisely when the graph is what we call a 1-blocker. This section proves that a bicolored graph that is not a 1-blocker has $OPT = \Phi$ (Lemma 3.4). Blockers are discussed in section 4.

The following definition of 1-blocker applies to graphs in general, not necessarily bicolored. Recall the function R^- introduced at the start of section 2.

DEFINITION 3.1. *A dag is a* 1-blocker *if for some color* B, (i) $\Phi = |B^+| + |B^-|$; (ii) $B^+, \overline{B}^+ \neq \emptyset$; (iii) $R^-(B) \subseteq \overline{B}$.

Figure 3.1 illustrates a 1-blocker. In a bicolored graph we can replace $\overline{B}$ by W in Definition 3.1. As mentioned a 1-blocker always has $OPT > \Phi$. This fact is not needed in this section and is proved in Lemma 4.3.

LEMMA 3.2. *A bicolored dag with* $\Phi = |B^+|+|B^-| > |W^+|+|W^-|$ *has* $OPT = \Phi$ *unless the dag is a* 1*-blocker.*

Proof. If $B^+ = \emptyset$, then $|B^-| = \Phi \geq |V^-|$ implies $V^- \subseteq B$. Hence the graph H is consistently colored and Lemma 2.7 shows $OPT = \Phi$. So by directional symmetry assume $B^+, B^- \neq \emptyset$.

Assume H is not a 1-blocker. This implies $r(B)$. In proof, we can assume $W^+ \neq \emptyset$, since otherwise $V^+ \subseteq B$ and $B^- \neq \emptyset$ imply $r(B)$. Now since H is not a 1-blocker condition (iii) fails, which means $r(B)$.

Choose two vertices $b^+ \in B^+$ and $b^- \in B^-$ with $r(b^+, b^-)$ and call them *witnesses*. Say that a pair of vertices x^-, y^+ is *witness-compatible* if it is compatible and in addition, for x^+ and y^- as in Definition 2.4, $x^- = b^-$ implies $y^- \in B$, and $y^+ = b^+$ implies $x^+ \in B$.

Assume $V^+ - b^+$, $V^- - b^- \neq \emptyset$, since otherwise H is rooted and Lemma 2.3 shows we are done. These two sets together contain a white extreme vertex, since otherwise all extremes are black and Lemma 2.2 shows we are done. The two sets contain a black extreme vertex, by the lemma's hypothesis on Φ. Hence without loss of generality assume there are vertices $y^+ \in W^+$ and $y^- \in B^- - b^-$. Observe that there exists a witness-compatible pair: If $r(y^+, y^-)$, then b^-, y^+ is witness-compatible. If $\neg r(y^+, y^-)$, then y^-, y^+ is witness-compatible.

Call the witness-compatible pair x^-, y^+ and augment it. Lemma 2.6 shows the augment decreases Φ by 1 and the resulting dag K satisfies the hypothesis of the lemma on Φ. K is not a 1-blocker because (iii) fails, i.e., K has two witnesses. This is clear if neither b^+ nor b^- is involved in the augment. If $x^- = b^-$, then K has source b^+ and sink y^-, both black with $r(b^+, y^-)$. Similarly if $y^+ = b^+$, then $r(x^+, b^-)$.

Now the lemma follows by applying induction to K. □

LEMMA 3.3. *A bicolored dag with* $\Phi = |B^+| + |B^-| = |W^+| + |W^-|$ *has* $OPT = \Phi$ *unless the dag is a* 1*-blocker.*

Proof. First observe that $\Phi = |V^+| = |V^-|$. In proof, the hypothesis on Φ implies $2\Phi \geq |V^+| + |V^-| = (|B^+| + |B^-|) + (|W^+| + |W^-|) = 2\Phi$. Thus equality holds throughout and $\Phi = |V^+| = |V^-|$.

If $B^+ = \emptyset$, then $\Phi = |V^-| = |B^-|$ so $V^- \subseteq B$. The graph H is consistently colored and Lemma 2.7 shows $OPT = \Phi$. Hence by symmetry assume $B^+, W^+ \neq \emptyset$. Also assume H is not a 1-blocker, i.e., $r(B)$ and $r(W)$.

Choose as witnesses four vertices $b^+ \in B^+$, $b^- \in B^-$, $w^+ \in W^+$, $w^- \in W^-$ where $r(b^+, b^-)$ and $r(w^+, w^-)$. Say that a pair of vertices x^-, y^+ is *witness-compatible* if it is compatible and in addition, for x^+ and y^- as in Definition 2.4, y^- has the same color as x^- if x^- is a witness, and x^+ has the same color as y^+ if y^+ is a witness.

If the four witnesses are not the only extreme vertices, then H has a witness-compatible pair. In proof, without loss of generality assume there is another source $y^+ \in W^+ - w^+$. Since $\Phi = |V^+| = |V^-| = |B^+| + |B^-|$ there is a sink $y^- \in B^- - b^-$. If $r(y^+, y^-)$, then b^-, y^+ is witness-compatible. If $\neg r(y^+, y^-)$, then y^-, y^+ is witness-compatible.

Augment the witness-compatible pair. This decreases Φ by 1 and the resulting dag satisfies the hypothesis of the lemma and is not a 1-blocker. This is proved by the same argument as Lemma 3.2 (even when x^- and y^+ are both witnesses).

Now we can assume by induction that the four witnesses are the only extremes. Augment b^-, w^+ to reduce Φ from 2 to 1. Lemma 2.3 completes the argument. □

Corollary 2.10 and Lemmas 3.2 and 3.3 give the following.

LEMMA 3.4. *A bicolored dag has* $OPT = \Phi$ *unless it is a* 1*-blocker.* □

We extend part of this analysis to arbitrarily colored dags as follows.

LEMMA 3.5. *An arbitrarily colored dag* H *with* $\Phi = |V^-| > |V^+|, |B^+| + |B^-|$ *for every* $B \in \mathcal{P}$ *has* $OPT = \Phi$.

Proof. Assume H is not rooted or bicolored by Lemma 2.3 and Corollary 2.10. Lemma 2.8 shows there is a compatible pair. We will show how to augment a com-

patible pair so that Φ decreases by one, and the new graph either has $OPT = \Phi$ or continues to satisfy the inequality of the lemma. By induction this completes the proof. Call a color B *big* if $|B^+| + |B^-| = \Phi - 1$.

Suppose a given compatible pair contains a vertex from every big color. Augment it. This decreases $|V^-|$ and $|V^+|$ each by 1, so Φ decreases by 1. Furthermore the inequality of the lemma is preserved. Hence we can assume that no compatible pair contains a vertex from every big color.

Suppose there are two (or more) big colors, say, B and W. Since H has fewer than 2Φ extreme vertices and is not bicolored, exactly one extreme vertex g is in $B \cup W$. By symmetry let g be a source. Without loss of generality g precedes a black sink. Change the color of g to black. This does not change Φ. The resulting bicolored graph is not a 1-blocker since $r(B)$ and W has fewer than Φ extremes. Hence Lemma 3.4 gives the desired conclusion.

We can now assume there is a unique big color B and no compatible pair contains a vertex of B. Take a compatible pair x^-, y^+. Every vertex of B^+ is incompatible with x^- and so precedes x^-. Similarly every vertex of B^- is preceded by y^+. B^+ and B^- are both nonempty. (This follows since $\Phi = |V^-| > |V^+| \geq 2$ implies that B has at least two extremes. If all these extremes are sources (sinks), then some compatible pair involves B by Lemma 2.5.) Now augment the pair x^-, y^+. This decreases Φ by 1 and makes $\Phi = |B^+| + |B^-|$. Also the new graph K has $r(B)$. Take B and $\overline{B}$ as the two colors of K. $\overline{B}$ has less than Φ extreme vertices, since H has $2\Phi > |V^+| + |V^-|$. We conclude that K is bicolored and is not a 1-blocker. A solution using $OPT = \Phi$ new edges on K gives a similar solution on H. □

To summarize the argument up to this point, we have shown how to augment a bicolored dag that is not a 1-blocker by adding Φ new edges. This does not solve even the bipartite strong-connectivity augmentation problem, since our reduction to the dag problem can create new (singleton) colors. We solve the augmentation problem by following the strategy of Lemma 3.5, i.e., we attempt to recolor vertices to get a bicolored dag with the same value of Φ that is not a 1-blocker. Successful recoloring can be blocked by 1-blockers as well as several other families, all of which we call blockers. The next section studies blockers. Then section 5 executes our recoloring strategy to solve the bipartite problem, and sections 6 and 7 do the same for the general problem.

4. Blockers. This section defines blockers and shows they have $OPT > \Phi$ (Lemma 4.3). It also completes the solution of our augmentation problem in another major case (Lemma 4.5).

We define blockers for arbitrarily colored dags. The four types of blockers are illustrated in Figure 4.1. For convenience we repeat Definition 3.1 for 1-blockers here.

DEFINITION 4.1. *A dag is a* blocker *if it is a* 1-, 2-, 3-, *or* 4-*blocker as defined by the conditions below. In these conditions B and W denote two distinct colors.*

A 1-blocker *has*

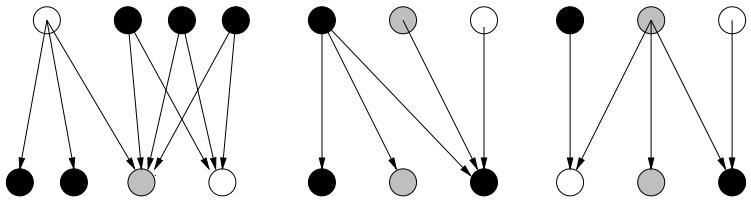
(i) $\Phi = |B^+| + |B^-|$;
(ii) $B^+, \overline{B}^+ \neq \emptyset$;
(iii) $R^-(B) \subseteq \overline{B}$.

The remaining blockers have the same first condition:

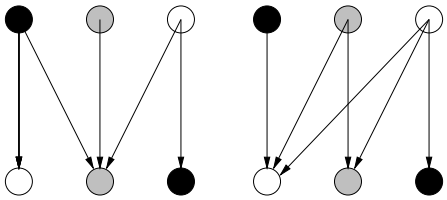
(i) $\Phi = |V^+| = |V^-|$.

A 2-blocker *has*

(ii) $B^+, \overline{B}^+ \neq \emptyset$;
(iii) $R^-(\overline{B}) \subseteq B$.



(a) 1-blocker for B. (b) 2-blocker for B. (c) 3^+-blocker for B, W.



(d) 3^--blocker for B, W. (e) 4-blocker for B, W.

FIG. 4.1. *Simple examples of blockers. Black, white, and shaded vertices are in B, W, and $V - B - W$, respectively.*

A 3-blocker *is a* 3^+*-blocker or a* 3^-*-blocker, where a* 3^+-blocker *has*

(ii) $B^+, W^+ \neq \emptyset$;

(iii) $R^-(B) \subseteq W$, $R^-(W) \subseteq B$, $|V^+ - B - W| = 1$;

and a 3^--blocker *satisfies the reverse conditions*

(ii) $B^-, W^- \neq \emptyset$;

(iii) $R^+(B) \subseteq W$, $R^+(W) \subseteq B$, $|V^- - B - W| = 1$.

A 4-blocker *has*

(ii) $B^+, B^- \neq \emptyset$;

(iii) $R^-(B) \cup R^+(B) \subseteq W$, $|B^+| + |B^-| = \Phi - 1$.

The five varieties of blockers (differentiating between 3^+ and 3^-) are distinct. For instance, each blocker in Figure 4.1 satisfies only one set of blocker conditions.

The *reachability conditions* of a blocker are the containment relations in condition (iii). We specify the color classes associated with a blocker by saying a 1- *or* 2-*blocker for* B, or a 3- *or* 4-*blocker for* B, W. A *blocker for* B *(*B, W*)* is a 1- or 2-blocker for B, or a 3- or 4-blocker for B, W. The roles of B and W are asymmetric in a 4-blocker for B, W. We say a 4-*blocker for* B if we do not wish to refer to the second color W. We say a 3^+- *(*3^-*-) blocker for* z if z is the (unique) vertex of $V^+ - B - W$ ($V^- - B - W$).

LEMMA 4.2. *The reverse graph of a blocker is a blocker of the same type* $(1, 2, 3,$ *or* $4)$*. Furthermore the reverse graph of a* 3^+*-blocker (*3^-*-blocker) is a* 3^-*-blocker (*3^+*-blocker).*

Proof. The lemma is obvious except for 1- and 2-blockers. For these we use two simple properties of the reachability functions: Consider any two sets of vertices X and Y in H. The set $R^+(X)$ ($R^-(X)$) in H is equal to the set $R^-(X)$ ($R^+(X)$) in the reverse graph. In H, $R^-(X) \subseteq Y$ if and only if $R^+(\overline{Y}) \subseteq \overline{X}$. Now we consider the two types of blockers.

1-*blocker.* The reachability condition $R^-(B) \subseteq \overline{B}$ is equivalent to $R^+(B) \subseteq \overline{B}$,

which in the reverse graph is the reachability condition $R^-(B) \subseteq \overline{B}$. Next we verify condition (ii) in the reverse graph. Conditions (ii) and (iii) in H imply $\overline{B}^- \neq \emptyset$. Conditions (i) and (ii) imply $|B^+| + |B^-| = \Phi \geq |V^+| > |B^+|$, so $B^- \neq \emptyset$.

2-*blocker.* The argument is similar: The reachability condition $R^-(\overline{B}) \subseteq B$ is equivalent to $R^+(\overline{B}) \subseteq B$, which in the reverse graph is the reachability condition $R^-(\overline{B}) \subseteq B$. For condition (ii) in the reverse graph, conditions (ii) and (iii) in H imply $B^- \neq \emptyset$. Conditions (i) and (ii) imply $|V^-| = \Phi \geq |B^+| + |B^-| > |B^-|$, so $\overline{B}^- \neq \emptyset$. □

LEMMA 4.3. *Any blocker has* $OPT > \Phi$.

Proof. We consider each of the four types of blockers in turn.

1-*blocker.* Let X be the set of all vertices preceded by a vertex of B^+. X is a nonempty proper subset of V, since $B^+ \subseteq X$ and $X \cap \overline{B}^+ = \emptyset$. Hence some new edge e leaves X. If $OPT = \Phi = |B^+| + |B^-|$, then every new edge either leaves a black sink or enters a black source, so e either leaves a black sink of X or enters a black source of $\overline{X}$. Such a sink does not exist since $R^-(B) \subseteq \overline{B}$, and such a source does not exist since $B^+ \subseteq X$. We conclude $OPT > \Phi$.

We give a similar argument for each of the remaining blockers. They all have $\Phi = |V^+| = |V^-|$. We will assume that $OPT = \Phi$. This implies every new edge must leave a sink and enter a source. For each blocker we will derive a contradiction.

2-*blocker.* Let X be the set of all vertices preceded by a vertex of $\overline{B}^+$. X is a nonempty proper subset of V, since $\overline{B}^+ \subseteq X$ and $X \cap B^+ = \emptyset$. Hence some new edge e leaves X, i.e., it leaves a sink of X and enters a source of $\overline{X}$. Every sink of X is black since $R^-(\overline{B}) \subseteq B$, and every source of $\overline{X}$ is black since $\overline{B}^+ \subseteq X$. Hence e joins two black vertices. This contradiction shows $OPT > \Phi$.

3-*blocker.* Let H be a 3^+-blocker for B, W and vertex z. Let X be the set of all vertices preceded by a vertex of B^+. X is a nonempty proper subset of V, since $B^+ \subseteq X$ and $z \notin X$. Hence some new edge e leaves X, i.e., it leaves a sink of X and enters a source of $\overline{X}$. The sink of X is white since $R^-(B) \subseteq W$. This implies the source of $\overline{X}$ is z, since $B^+ \subseteq X$. We conclude e is directed from a white sink to z. A similar argument shows that a new edge e' is directed from a black sink to z. Clearly $e \neq e'$. This gives the desired contradiction, since $OPT = \Phi$ implies at most one new edge is directed to z.

If H is a 3^--blocker, the lemma follows from the previous argument with Lemma 4.2.

4-*blocker.* Let X be the set of all vertices preceded by a vertex of B^+. X is a nonempty proper subset of V, since $B^+ \subseteq X$ and $X \cap B^- = \emptyset$. (The latter holds since $R^-(B) \subseteq W$.) Hence a new edge e leaves X. The condition $R^-(B) \subseteq W$ implies e goes from a white sink to a nonblack source.

The same argument on the reverse graph shows a new edge e' goes from a nonblack sink to a white source. A new edge cannot join two white vertices so $e' \neq e$. Thus we have two new edges, neither incident to a black vertex. But $|B^+| + |B^-| = \Phi - 1 = OPT - 1$ implies there is at most one such new edge. □

The next lemma is needed only for the general case of our theorem in sections 6 and 7. Call a dag a *proper* i*-blocker* if it is an i-blocker and in addition, for i equal to 3 or 4, it is not also a 1- or 2-blocker. Note that Lemma 4.2 implies the reverse of a proper i-blocker is a proper i-blocker.

LEMMA 4.4. *A* 3- *or* 4-*blocker for* B, W *has* $r(\overline{B \cup W})$ *if it is proper.*

Proof. Consider a proper 3-blocker H. Lemma 4.2 implies we can assume H is a 3^+-blocker. We claim $\overline{B \cup W}$ contains a sink. This claim establishes the lemma

for 3^+-blockers, since any source that precedes such a sink must be in $\overline{B \cup W}$ by the reachability conditions of a 3^+-blocker.

To prove the claim note that $\Phi > |B^+| + |B^-|, |W^+| + |W^-|$ since H is not a 1-blocker. Since a 3^+-blocker has 2Φ extremes, this implies at least two extremes are not in $B \cup W$. The definition of 3^+-blocker shows at least one sink is not in $B \cup W$.

Suppose H is a proper 4-blocker. Since H is not a 2-blocker and $B^+, W^+ \neq \emptyset$, $R^-(\overline{W}) \not\subseteq W$. So let x be a source of $\overline{W}$ that precedes a sink y of $\overline{W}$. The reachability conditions of a 4-blocker show $x, y \notin B$. Thus x and y are the desired vertices. □

We conclude this section by analyzing another type of arbitrarily colored dag.

LEMMA 4.5. *An arbitrarily colored dag H with $\Phi = |B^+| + |B^-|$ for some $B \in \mathcal{P}$ has $OPT = \Phi$ unless H is a* 1- *or* 2*-blocker.*

Proof. Let K be the given dag H with color classes taken to be B and $\overline{B}$. As shown at the end of Lemma 2.9, K has the same value of Φ as H and a solution on K is valid on H. Lemma 3.4 shows K has $OPT = \Phi$ unless K is a 1-blocker. If K is a 1-blocker for B, then so is H. Suppose K is a 1-blocker for $\overline{B}$. Thus $\Phi = |\overline{B}^+| + |\overline{B}^-|$. Hence $|V^+| + |V^-| = |B^+| + |B^-| + |\overline{B}^+| + |\overline{B}^-| = 2\Phi \geq |V^+| + |V^-|$, where we have used $\Phi \geq |V^+|,\ |V^-|$. We conclude equality holds, i.e., $\Phi = |V^+| = |V^-|$. This plus $R^-(\overline{B}) \subseteq B$ shows H is a 2-blocker for B. □

Lemma 4.5, Lemma 3.5, and symmetry leave only one case to complete the analysis of arbitrarily colored dags,

$$\Phi = |V^-| = |V^+| > |B^+| + |B^-| \text{ for every } B \in \mathcal{P}.$$

The next section analyzes this last case for the dags that arise in bipartite strong-connectivity augmentation. Sections 6 and 7 do the same for general augmentation.

5. Bipartite augmentation theorem. Consider the bipartite strong-connectivity augmentation problem. The reduction of section 2 transforms a given bipartite digraph into a dag H whose nonsingleton vertices have only two colors. We call such a dag *core-bicolored*. This section gives the analysis of the last case mentioned above for core-bicolored dags (Lemma 5.2). This completes the solution to the bipartite strong connectivity augmentation problem and the cable-framework rigidity problem. A dag is *core-bicolored for B, W* if every vertex not colored B or W is a singleton.

LEMMA 5.1. *A core-bicolored dag for B, W is consistently colored if it is not a blocker but it satisfies conditions* (i) *and* (iii) *of a blocker for B (B, W).*

Proof. Let H be a dag satisfying the hypotheses of the lemma. We consider each of the five varieties of blockers in turn. For each variety we assume H satisfies conditions (i) and (iii) of that blocker for B (or B, W), and we show H is consistently colored.

1-*blocker.* If $\overline{B}^+ = \emptyset$, then $V^+ \subseteq B$ and condition (iii) implies $V^- = R^-(V) = R^-(B) \subseteq \overline{B}$. Thus H is consistently colored. If $B^+ = \emptyset$, then $\Phi = |B^+| + |B^-| \geq |V^-|$ implies $V^- \subseteq B$. Again H is consistently colored.

2-*blocker.* If $B^+ = \emptyset$, then $V^+ \subseteq \overline{B}$ and (iii) implies $V^- = R^-(V) = R^-(\overline{B}) \subseteq B$. Hence H is consistently colored. If $\overline{B}^+ = \emptyset$, then $V^+ \subseteq B$. Thus $|B^+| + |B^-| \geq |V^+| = \Phi$ so equality holds, and $B^- = \emptyset$. Thus H is consistently colored.

3^+-*blocker.* Without loss of generality assume $W^+ = \emptyset$. With (iii) this shows $|B^+| = |V^+| - 1$. We can assume $B^+, B^- \neq \emptyset$ since otherwise H is consistently colored. Now (i) implies $\Phi = |B^+| + |B^-|$. In fact H is a 1-blocker for B, a contradiction.

3^--*blocker.* The reverse graph satisfies the hypotheses of the lemma for a 3^+-blocker (Lemma 4.2), so the previous argument applies.

4-*blocker.* Suppose $B^- = \emptyset$. Conditions (i) and (iii) imply there are $|V^+|-1$ black sources. We can assume $W^+, W^- \neq \emptyset$ since otherwise H is consistently-colored. Now H is a 2-blocker for W, a contradiction.

If $B^+ = \emptyset$, then using Lemma 4.2 the previous argument applies to the reverse graph. □

LEMMA 5.2. *If H is core-bicolored and $\Phi = |V^+| = |V^-| > |B^+|+|B^-|$ for every $B \in \mathcal{P}$, then $OPT = \Phi$ unless H is a blocker.*

Proof. Let H be core-bicolored for B, W. Let $G = V - B - W$, so every vertex of G is a singleton. Assume H is not a blocker. Lemma 2.7 shows we can assume H is not consistently colored. Lemma 5.1 implies condition (iii) of a 2-, 3-, or 4-blocker for B (B, W) fails. Define the sets

$$\mathcal{W} = \{(x^+, x^-) \in \overline{B}^+ \times \overline{B}^- : r(x^+, x^-)\};$$
$$\mathcal{B} = \{(x^+, x^-) \in \overline{W}^+ \times \overline{W}^- : r(x^+, x^-)\}.$$

We establish the properties of these sets in four claims.

Claim 1. $\mathcal{W}, \mathcal{B} \neq \emptyset$.

Proof. $\mathcal{W} \neq \emptyset$ since condition (iii) of a 2-blocker for B fails. □

Claim 2. No vertex is common to every ordered pair of $\mathcal{W} \cup \mathcal{B}$.

Proof. A common vertex z must belong to G. If $z \in G^+$, then $R^-(B) \subseteq W$, $R^-(W) \subseteq B$, and $G^+ = \{z\}$. We have deduced condition (iii) of a 3^+-blocker for B, W, a contradiction. A symmetric argument holds if $z \in G^-$. □

Claim 3. $\mathcal{W} \cup \mathcal{B}$ contains two ordered pairs that do not contain a common vertex.

Proof. Take any pair $(x^+, x^-) \in \mathcal{W} \cup \mathcal{B}$. Since x^+ is not in every pair of $\mathcal{W} \cup \mathcal{B}$, there is a pair containing some $y^+ \in V^+ - x^+$. The pair must be (y^+, x^-), else we are done. Similarly there is a pair (x^+, y^-) with $y^- \in V^- - x^-$. Pairs (y^+, x^-) and (x^+, y^-) have no common vertex. □

Claim 4. Some ordered pair in $\mathcal{W}$ and some ordered pair in $\mathcal{B}$ do not contain a common vertex.

Proof. Assume Claim 4 fails. Claim 3 shows there are ordered pairs (x^+, x^-), (y^+, y^-) with all four vertices distinct, so by symmetry assume both pairs belong to $\mathcal{W} - \mathcal{B}$. Claim 1 shows $\mathcal{B}$ has a pair, which must have a vertex in common with both (x^+, x^-) and (y^+, y^-). Hence assume $(x^+, y^-) \in \mathcal{B}$. This makes

$$x^+, y^- \in G, \quad x^-, y^+ \in W.$$

It is now clear that $\mathcal{B} = \{(x^+, y^-)\}$. This implies $R^-(B) \cup R^+(B) \subseteq W$.

Next we show $G = \{x^+, y^-\}$. Suppose there is a vertex $g^+ \in G^+ - x^+$. Let g^+ precede a sink z^-. Since $(g^+, z^-) \notin \mathcal{B}$, $z^- \in W$. We have $z^- \neq y^-$, $(g^+, z^-) \in \mathcal{W}$ and $(x^+, y^-) \in \mathcal{B}$, contradicting the failure of Case 4. A symmetric argument applies to a vertex $g^- \in G^- - y^-$.

Since $|G^+| + |G^-| = 2$, the hypothesis of the lemma implies $|B^+| + |B^-| = |W^+| + |W^-| = |V^+| - 1$. Hence condition (iii) of a 4-blocker for B, W is satisfied, a contradiction. □

By Claim 4, take $(x^+, x^-) \in \mathcal{B}$ and $(y^+, y^-) \in \mathcal{W}$ with all four vertices distinct. Recolor the vertices of G to B or W so $x^+, x^- \in B$, $y^+, y^- \in W$, and, if possible, $|V^+| \geq |B^+| + |B^-|, |W^+| + |W^-|$. If possible, this recoloring gives a bicolored graph K that has the same value of Φ and is not a 1-blocker. Hence Lemma 3.4 shows K has $OPT = \Phi$. The solution to the augmentation problem on K is valid on H and shows H has $OPT = \Phi$.

If the recoloring is impossible, the inequality hypothesized in the lemma and symmetry imply $x^+, x^- \in G$ and $|V^+| - 1 = |B^+| + |B^-|$. Since condition (iii) of a 4-blocker fails, $R^-(B) \cup R^+(B) \not\subseteq W$. Thus $\mathcal{B}$ contains a pair (z^+, z^-) with at least one black vertex.

Suppose the pairs $(z^+, z^-) \in \mathcal{B}$ and $(y^+, y^-) \in \mathcal{W}$ have all four vertices distinct. Use these pairs in the above recoloring procedure. The procedure succeeds and shows $OPT = \Phi$. To prove this note the procedure fails only if $y^+, y^- \in G$ and $|V^+| - 1 = |W^+| + |W^-|$. But this implies the contradiction $2|V^+| = |V^+| + |V^-| = (|B^+| + |B^-|) + (|W^+| + |W^-|) + (|G^+| + |G^-|) \geq 2(|V^+| - 1) + 4$.

Finally suppose vertices z^+, z^-, y^+, y^- are not all distinct. The unique common vertex belongs to G. This makes vertices z^+, z^-, x^+, x^- distinct. Furthermore $|W^+| + |W^-| \leq |V^+| - 2$, since $|B^+| + |B^-| = |V^+| - 1$ and $|G^+| + |G^-| \geq 3$. Thus we can use the pairs $(z^+, z^-) \in \mathcal{B}$ and $(x^+, x^-) \in \mathcal{W}$ in the recoloring procedure to show $OPT = \Phi$. □

We now put together the various results that solve the bipartite strong connectivity augmentation and cable-framework rigidity problems.

LEMMA 5.3. *Any core-bicolored dag has $OPT = \Phi$ unless it is a blocker, in which case $OPT = \Phi + 1$.*

Proof. If H is a blocker, then $OPT = \Phi + 1$ by Lemmas 4.3 and 2.9. If H is not a blocker, then $OPT = \Phi$ by Lemmas 4.5, 3.5, and 5.2.

Finally recall that we assumed the graph H has no isolated vertices. If there are isolated vertices, we use the transformation of Proposition 2.1. The transformed graph is a blocker if and only if the original graph is a blocker. This follows from the observation that the transformation does not change any of the colors appearing in any set $R^-(X)$, $R^+(X)$. □

6. Merging colors. The general case of our theorem is proved in two steps, presented in this section and the next. This section reduces the number of colors to at most three, i.e., a tricolored dag (Lemma 6.2). The next section completes the proof by analyzing tricolored dags.

We reduce the number of colors by merging colors. To *merge* color classes C_i, $i = 1, \ldots, k$, means to replace them by a new color $\bigcup_1^k C_i$ in $\mathcal{P}$. Throughout this section assume H is the given graph with partition $\mathcal{P}$, where $\mathcal{P}$ contains at least four nonempty color classes.

We start with two observations that follow easily from the definition of blocker. First, if H is not a blocker and a merge does not change Φ but results in a blocker for color B or colors B, W, then B or W is not a class of $\mathcal{P}$. Second, if a color class with no extreme vertices is merged with any other color, the new graph is an i-blocker if and only if the original graph is. In particular we can assume every color of $\mathcal{P}$ contains an extreme vertex.

We avoid creating blockers of certain types by using witnesses analogous to those used in Lemmas 3.2 and 3.3. The following lemma gathers together some useful facts for establishing witnesses.

LEMMA 6.1. *Let H be a blocker with exactly three colors C_i, $i = 1, 2, 3$.*

(i) *Suppose $r(C_i)$ for $i = 1, 2$. Then H is a 1- or 4-blocker for C_3.*

(ii) *Suppose $r(C_1)$, $r(C_2, C_3)$ and either $r(C_1, C_3)$ and $|C_1^+| > 1$ or $r(C_3, C_1)$ and $|C_1^-| > 1$. Then H is a 1-blocker for C_2 or C_3, or a 4-blocker for C_2.*

Proof. First consider an arbitrary dag having a color C with $r(C)$. The definition of blocker shows that H is not a 1-, 3-, or 4-blocker for C. If H is a 2-blocker, it is a 2-blocker for C.

Part (i) follows by applying the above observation. We turn to part (ii). If H is a 2-blocker, the above observation shows it is a 2-blocker for C_1, but $r(C_2, C_3)$ makes this impossible. If H is a 3^+-blocker, say, for vertex z, then $z \in C_1$ (since $r(C_1)$). But this implies $|C_1^+| = 1$ and $\neg r(C_3, C_1)$, contradicting the hypothesis. Similarly if H is a 3^--blocker for z, then $z \in C_1$. But this implies $|C_1^-| = 1$ and $\neg r(C_1, C_3)$, contradicting the hypothesis. H is not a 4-blocker for C_1 by the preliminary observation. Suppose H is a 4-blocker for C_3. The other color of the blocker must be C_2 since $r(C_2, C_3)$, but it must be C_1 since $r(C_1, C_3)$ or $r(C_3, C_1)$. This contradiction gives part (ii). □

LEMMA 6.2. *If a graph H has $|\mathcal{P}| \geq 4$, then some merge gives a new graph with the same value of Φ and $|\mathcal{P}| = 3$. Furthermore the new graph is a blocker only if H is.*

Proof. Call a merge of $\mathcal{P}$ into three classes *permissible* if it does not change Φ. A permissible merge can be found by repeatedly merging two color classes until a merge of classes C_1 and C_2 would produce a class with at least Φ extreme vertices. Then the desired permissible merge consists of classes C_1, C_2, and $\overline{C_1 \cup C_2}$.

For the rest of the argument assume H is not a blocker. We must show some permissible merge gives a graph that is not a blocker. Order the blocker types as 2, 1, 3, 4. We will consider the blockers in this order. For each type i, we suppose K is an i-blocker that results from a permissible merge of H. We will construct a new permissible merge of H which results in a graph L. We will show that L is not a blocker of type i or a type listed before i. Repeated application of this argument eventually gives the desired merge.

2- and 1-blockers.

Claim. Let V be the disjoint union of X and Y, where X and Y are both the result of a merge. Suppose further that $R^-(X) \subseteq Y$ and $X^+, X^- \neq \emptyset$. Then X (Y) can be partitioned into disjoint sets X_i (Y_i), $i = 0, 1$, where each X_i (Y_i) is the result of merging classes of $\mathcal{P}$ contained in X (Y), such that $r(X_i \cup Y_i)$ for $i = 0, 1$.

Proof. Start by choosing $X_0 \subseteq X$ as a class of $\mathcal{P}$ containing a source. Take $Y_0 \subseteq Y$ as a class of $\mathcal{P}$ containing a sink preceded by X_0^+. Let X_1 and Y_1 consist of the remaining classes of $\mathcal{P}$ contained in X (Y). If $r(X_1 \cup Y_1)$, we are done. Hence assume the opposite:

$$R^-(X_1) \subseteq Y_0 \text{ and } R^-(Y_1) \subseteq X_0 \cup Y_0.$$

We will prove $r(X_0 \cup Y_1)$ and $r(X_1 \cup Y_0)$. Clearly this implies the claim.

The first relation $r(X_0 \cup Y_1)$ holds if $r(Y_1, X_0)$, so suppose the opposite. This implies $R^-(Y_1) \subseteq Y_0$. Next note that $Y_0^+ \neq \emptyset$. This follows since $X^- \neq \emptyset$, so a sink in X is preceded by a source of Y, and that source cannot be in Y_1. Now since H is not a 2-blocker for Y_0, $R^-(X_0) \not\subseteq Y_0$. Thus $r(X_0, Y_1)$. This implies the first relation.

Consider the second relation $r(X_1 \cup Y_0)$. If $X_1^+ \neq \emptyset$, then $r(X_1, Y_0)$. If $X_1^+ = \emptyset$, then $X_1^- \neq \emptyset$, which implies $r(Y_0, X_1)$. □

Now suppose K is a 2- or 1-blocker. First note that the claim applies to B and $\overline{B}$. (For a 2-blocker the claim requires $\overline{B}^- \neq \emptyset$. For a 1-blocker the claim requires $B^- \neq \emptyset$. Both of these are shown in the proof of Lemma 4.2.)

So let B $(\overline{B})$ be partitioned into sets A_i (C_i), $i = 0, 1$, where $r(A_i \cup C_i)$ for $i = 0, 1$. For at least one index i, $A_i \cup C_i$ contains at most Φ extreme vertices. Without loss of generality let the index be $i = 0$. Let the three new color classes be $A_0 \cup C_0$, A_1, and C_1.

The new graph L is not a 2-blocker. In proof $r(A_0 \cup C_0)$ implies L can only be a 2-blocker for $X = A_0 \cup C_0$. However, this is inconsistent with $r(\overline{X})$, i.e., $r(A_1 \cup C_1)$.

L is not a 1-blocker if K is a 1-blocker. In proof suppose L is a 1-blocker for color X. $X \neq A_0 \cup C_0$ because $r(A_0 \cup C_0)$. $X \neq A_1, C_1$ since $A_0 \cup A_1$ contains exactly Φ extreme vertices, so $C_0 \cup C_1$ contains at most Φ extreme vertices. This implies A_1 and C_1 both contain fewer than Φ extremes.

3^+-blockers. Suppose K is a proper 3^+-blocker for B, W.

Claim. Without loss of generality B can be partitioned into disjoint sets A_i, $i = 0, 1$, where each A_i is a merge of classes of $\mathcal{P}$, such that (i) $r(A_0, W)$, (ii) $r(A_1, W)$ or $r(W, A_1)$.

Proof. Suppose B contains at least two classes of $\mathcal{P}$. Let A_0 be a class of $\mathcal{P}$ contained in B that contains a source, and let A_1 consist of the remaining classes of $\mathcal{P}$ in B. Clearly property (i) holds. Suppose (ii) fails. Thus $A_1^+ = \emptyset$ and $\neg r(W, A_1)$. This means that K is a proper 3^+-blocker for A_0, W. W must contain at least two classes of $\mathcal{P}$ since H is not a blocker. Repeat the above argument on the set W instead of B. This time (ii) cannot fail, since that would make H a 3-blocker. □

Assume B, A_0, and A_1 satisfy the claim. Let $G = \overline{B \cup W}$. Either $A_0 \cup W$ or $A_1 \cup G$ contains at most Φ extreme vertices. Thus we get a permissible merging consisting of either colors $A_0 \cup W$, A_1, and G or A_0, W, and $A_1 \cup G$. Recall the relation $r(G)$ from Lemma 4.4. Also note that A_0, A_1, and W each have fewer than Φ extreme vertices, since both B and W contain fewer than Φ extremes in any proper 3-blocker for B, W.

Suppose the permissible merging is $A_0 \cup W$, A_1, and G. Lemma 6.1(i) with $r(A_0 \cup W)$ and $r(G)$ shows that if L is a blocker, it is a 1-blocker for A_1 or a 4-blocker. The former is impossible since, as noted, A_1 has fewer than Φ extreme vertices.

Suppose the permissible merging is $A_1 \cup G$, A_0, and W. These sets satisfy the hypothesis of Lemma 6.1(ii), since we have $r(A_1 \cup G)$, $r(A_0, W)$ and furthermore, either $r(A_1, W)$, in which case $A_1 \cup G$ has at least two sources, or $r(W, A_1)$, in which case $A_1 \cup G$ has at least two sinks. Now Lemma 6.1(ii) shows that if L is a blocker, it is either a 1-blocker for A_0 or W, or L is a 4-blocker. It is not a 1-blocker because, as noted above, both A_0 and W have fewer than Φ extreme vertices.

3^--blockers. Suppose K is a proper 3^--blocker. Lemma 4.2 shows its reverse graph is a proper 3^+-blocker. Apply the construction for proper 3^+-blockers to the reverse graph. If the new merging is a blocker, it must be a 4-blocker. The same holds for the original graph.

4-blockers. Suppose K is a proper 4-blocker. Let $X = B$, $Y = W$ if B contains two or more classes of $\mathcal{P}$; otherwise let $X = W$, $Y = B$. So X contains two or more classes of $\mathcal{P}$. Let $G = \overline{B \cup W} = \overline{X \cup Y}$.

Claim. X can be partitioned into disjoint sets A_i, $i = 0, 1$, where each A_i is a merge of classes of $\mathcal{P}$, such that $r(A_0, Y)$ and either $r(A_1, Y)$ or $r(Y, A_1)$. In addition A_0 and A_1 each contain fewer than $\Phi - 1$ extreme vertices. Furthermore $r(G)$.

Proof. The last part of the claim $r(G)$ holds by Lemma 4.4.

Suppose $X = B$. Let A_0 be a class of $\mathcal{P}$ contained in X that contains a source, and let A_1 consist of the remaining classes in B. Hence $r(A_0, W)$. Furthermore $r(A_1, W)$ or $r(W, A_1)$. In addition each A_i has fewer than $\Phi - 1$ extreme vertices since $A_i \subset B$.

Suppose $X = W$. Let A_0 be a class of $\mathcal{P}$ contained in W such that $r(A_0, B)$. Let A_1 consist of the remaining classes of $\mathcal{P}$ in W. We have $r(A_1, B)$ or $r(B, A_1)$, since

otherwise H is a 4-blocker for B, A_0 (both classes of $\mathcal{P}$). Each set A_i contains fewer than $2\Phi - (\Phi - 1) - 2 = \Phi - 1$ extreme vertices. □

Either $A_0 \cup Y$ or $A_1 \cup G$ contains at most Φ extreme vertices. Thus we get a permissible merging consisting of colors $A_0 \cup Y$, A_1, and G or A_0, Y, and $A_1 \cup G$.

Consider a permissible merging $A_0 \cup Y$, G, and A_1. These sets satisfy the hypothesis of Lemma 6.1(i). The new graph L is not a 1- or 4-blocker for A_1, since the claim shows A_1 has too few extreme vertices. Thus L is not a blocker.

Consider a permissible merging $A_1 \cup G$, A_0, and Y. These sets satisfy the hypothesis of Lemma 6.1(ii). L is not a 1- or 4-blocker for A_0, since the claim shows A_0 has too few extremes. Finally L is not a 1-blocker for Y, since Y is B or W and K is a proper 4-blocker. This proves Lemma 6.2. □

We finish this section with a simple result also proved by merging colors. The result can be seen from Lemma 5.3, but the following proof is independent of section 5.

LEMMA 6.3. *Let H be a tricolored dag with $\Phi = |V^+| = |V^-|$, where some color does not contain a source. Then $OPT = \Phi$ if H is not a blocker.*

Proof. Let the extreme vertex colors of H be B, W, and G, where $G^+ = \emptyset$. By Lemma 4.5 assume each color has fewer than Φ extreme vertices.

We can change the color of every extreme vertex of G to B or W without changing Φ. This follows since $2\Phi = |V^+| + |V^-|$, and we start with both B and W containing fewer than Φ extremes. In fact we can do this recoloring even with the requirement that a given vertex of G be colored B and another given vertex of G be colored W.

If every source has the same color, then $\Phi = |V^+|$ implies H is consistently colored. In this case $OPT = \Phi$ (Lemma 2.7). So assume $B^+, W^+ \neq \emptyset$. We can assume H is not a 2-blocker, so $r(\overline{B})$. This implies $r(W, \overline{B})$. Similarly $r(B, \overline{W})$.

G contains at least two sinks, since $|V^+| + |V^-| = 2\Phi \geq |B^+| + |B^-| + |W^+| + |W^-| + 2$. Now it is easy to see that there is a sink $w \in \overline{B}$ preceded by a source of W and a different sink $b \in \overline{W}$ preceded by a source of B.

If b is colored G, change it to B. If w is colored G, change it to W. Change the color of the remaining extreme vertices of G to B or W without changing Φ. The new bicolored graph is not a 1-blocker since $r(B)$ and $r(W)$. Lemma 3.4 shows it has a solution with $OPT = \Phi$ new edges, which gives a similar solution on H. □

7. General augmentation theorem. This section solves the augmentation problem for tricolored dags as follows. Section 7.1 presents various witness relations that allow us to avoid generating blockers, as in sections 3 and 6. Then section 7.2 shows how to find an augmenting pair that does not create a blocker.

We use the following notational conventions throughout this section. The three colors of the tricolored dag are designated as B, W, and G ("black," "white," and "gray"). Sources of $B, W, G, \overline{B}, \overline{W}, \overline{G}$ are usually designated by $b, w, g, \overline{b}, \overline{w}, \overline{g}$, respectively. Sinks of $B, W, G, \overline{B}, \overline{W}, \overline{G}$ are usually designated by $\beta, \omega, \gamma, \overline{\beta}, \overline{\omega}, \overline{\gamma}$, respectively.

7.1. Witness relations. We begin with some simple relations that preclude various blocker types.

LEMMA 7.1. *Let H be a tricolored blocker with colors B, W, G.*

(i) *$r(B)$ and $r(\overline{B})$ imply H is not a 2-blocker.*

(ii) *$r(\overline{B})$, $r(\overline{W})$, and $r(\overline{G})$ imply H is not a 2-blocker.*

(iii) *$r(B, W)$, $r(W, G)$, and $r(G, B)$ imply H is a 1-blocker.*

(iv) *Suppose*

$$r(\overline{b}, \omega), \ r(w, \beta), \text{ and } r(g, \gamma)$$

for $\overline{b} \in \overline{B}^+$, $\omega \in W^-$, $w \in W^+$, $\beta \in B^-$, $g \in G^+$, $\gamma \in G^-$, where the six vertices are distinct (i.e., $\overline{b} \neq w, g$). Then either H is a 1-blocker or H is a 4-blocker for B.

Proof. Parts (i) and (ii) follow from the definition of 2-blocker.

(iii) H is not a 2-blocker for B because $r(W, G)$. H is not a 3^+-blocker for B because $r(B, W)$ and $r(W, G)$. Similarly H is not a 3^--blocker for B because $r(G, B)$ and $r(W, G)$. H is not a 4-blocker for B because $r(B, W)$ and $r(G, B)$. Now symmetry of the colors gives the desired conclusion.

(iv) $r(g, \gamma)$ and $r(w, \beta)$ show H is not a 2-blocker. If H is a 3^+-blocker, then $r(g, \gamma)$ shows it is a 3^+-blocker for vertex g. Since $\overline{b} \neq g$ we must have $\overline{b} \in W$, but then $r(\overline{b}, \omega)$ shows H is not a 3^+-blocker. If H is a 3^--blocker, then $r(g, \gamma)$ shows it is a 3^--blocker for γ. But this is inconsistent with $r(\overline{b}, \omega)$. $r(g, \gamma)$ shows H is not a 4-blocker for G. $r(w, \beta)$ and $r(\overline{b}, \omega)$ show it is not a 4-blocker for W. □

Now we analyze the blocker types that can occur after augmenting a pair in some commonly occurring configurations.

COROLLARY 7.2. *Let H be a tricolored dag, with colors B, W, G, that is not a blocker.*

(i) *Suppose*

$$r(b, \chi), r(x, \beta), \text{ and } r(\overline{b}, \overline{\beta})$$

for six distinct vertices $b \in B^+$, $\chi \in V^-$, $x \in V^+$, $\beta \in B^-$, $\overline{b} \in \overline{B}^+$, $\overline{\beta} \in \overline{B}^-$, where χ and x have different colors. Augmenting the compatible pair χ, x does not give a 2-blocker.

(ii) *Suppose*

$$r(g, \gamma), r(w, \chi), \text{ and } r(x, \omega)$$

for six distinct vertices $g \in G^+$, $\gamma \in G^-$, $w \in W^+$, $\chi \in V^-$, $x \in V^+$, and $\omega \in W^-$, where χ and x have different colors. Augmenting the compatible pair χ, x gives either a nonblocker, a 1-blocker, or a 4-blocker for B. The latter cannot occur if $B - \{x, \chi\}$ has less than $\Phi - 2$ extremes.

(iii) *Suppose*

$$r(b, \omega),\ r(w, \beta),\ r(g, \gamma), \text{ and } r(\overline{b}, \overline{\beta})$$

for eight distinct vertices $b \in B^+$, $\omega \in W^-$, $w \in W^+$, $\beta \in B^-$, $g \in G^+$, $\gamma \in G^-$, $\overline{b} \in \overline{B}^+$, $\overline{\beta} \in \overline{B}^-$ (i.e., $\overline{b} \neq w, g$ and $\overline{\beta} \neq \omega, \gamma$). Augmenting the compatible pair $\overline{\beta}, b$ gives either a nonblocker, a 1-blocker, or a 4-blocker for B. The latter cannot occur if B has less than $\Phi - 1$ extremes.

Proof. (i) The new graph has $r(b, \beta)$ and $r(\overline{b}, \overline{\beta})$, so it is not a 2-blocker by Lemma 7.1(i).

(ii) Follows from Lemma 6.1(i).

(iii) Follows from Lemma 7.1(iv). □

7.2. Finding an augmenting pair. To complete the solution of our augmentation problem we wish to show that a dag H that is not a blocker has $OPT = \Phi$. Toward this end we can make the following assumptions: H satisfies

$$\Phi = |V^-| = |V^+| > |B^+| + |B^-| \text{ for every } B \in \mathcal{P},$$

since as mentioned at the end of section 4 this is the only case that remains to be analyzed. H has exactly three nonempty colors, by Lemmas 3.4 and 6.2. Each color

contains both a source and a sink, by Lemma 6.3. We make all these assumptions throughout this section until the final result, Theorem 7.8, is presented. Now we give the overall structure of the argument proving that H has $OPT = \Phi$.

Lemma 2.8 shows H has a compatible pair. Augment such a pair. This decreases Φ by 1, by the above assumption on Φ. The new graph K is still tricolored. If K is not a blocker, induction (and the analysis of the previous sections) shows we are done. So assume K is a blocker. It suffices to show how to augment a different compatible pair so the new graph is not a blocker.

We accomplish this in a manner similar to section 6: Order the blocker types as 2, 3, 4, 1. For each type i, we suppose K is an i-blocker resulting from an augment of H. We will augment a new compatible pair of H to get a new graph L. We will show L is not a blocker of type i or a type listed before i. Repeated application of this argument eventually gives the desired augment.

The rest of this section shows for each blocker type how to find a compatible pair giving the desired graph L. The arguments refer to the three graphs H, K, L. The functions R^- and R^+ always refer to H, as does the predicate r. r_K and r_L refer to graphs K and L, respectively.

The following simple principle will be used repeatedly. For motivation note that if χ, x is a compatible pair, then some source u and sink ξ satisfy $r(u, \chi)$ and $r(x, \xi)$.

LEMMA 7.3. *Suppose graph K results from augmenting the pair χ, x. Then $r_K(a, \delta)$ implies $r(a, \delta)$ when*

$$r(u, \chi), r(x, \xi), \text{ and either } \neg r_K(a, \xi) \text{ or } \neg r_K(u, \delta)$$

for vertices $u, x, a \in V^+$, $\chi, \xi, \delta \in V^-$.

Proof. $r_K(a, \delta)$ with $\neg r(a, \delta)$ means the path from a to δ in K includes edge (χ, x). Thus $r_K(a, \xi)$ and $r_K(u, \delta)$. □

Now we present the argument for each blocker type. It is convenient to consider 1-blockers first since that case gets used by the others.

1-blockers. This section shows that if an augment gives a 1-blocker, a different augment gives a bicolored graph. In fact we prove a stronger result that is useful later on.

Suppose an augment of H gives a 1-blocker for B. B must have $\Phi - 1$ extremes (by the assumption on Φ) and the augment cannot involve a vertex colored B. The latter implies $\neg r(B)$. These facts show the following lemma handles the case of augments that give 1-blockers.

LEMMA 7.4. *Let H be a dag having a color B with exactly $\Phi - 1$ extreme vertices and $\neg r(B)$. Then some augment of H gives a dag L that is not a blocker. Furthermore two colors of L can be merged without changing Φ to get a bicolored dag that is not a blocker.*

Proof. Vertices x^-, y^+ form a *good pair* if neither vertex is colored B, the vertices are compatible, and augmenting x^-, y^+ gives a new graph L with

$$r_L(B) \text{ and } r_L(\overline{B}).$$

We will find a good pair of vertices. This suffices to prove the lemma: L has $\Phi = |B^+| + |B^-| = |\overline{B}^+| + |\overline{B}^-|$, so using colors $B, \overline{B}$ gives a bicolored graph that is not a 1-blocker, as desired. We will argue by contradiction. So suppose no good pair exists.

We first show there are four distinct vertices $b \in B^+$, $\omega \in W^-$, $g \in G^+$, $\beta \in B^-$ such that

$$r(b, \omega) \text{ and } r(g, \beta).$$

In proof $\neg r(B)$ implies there are vertices $b \in B^+$ and $\beta \in B^-$ with $r(b,\chi)$, $r(x,\beta)$ for some $x, \chi \notin B$. Since H is not a 4-blocker x and χ can be chosen to have different colors. Naming colors appropriately gives the desired vertices.

The compatible pair ω, g is a good pair if $r_L(\overline{B})$. (Whenever we refer to a good pair, L denotes the graph that results from augmenting that pair.) The latter holds if either $r(\overline{B}-g, \overline{B}-\omega)$ or $r(\overline{B}-g,\omega)$ and $r(g,\overline{B}-\omega)$. So neither of these two alternatives holds. The failure of the second alternative implies that we can assume $\neg r(\overline{B}-g,\omega)$ (since we can replace H by its reverse graph). This plus the failure of the first alternative implies

$$R^-(\overline{B}-g) \subseteq B. \tag{7.1}$$

Equation (7.1) with $r(\overline{B})$ (since H is not a 2-blocker) implies there is a vertex $\overline{\beta} \in \overline{B}^-$ with

$$r(g,\overline{\beta}).$$

(It is possible that $\overline{\beta} = \omega$.)

Next we show

$$r(B,G) \Longrightarrow G^- = \{\overline{\beta}\} \text{ and } R^+(W) \subseteq B. \tag{7.2}$$

Suppose $r(b',\gamma)$ for vertices $b' \in B^+, \gamma \in G^-$. $W^+ \neq \emptyset$ and (7.1) show there are vertices $w \in W^+, \beta' \in B^-$ with $r(w,\beta')$. The compatible pair γ, w is good if $r_L(\overline{B})$, so $\neg r_L(\overline{B})$. Since $r(g,\overline{\beta})$ and $g \neq w$ we must have $\gamma = \overline{\beta}$. Thus $\overline{\beta} \in G$. Furthermore $R^+(W) \subseteq B$, since otherwise we can choose $\overline{\beta} \in W$.

We complete the proof of (7.2) by showing $G^- = \{\overline{\beta}\}$: Consider a vertex $\gamma' \in G^- - \overline{\beta}$. We have shown $\gamma = \overline{\beta}$ holds for every choice of γ and $\overline{\beta}$, so these two vertices are unique. We have $\neg r(B,\gamma')$ (since otherwise γ' could be chosen as γ), $\neg r(g,\gamma')$ (since otherwise γ' could be chosen as $\overline{\beta}$), and $\neg r(\overline{B}-g,\gamma')$ (by (7.1)). Hence γ' does not exist.

We can now assume without loss of generality that $\overline{\beta} \in G$: This relation has just been proved if $r(B,G)$. On the other hand $\neg r(B,G)$ implies any vertex of G^- can be chosen as $\overline{\beta}$.

Next we show $G^+ = \{g\}$: Suppose $g' \in G^+ - g$. g' precedes a black sink β' by (7.1). The pair ω, g' is compatible since we have $r(b,\omega)$ and $r(g',\beta')$. In fact ω, g' is a good pair: The relation $r_L(\overline{B})$ holds since $r(g,\overline{\beta})$ and $\overline{\beta} \neq \omega$ (because $\overline{\beta} \in G$). Since there are no good pairs, g' does not exist.

Since H is not a 3^+-blocker but $|G^+| = 1$ and $R^-(W) \subseteq B$, we have $r(B,\overline{W})$. Together with the hypothesis $\neg r(B)$ we get $r(B,G)$. Hence the hypothesis and conclusion of (7.2) hold. Thus $|G^+| = |G^-| = 1$, so W (as well as B) has exactly $\Phi - 1$ extreme vertices. This along with $R^-(W) \cup R^+(W) \subseteq B$ makes H a 4-blocker, a contradiction. □

2-blockers.

LEMMA 7.5. *If an augment of H gives a graph K that is a 2-blocker, a different augment of H gives a graph L that is not a 2-blocker.*

Proof. Suppose K is a 2-blocker for B. Since H is not a 2-blocker there are vertices $x \in \overline{B}^+, \chi \in \overline{B}^-$ with $r(x,\chi)$ and either x or χ (or both) a vertex in the augment. By possibly changing to the reverse graph (using Lemma 4.2) we can assume x is a vertex in the augment. Without loss of generality assume $x \in G^+$. We consider two cases.

Case 1. χ is not a vertex in the augment.

In this case the augment forming K gives four distinct vertices $b \in B^+$, $\overline{\gamma} \in \overline{G}^-$, $g \in G^+$, $\overline{\beta} \in \overline{B}^-$ such that

$$r(b, \overline{\gamma}) \text{ and } r(g, \overline{\beta}).$$

Here we have replaced x and χ by g and $\overline{\beta}$, respectively, and b and $\overline{\gamma}$ are the other two vertices involved in the augment. $\overline{\gamma}$ must be in $\overline{G}$ since the edge added by the augment joins differently colored vertices. b must be in B because it precedes $\overline{\beta}$ in K, which is a 2-blocker for B. Finally $\overline{\gamma} \neq \overline{\beta}$ by the assumption of Case 1.

Next we show there are vertices $\overline{b} \in \overline{B}^+$, $\beta \in B^-$ that are distinct from the other four vertices such that

$$r(\overline{b}, \beta).$$

In proof, such a pair of vertices with $r_K(\overline{b}, \beta)$ exists, since a 2-blocker has $\overline{B}^+ \neq \emptyset$. The relation $r(\overline{b}, \beta)$ follows from Lemma 7.3 (using $\neg r_K(\overline{b}, \overline{\beta})$).

If $\overline{b}$ can be chosen to have a color different from $\overline{\gamma}$, then Corollary 7.2(i) applies to the six vertices $b, \overline{\gamma}, \overline{b}, \beta, g, \overline{\beta}$ and we are done. So suppose no such vertex $\overline{b}$ exists. This implies

$$\overline{\gamma},\ \overline{b} \in W.$$

In proof, the two vertices have different colors if $\overline{\gamma} \in B$. Hence $\overline{\gamma} \in W$ and so $\overline{b} \in W$.

We now show $R^-(W \cup G - g) \subseteq B$. If not, then $r(W \cup G - g, \overline{B})$. This continues to hold in graph K, even if the only sink of $\overline{B}$ reachable from $W \cup G - g$ is $\overline{\gamma}$. But this contradicts K a 2-blocker.

The inclusion $R^-(W \cup G - g) \subseteq B$ implies $|G^+| = 1$ (otherwise a vertex of $G^+ - g$ could be chosen as $\overline{b}$; here we use $\overline{\gamma} \in W$). The inclusion also implies $R^-(W) \subseteq B$. Since H is not a 3^+-blocker we conclude $r(B, \overline{W})$. Hence we can take vertices $b' \in B^+$, $\overline{\omega} \in \overline{W}^-$ with

$$r(b', \overline{\omega}).$$

Corollary 7.2(i) applies to the six vertices $b', \overline{\omega}, \overline{b}, \beta, g, \overline{\beta}$ if the three sinks are distinct (recall $\overline{b} \in W$). That leaves us with two possibilities, $\overline{\omega} = \beta$ and $\overline{\omega} = \overline{\beta}$.

Suppose $\overline{\omega} = \beta$. Form L by augmenting the pair $\overline{\beta}, b$. If $b' \neq b$, then $r_L(b', \beta)$ and $r_L(g, \overline{\gamma})$, with $\overline{\gamma} \in W$, show L is not a 2-blocker by Lemma 7.1(i). If $b' = b$, then $r_L(g, \overline{\gamma})$, $r_L(g, \beta)$, and $r_L(\overline{b}, \beta)$ with $\overline{\gamma}, \overline{b} \in W$ show L is not a 2-blocker by Lemma 7.1(ii).

Suppose $\overline{\omega} = \overline{\beta}$. Form L by augmenting the pair β, g. We have $r_L(b, \overline{\gamma})$, $r_L(b', \overline{\beta})$, and $r_L(\overline{b}, \overline{\beta})$. The supposition implies $\overline{\beta} \in G$, and we have $\overline{\gamma}, \overline{b} \in W$. So Lemma 7.1(ii) shows L is not a 2-blocker.

Case 2. χ is a vertex in the augment.

The augment forming K gives four distinct vertices $b \in B^+$, $\omega \in W^-$, $g \in G^+$, $\beta \in B^-$ such that

$$r(b, \omega), r(g, \omega), \text{ and } r(g, \beta).$$

Here we have replaced x and χ by g and ω, respectively. We can assume $b, \beta \in B$ because otherwise, renaming vertices puts us in Case 1.

Lemma 6.3 shows there are vertices $w \in W^+$ and $\gamma \in G^-$. Clearly these vertices are distinct from the four vertices above. Since K is a 2-blocker, $\neg r_L(w,\gamma)$, so $\neg r(w,\gamma)$. Thus w,γ is a compatible pair. Form L by augmenting w,γ. The three relations displayed above hold in L. So Lemma 7.1(ii) shows that L is not a 2-blocker. □

3-blockers.

LEMMA 7.6. *If an augment of H gives a graph K that is a proper 3^+-blocker, a different augment of H gives a graph L that is not a 2- or 3-blocker.*

Proof. Suppose K is a 3^+-blocker for B, W, and vertex $g \in G$. H has $|G^+|$ equal to 1 or 2. We consider these cases separately.

Case 1. $|G^+| = 2$.

Case 1.1. $R^-(B) \subseteq W$ and $R^-(W) \subseteq B$.

If B has $\Phi - 1$ extreme vertices, we are done by Lemma 7.4. Hence we can assume both B and W have less than $\Phi - 1$ extremes. This implies $|G^-| \geq 2$.

We claim there are vertices $b \in B^+$, $\omega \in W^-$, $w \in W^+$, $\beta \in B^-$, $g \in G^+$, and $\gamma \in G^-$ with

$$r(b,\omega), r(w,\beta), \text{ and } r(g,\gamma).$$

The first two relations follow from the assumptions of Case 1.1. $r_K(G)$ holds since K is a proper 3-blocker (recall Lemma 4.4). This implies $r(G)$, since the augment involves a source of G^+. Now we have proved the last relation $r(g,\gamma)$.

There are vertices $g' \in G^+ - g$ and $\gamma' \in G^- - \gamma$. Suppose these vertices can be chosen so $r(g',\gamma')$. This gives eight vertices satisfying the hypothesis of Corollary 7.2(iii), so we are done. Now assume such vertices do not exist.

Case 1.1 implies $R^+(G) \subseteq G$. So the assumption just made implies $R^+(G) = \{g\}$. Thus g' precedes a sink of $B \cup W$. Without loss of generality $r(g',\omega')$ for some $\omega' \in W^-$. Now the relations $r(g,\gamma)$, $r(w,\beta)$, and $r(g',\omega')$ satisfy the hypothesis of Corollary 7.2(ii), so we are done.

If Case 1.1 does not hold, then without loss of generality the following applies.

Case 1.2. $R^-(B) \not\subseteq W$.

The augment that produces K involves the source $g' \in G^+$. Hence there are vertices $b \in B^+$, $\beta \in B^-$, $\omega \in W^-$ with

$$r(b,\beta) \text{ and } r(g',\omega).$$

Here the augment is for β, g'. The assumption of Case 1.2 implies we must have β a nonwhite vertex preceded by a black source b. Since β is also nongray, $\beta \in B$. Finally $\omega \in W$ since $r_K(b,\omega)$.

Since K is a proper 3-blocker there are vertices $g \in G^+ - g'$, $\gamma \in G^-$ with $r_K(g,\gamma)$. This implies $r(g,\gamma)$ by Lemma 7.3 (since $\neg r_K(b,\gamma)$). Similarly there are vertices $w \in W^+, \beta' \in B^- - \beta$ with $r(w,\beta')$ (use $\neg r_K(b,\beta')$). Now the relations $r(g,\gamma)$, $r(w,\beta')$, and $r(g',\omega)$ show Corollary 7.2(ii) applies and we are done.

Case 2. $|G^+| = 1$.

Case 2.1. $R^-(B) \not\subseteq W$, $R^-(W) \not\subseteq B$.

We can assume there are vertices $b \in B^+$, $\overline{\omega} \in \overline{W}^-$, $w \in W^+$, $\omega \in W^-$ such that

$$r(b,\overline{\omega}) \text{ and } r(w,\omega).$$

Here the augment is for $\overline{\omega}, w$. The source of the augment is not in G by Case 2, so without loss of generality $w \in W$. The first relation of Case 2.1 now shows $\overline{\omega} \in \overline{W}$ and is preceded by a source $b \in B$. Finally $\omega \in W^-$ since $r_K(b,\omega)$.

Since K is a 3^+-blocker it contains vertices $w' \in W^+ - w$, $\beta \in B^- - \overline{\omega}$ such that $r_K(w', \beta)$. Lemma 7.3 shows $r(w', \beta)$, since $\neg r_K(w', \omega)$. Now the relations $r(w, \omega)$, $r(b, \overline{\omega})$, and $r(w', \beta)$ show Corollary 7.2(ii) applies and we are done.

If Case 2.1 does not hold, then without loss of generality the following applies.

Case 2.2. $R^-(B) \subseteq W$, $R^-(W) \not\subseteq B$.

Call vertices $w \in W^+$, $\overline{\beta} \in \overline{B}^-$ a *bad pair* if $r(w, \overline{\beta})$. The augment producing K involves a vertex in a bad pair. We examine three corresponding possibilities.

First suppose there is only one bad pair, and the augment involves both its vertices. Then there are vertices $g \in G^+$, $\gamma \in G^-$, $w \in W^+$, $\beta \in B^-$ such that

$$r(g, \gamma), r(w, \gamma), \text{ and } r(w, \beta),$$

where the augment is for the bad pair γ, w. $\beta \in B$ since w, β is not a bad pair. $\gamma \in G$ because $\gamma \in \overline{B}$ (since γ, w is bad) and $\gamma \in \overline{W}$ (since we augment γ, w). This implies $g \in G$ since g, γ is not a bad pair and $R^-(B) \subseteq W$.

Since K is a 3^+-blocker it contains vertices $b \in B^+$, $\omega \in W^-$ with $r_K(b, \omega)$. Lemma 7.3 shows $r(b, \omega)$. Lemma 4.4 shows there is a vertex $\gamma' \in G^- - \gamma$ with $r_K(g, \gamma')$. The uniqueness of the bad pair shows $\neg r(w, \gamma')$, so $r(g, \gamma')$. The relations $r(g, \gamma')$, $r(w, \gamma)$, and $r(b, \omega)$ show Corollary 7.2(ii) applies and we are done.

Now we can assume there is a bad pair with only one of its vertices in the augment. Suppose the sink is in the augment. Then there are vertices $w, w' \in W^+$, $\gamma \in G^-$, $\beta \in B^-$ such that

$$r(w', \gamma) \text{ and } r(w, \beta),$$

where the augment is for γ, w. The assumed bad pair is w', γ, so $w' \in W$. This implies $\beta \in B$ since K is a 3^+-blocker. The assumptions of Cases 2 and 2.2 imply $w \in W$. Finally $\gamma \in G$ since $\gamma \in \overline{B}$ (for a bad pair) and $\gamma \in \overline{W}$ (by the augment).

Since K is a 3^+-blocker it contains vertices $b \in B^+$, $\omega \in W^-$ with $r_K(b, \omega)$. Lemma 7.3 shows $r(b, \omega)$, since $\neg r_K(w', \omega)$. Lemma 4.4 shows there are vertices $g \in G^+$, $\gamma' \in G^- - \gamma$ with $r_K(g, \gamma')$. Lemma 7.3 shows $r(g, \gamma')$, since $\neg r_K(w', \gamma')$. The relations $r(g, \gamma')$, $r(w', \gamma)$, and $r(b, \omega)$ show Corollary 7.2(ii) applies and we are done.

In the remaining case there is a bad pair having only its source in the augment. There are vertices $g \in G^+$, $\overline{\omega} \in \overline{W}^-$, $w \in W^+$, $\overline{\beta} \in \overline{B}^-$ such that

$$r(g, \overline{\omega}) \text{ and } r(w, \overline{\beta}),$$

where the augment is for $\overline{\omega}, w$. $w, \overline{\beta}$ is the assumed bad pair. $g \in G$ because $g \notin B$ (since $R^-(B) \subseteq W$) and $g \notin W$ (since K is a 3^+-blocker).

Since K is a 3^+-blocker there are vertices $w' \in W^+ - w$, $\beta \in B^- - \overline{\omega}$ with $r_K(w', \beta)$. Lemma 7.3 with $\neg r_K(w', \overline{\beta})$ shows

$$r(w', \beta).$$

The pair $\overline{\omega}, w'$ is compatible, so augment it to get the graph L. We will show L has the desired properties. We consider two cases depending on the color of $\overline{\beta}$.

Suppose every possible choice for $\overline{\beta}$ is colored W. Since K is a 3^+-blocker there is a vertex $\gamma \in G^- - \overline{\omega}$ with $r_K(g, \gamma)$ (Lemma 4.4). The assumption on $\overline{\beta}$ shows $\neg r(w, \gamma)$, which implies $r(g, \gamma)$. Graph L has $r_L(g, \gamma)$ and $r_L(w, \overline{\beta})$. Since $\overline{\beta} \in W$, Lemma 6.1(i) shows we are done.

Suppose we can choose $\overline{\beta} \in G$. The hypothesis of Case 2.2 shows there are vertices $b \in B^+$, $\omega \in W^-$ with $r(b, \omega)$. Graph L has $r_L(g, \beta)$, $r_L(b, \omega)$, and $r_L(w, \overline{\beta})$. Since $\overline{\beta} \in G$, Lemma 7.1(iii) gives the desired conclusion. □

It remains to consider when an augment of H gives a graph K that is a proper 3^--blocker. Lemma 4.2 shows the reverse of K is a proper 3^--blocker resulting from an augment. So the above lemma shows a different augment gives a graph L that is not a 2- or 3-blocker. The reverse of L is the result of the reverse augment and is also not a 2- or 3-blocker.

4-blockers.

LEMMA 7.7. *If an augment of H gives a graph K that is a proper 4-blocker, a different augment of H gives a graph L that is a nonblocker or a 1-blocker.*

Proof. Suppose K is a proper 4-blocker for B. In H color B has $\Phi - 1$ or $\Phi - 2$ extreme vertices, since the augment decreases Φ by one. We consider these cases separately.

Case 1. B has $\Phi - 1$ extreme vertices.

Lemma 7.4 shows we can assume $r(B)$. Hence the augment producing K involves a black extreme vertex. By possibly changing to the reverse graph, assume it involves a black source. Thus there are vertices $w \in W^+$, $\overline{\beta} \in \overline{B}^-$, $b \in B^+$, $\beta \in B^-$ with

$$r(w, \overline{\beta}) \text{ and } r(b, \beta).$$

The relation $r_K(w, \beta)$ implies $w \in \overline{B}$, so without loss of generality take $w \in W$, making K a 4-blocker for B, W.

Lemma 4.4 shows there are vertices $g \in G^+, \gamma \in G^- - \overline{\beta}$ with $r_K(g, \gamma)$. Lemma 7.3 shows $r(g, \gamma)$, since $\neg r_K(g, \beta)$. Augment the compatible pair γ, w. In the resulting graph L, $r_L(b, \beta)$, $r_L(g, \overline{\beta})$, and B has Φ extreme vertices. Taking the colors as B and $\overline{B}$, L is not a blocker.

Case 2. B has $\Phi - 2$ extreme vertices.

In this case the augment involves two nonblack extremes. Note this implies $\neg r(B)$. As before assume K is a 4-blocker for B, W.

Case 2.1. $R^-(B) \cup R^+(B) \not\subseteq W$.

Together with the preceding observation this implies $r(B, G)$ or $r(G, B)$. By possibly changing to the reverse graph assume $r(B, G)$. Hence there are vertices $b \in B^+$, $\gamma \in G^-$, $w \in W^+$, $\omega \in W^-$ with

$$r(b, \gamma) \text{ and } r(w, \omega).$$

Here vertices b, γ come from the assumption $r(B, G)$, and the augment is for γ, w. $\omega \in W$ since K is a 4-blocker.

Next we show there are vertices $w' \in W^+ - w$, $\beta \in B^-$, $g \in G^+$, $\gamma' \in G^- - \gamma$ with

$$r(w', \beta) \text{ and } r(g, \gamma').$$

Vertices satisfying these relations in graph K exist since K is a proper 4-blocker. Lemma 7.3 with $\neg r_K(b, \{\beta, \gamma'\})$ shows the same relations hold in H.

Augment the compatible pair β, w. The resulting graph L has $r_L(g, \gamma')$ and $r_L(w', \omega)$. Lemma 6.1(i) shows L is either a nonblocker, a 1-blocker, or a 4-blocker for B. The last alternative does not hold since B has $\Phi - 2$ extreme vertices in L.

Case 2.2. $R^-(B) \cup R^+(B) \subseteq W$.

There are vertices $b \in B^+$, $\omega \in W^-$, $w \in W^+$, $\beta \in B^-$, $g \in G^+$, $\gamma \in G^-$ with

$$r(b,\omega), r(w,\beta), \text{ and } r(g,\gamma).$$

The first two relations and their four vertices come from Case 2.2. The two gray vertices exist and satisfy $r_K(g,\gamma)$, since K is a proper 4-blocker. We can assume $r(g,\gamma)$, since the augment involves a gray source or sink.

Suppose there are vertices $g' \in G^+ - g$, $\gamma' \in G^- - \gamma$ with $r(g',\gamma')$. Corollary 7.2(iii) shows we are done.

Next suppose there are vertices $g' \in G^+ - g$, $\omega' \in W^-$ with $r(g',\omega')$. This relation plus $r(w,\beta)$ and $r(g,\gamma)$ satisfy the hypothesis of Corollary 7.2(ii), so we are done. We can apply the same argument to the reverse graph if there are vertices $w' \in W^+$, $\gamma' \in G^- - \gamma$ with $r(w',\gamma')$.

Note that $|G^+|+|G^-| \geq 3$ since $|B^+|+|B^-|+|W^+|+|W^-| \leq (\Phi-2)+(\Phi-1) = 2\Phi-3$. By possibly changing to the reverse graph, assume $|G^-| \geq 2$. Every vertex of $G^- - \gamma$ has g as its only source predecessor, since otherwise one of the previous cases applies. Now the same reasoning extends this remark to every vertex of G^- (including γ). If $|G^+| \geq 2$, then one of the previous cases applies (specifically the case for $r(g',\omega')$). Thus $|G^+| = 1$. Since H is not a 3^+-blocker we conclude $r(W)$.

The preceding paragraph shows there are vertices $w' \in W^+$, $\omega' \in W^-$, and $\gamma' \in G^- - \gamma$ with

$$r(w',\omega') \text{ and } r(g,\gamma').$$

(Possibly $\omega' = \omega$.) Augment the compatible pair γ', b. For the resulting graph L, $r_L(g,\gamma)$, $r_L(w',\omega')$, and B has $\Phi - 2$ extreme vertices. Lemma 6.1(i) shows we are done. □

THEOREM 7.8. *Any dag has $OPT = \Phi$ unless it is a blocker, in which case $OPT = \Phi + 1$.*

Proof. If H is a blocker, then $OPT = \Phi + 1$ by Lemmas 4.3 and 2.9. Suppose H is not a blocker. $OPT = \Phi$ unless $\Phi = |V^-| = |V^+| > |B^+| + |B^-|$ for every $B \in \mathcal{P}$, by Lemmas 4.5 and 3.5. For the remaining case, Lemma 6.2 reduces an arbitrarily colored H to a tricolored dag. Lemmas 7.4–7.7 prove the theorem on tricolored dags in the remaining case. If the given dag has an isolated vertex, the theorem still applies, as in the proof of Lemma 5.3. □

8. Algorithms. This section gives algorithms to solve the bipartite strong-connectivity augmentation problem in time $O(n+m)$ and the general augmentation problem in time $O((n+m)\log n)$. Here n and m denote the number of vertices and edges in the input graph, respectively. The main ingredient is an algorithm to find compatible pairs. This is given in section 8.1. The rest of the algorithm consists of procedures that follow the proofs already presented. These procedures are given in sections 8.2–8.5.

We start with two general observations. First recall that the strong component graph of a given digraph can be constructed in linear time, using routines for depth-first search and contraction [5]. Next observe that $OPT \leq \Phi + 1 = O(n)$ new edges get added to the graph. Hence all graphs we work with contain $O(n+m)$ edges.

8.1. Compatible pairs. This section describes several routines to find compatible pairs. A *compatible pair sequence* x_i^-, y_i^+, $i = 1, \ldots, k$, in a dag H has $x_i^- \in V^-$, $y_i^+ \in V^+$ with x_i^-, y_i^+ a compatible pair in the strong component dag of $H + \{(x_j^-, y_j^+) : 1 \leq j < i\}$. Consider two sets $X^- \subseteq V^-$, $Y^+ \subseteq V^+$ that have equal

cardinality and are consistently colored. We give an algorithm to find a compatible pair sequence whose vertices are in $X^- \cup Y^+$ and whose length is $|X^-|-1 = |Y^+|-1$.

The algorithm is based on a procedure ONE-PAIR that finds the next pair in the desired sequence. Suppose we have found compatible pairs (x_j^-, y_j^+), $j = 1, \ldots, i-1$. Let H_i be the strong component graph of $H + \{(x_j^-, y_j^+) : 1 \le j < i\}$. Throughout this discussion let V^-, V^+, X^-, and Y^+ refer to H_i. Thus V^- and V^+ represent the sinks and sources of H_i. X^- and Y^+ represent the vertices of these sets V^- and V^+ that are in the original sets X^- and Y^+. The algorithm maintains a mapping $\sigma : X^- \cup Y^+ \to V^- \cup V^+$ with the property that for each source $z^+ \in Y^+$ and each sink $z^- \in X^-$,

$$r(z^+, \sigma(z^+)) \text{ and } r(\sigma(z^-), z^-).$$

The following procedure finds a compatible pair x^-, y^+ with $x^- \in X^-$, $y^+ \in Y^+$ and augments it. The procedure assumes $|X^-| = |Y^+| \ge 2$. It uses the above mapping σ and updates σ to make it valid in the augmented graph.

PROCEDURE ONE-PAIR.

Step 1. Choose distinct vertices $v_i^- \in X^-$, $i = 1, 2$, having $|\sigma^{-1}(v_i^-)| \le 2$. Similarly choose distinct vertices $w_i^+ \in Y^+$, $i = 1, 2$, having $|\sigma^{-1}(w_i^+)| \le 2$.

Step 2. This step chooses x^- as some v_i and y^+ as some w_j so that after possibly modifying σ,

$$\sigma(x^-) \ne y^+ \text{ and } \sigma(y^+) \ne x^-. \tag{8.1}$$

If for some $i \in \{1, 2\}$, $\sigma(v_i^-) \ne w_i^+$ and $\sigma(w_i^+) \ne v_i^-$, take (x^-, y^+) to be (v_i^-, w_i^+). Otherwise redefine $\sigma(v_1^-) = w_1^+$, $\sigma(w_2^+) = v_2^-$ and take (x^-, y^+) to be (v_1^-, w_2^+).

Step 3. Augment the compatible pair x^-, y^+. Update σ for the new graph H_i as follows: For every $z^+ \in \sigma^{-1}(x^-) - y^+$, change $\sigma(z^+)$ to $\sigma(y^+)$. Similarly for every $z^- \in \sigma^{-1}(y^+) - x^-$, change $\sigma(z^-)$ to $\sigma(x^-)$. □

We now show that ONE-PAIR is correct. First note that the vertices of Step 1 exist. In proof take v_i^-, $i = 1, 2$, to be the two vertices of X^- with smallest value $|\sigma^{-1}(v_i^-)|$. Since the average value of $|\sigma^{-1}(v_i^-)|$, $v_i \in X^-$, is at most $|Y^+|/|X^-| = 1$, we have $|\sigma^{-1}(v_1^-)| + |\sigma^{-1}(v_2^-)| \le 2$. Hence these vertices have the desired property.

It is clear that Step 2 chooses (x^-, y^+) so (8.1) holds. Also note that if Step 2 redefines σ we have $r(w_i^+, v_i^-)$ for $i = 1, 2$, so the defining property of σ is maintained.

Equation (8.1) implies x^-, y^+ is compatible by the defining property of σ. Hence the augment in Step 3 is valid. Finally we show Step 3 updates σ correctly. Recall from Lemma 2.6 that the new graph H_i has sources $V^+ - y^+$ and sinks $V^- - x^-$. Hence Step 3 updates all vertices whose σ value is no longer an extreme vertex. The update is correct by (8.1).

We conclude that ONE-PAIR is correct. This procedure can be repeated to find a compatible pair until $|X^-| = |Y^+| = 1$.

Now we show how to find the desired compatible pair sequence in linear time $O(m + n)$. We use these data structures: The values $\sigma(z)$ are stored in an array. Each vertex $z \in X^- \cup Y^+$ has a list of the vertices in $\sigma^{-1}(z)$. There is a list $SHORT[X^-]$ containing all vertices $x^- \in X^-$ that have $|\sigma^{-1}(x^-)| \le 2$, and a similar list $SHORT[Y^+]$. All lists are doubly linked.

The mapping σ can be constructed initially in time $O(m + n)$ by two depth-first searches: The first labels each vertex of Y^+ with a sink that it can reach. The second labels each vertex of X^- with a source that can reach it. Finally we initialize the $SHORT$ lists.

Now we implement ONE-PAIR in time $O(1)$. (This implies the desired time bound for finding the compatible pair sequence.) Step 1 chooses vertices from the appropriate $SHORT$ list. In Step 2, suppose $\sigma(x^-) = z^+ \in Y^+$. Delete x^- from $\sigma^{-1}(z^+)$; if this makes $|\sigma^{-1}(z^+)| \leq 2$, add z^+ to $SHORT[Y^+]$. Process y^+ similarly. Step 3 changes the appropriate σ-values and adds vertices to the lists $\sigma^{-1}(z)$ for $z = \sigma(x^-), \sigma(y^+)$. This may cause these vertices to be deleted from the $SHORT$ lists. Note that Steps 1 and 2 imply $|\sigma^{-1}(x^-)|,\ |\sigma^{-1}(y^+)| \leq 2$. Hence Step 3 changes $O(1)$ σ values.

We extend this algorithm using the following observations.

LEMMA 8.1. *Take any vertex $b \in B^+$ and let C be the set of all vertices compatible with b. Let $S = R^-(b)$ and $T = R^-(V^+ - b) - B$. If $|S| > 1$, then $C = T$. If $|S| = 1$, then $C = T - S$.* □

LEMMA 8.2. *A pair of vertices x^-, y^+ that is compatible after an augment was compatible before the augment.*

Proof. Suppose x^-, y^+ was incompatible before the augment, i.e., before the augment we have $R^-(y^+) = \{x^-\}$ or $R^+(x^-) = \{y^+\}$. Suppose $R^-(y^+) = \{x^-\}$. The edge added by the augment is directed from a sink other than x^-, since x^- is a sink after the augment. Thus $R^-(y^+) = \{x^-\}$ after the augment. Similarly the condition $R^+(x^-) = \{y^+\}$ is preserved by the augment. □

In the given dag H let U be an arbitrary set of vertices. A *maximal compatible pair sequence for U* is a compatible pair sequence with all its vertices in U that cannot be extended by another pair. We will give an algorithm to find such a sequence. We start with an algorithm C-SEQUENCE that finds a maximal compatible pair sequence for U when the sources and sinks of U are consistently colored.

PROCEDURE C-SEQUENCE(U).

Step 1. Define sets $X^- \subseteq U^-$ and $Y^+ \subseteq U^+$ arbitrarily except for the constraint that each has cardinality $\min\{|U^-|,\ |U^+|\}$. If this cardinality is 0, return. Otherwise apply the algorithm given above to find and augment a compatible pair sequence with $|X^-| - 1$ pairs. (In the resulting graph at least one of the sets U^-, U^+ has cardinality one. Hence there may be one more compatible pair that can be augmented, which is found in Step 2.)

Step 2. Let u be the unique vertex of U^+ (U^-). Apply Lemma 8.1 to find the vertices of U compatible with u. If such a vertex exists, augment the corresponding pair. □

Next we show how to find a maximal compatible pair sequence for U when the extreme vertices of U have two colors, say, B and W. First execute C-SEQUENCE($(U^- \cap B) \cup (U^+ \cap W)$). Then execute C-SEQUENCE($(U^- \cap W) \cup (U^+ \cap B)$). This procedure is correct by Lemma 8.2.

Finally we give an algorithm MAX-SEQUENCE(U) to find a maximal compatible pair sequence for U when the extreme vertices of U have three colors, B, W, and G. Apply the above two-color procedure, taking the two colors as B and $\overline{B}$. Then apply the two-color procedure taking the two colors as W and $\overline{W}$.

LEMMA 8.3. MAX-SEQUENCE(U) *finds a maximal compatible pair sequence for U in a tricolored dag in time $O(n + m)$.*

Proof. MAX-SEQUENCE is correct by Lemma 8.2 and the fact that any augment involves either a black vertex or a white vertex. For the time bound note that Step 2 of C-SEQUENCE finds the sets of Lemma 8.1 by depth-first search. As already noted the graph contains $O(n + m)$ edges. □

We conclude this section by implementing the procedures for section 2. The

implementations for Proposition 2.1, Lemmas 2.2 and 2.3 are straightforward. To implement Lemmas 2.7 and 2.9, use MAX-SEQUENCE. In Lemma 2.9 if the graph is not bicolored, after the augments are executed we determine the first time a color B has $\Phi = |B^+| + |B^-|$. We undo all subsequent augments.

8.2. Bicolored dags.

Lemmas 3.2 and 3.3. To implement both lemmas, we must augment a maximal sequence of witness-compatible pairs, where the witnesses are $b^+ \in B^+$, $b^- \in B^-$ in Lemma 3.2 and $b^+ \in B^+$, $b^- \in B^-$, $w^+ \in W^+$, $w^- \in W^-$ in Lemma 3.3. We give a linear-time procedure for this task.

Define the initial two or four witnesses as in Lemma 3.2 or 3.3. Let X be the set of these initial witnesses. Execute MAX-SEQUENCE($V - X$). Then do the following for the two or four vertices of X. We illustrate for $b^- \in X$ (the other witnesses are similar). Suppose in the current graph $V - X$ contains both a white source y^+ and a black sink y^-. These vertices are incompatible (by Lemma 8.2), so $r(y^+, y^-)$. Hence b^-, y^+ is witness-compatible. Augment this pair and (as in Lemmas 3.2 and 3.3) change b^- to y^-.

This procedure augments a maximal sequence of witness-compatible pairs. In proof let K be the dag at the end of the procedure and let X' be the set of witnesses at the end. Lemma 8.2 shows that in K, $V - X'$ has no pairs that are compatible in the ordinary sense.

We claim that in K, $V - X'$ does not contain both a white source and a black sink. This is obvious if no augment was done for b^-, so suppose an augment was done. After executing MAX-SEQUENCE($V - X$), we have either a unique white source in $W^+ - X$ or a unique black sink in $B^- - X$ (by Lemma 2.5). The augment destroys that source or sink.

The claim implies that K does not have a witness-compatible pair b^-, y^+ for $y^+ \notin X'$. The only other possibility is that b^-, w^+ is witness-compatible. If it were there would be a white source $y^+ \notin X'$ and a black sink $y^- \notin X'$ with $r(y^+, b^-)$ and $r(w^+, y^-)$. But this makes y^-, y^+ compatible, a contradiction.

We conclude our procedure augments a maximal sequence of witness-compatible pairs. It is clear that the time is $O(m + n)$.

Lemma 3.5. The algorithm starts by augmenting a maximal compatible pair sequence until the graph becomes bicolored or some color becomes big. This is done by executing MAX-SEQUENCE(V) and undoing the augments that occur after the first time the termination condition is achieved. If there are two big colors, we reduce to a bicolored dag as shown in the lemma. If there is a unique big color B, augment a maximal compatible sequence where each augment involves a black vertex by executing MAX-SEQUENCE(V) with the colors taken to be B and $\overline{B}$. If some augment creates a second big color, undo any subsequent augments and proceed as above.

8.3. Blockers.

Detecting a blocker. We decide if an arbitrarily colored dag is a blocker in linear time as follows. A set $R^-(X)$ or $R^+(X)$ can be computed in linear time by depth-first search. Hence we can check if a dag is a blocker for a given color B in linear time. (For 3- and 4-blockers, color B determines W.) At most two colors can have Φ extreme vertices, so we can check for a 1-blocker in linear time. At most three colors of a proper 4-blocker can have $\Phi - 1$ extreme vertices (Lemma 4.4 implies $\Phi \geq 3$). So we can check for such a blocker in linear time. V^+ contains vertices of exactly three colors in a 3^+-blocker. Hence we can check for a 3-blocker in linear time. For a 2-blocker, find two vertices $x \in V^+$, $y \in V^-$ with $r(x, y)$. If the dag is a 2-blocker, it

must be a 2-blocker for the color of x or the color of y. So we can check for a 2-blocker in linear time.

Implementing Lemma 4.5 is trivial.

8.4. Core-bicolored dags.

Lemma 5.2. Call a pair $(x^+, x^-) \in \mathcal{B}$ *good* if $|B^+| + |B^-| \leq |V^+| - 2$ or $|B^+| + |B^-| = |V^+| - 1$ and at most one vertex of the pair belongs to G. A pair of $\mathcal{W}$ is good if it satisfies the symmetric condition. The proof of Lemma 5.2 shows if H is not a blocker, then it has good pairs $(x^+, x^-) \in \mathcal{B}$, $(y^+, y^-) \in \mathcal{W}$ with all four vertices distinct. We give an algorithm to find two such pairs. Using these pairs we recolor vertices to reduce to the bicolored case, as shown in the lemma.

First note that we can find a good pair in $\mathcal{B}$ by depth-first search: If $|B^+|+|B^-| \leq |V^+|-2$, we do a depth-first search from $(B \cup G)^+$. If $|B^+| + |B^-| = |V^+| - 1$, we do a depth-first search from B^+ and a second depth-first search from G^+.

Start by finding a good pair $(x^+, x^-) \in \mathcal{B}$. Then try to find a good pair of $\mathcal{W}$ in the graph $H - \{x^+, x^-\}$. If such a pair exists, we are done. Otherwise execute the case below that applies.

Case 1. $H - x^+$ does not contain a good pair of $\mathcal{W}$.

This implies x^+ is in every good pair of $\mathcal{W}$. Let (x^+, y_1^-) be a good pair of $\mathcal{W}$. If it is the only good pair of $\mathcal{W}$, then use it with a good pair of $\mathcal{B}$ in $H - \{x^+, y^-\}$ as the desired pairs. Otherwise let (x^+, y_2^-) be a second good pair of $\mathcal{W}$. There is a good pair of $\mathcal{B}$ in $H - \{x^+\}$, say, (z^+, z^-). Use (z^+, z^-) with (x^+, y_i^-), where $i \in \{1, 2\}$ is chosen so $y_i^- \neq z^-$, as the desired pairs.

Case 2. Both $H - x^+$ and $H - x^-$ contain a good pair of $\mathcal{W}$, but $H - \{x^+, x^-\}$ does not.

This implies we can find good pairs (x^+, y^-) and (y^+, x^-) of $\mathcal{W}$, where $y^- \neq x^-$ and $y^+ \neq x^+$. $x^+, x^- \in G$ since $(x^+, x^-) \in \mathcal{B}$. If $y^- \in G$, then (x^+, y^-) is good for $\mathcal{B}$, so take (x^+, y^-) and (y^+, x^-) as the desired pairs. Now without loss of generality assume that both y^+ and y^- are white. There is a good pair of $\mathcal{B}$ in either $H - x^+$ or $H - x^-$. Find such a pair and use it with either (x^+, y^-) or (y^+, x^-) to form the desired pairs.

THEOREM 8.4. *The bipartite strong-connectivity augmentation problem and the cable-framework rigidity problem can be solved in linear time.* □

8.5. General algorithm.

Merging colors. Lemma 6.2 is straightforward to implement in time $O(n+m)$. The algorithm finds a permissible merge into three colors that gives a nonblocker after trying at most five merges.

The implementation of Lemma 6.3 is similar to Case 1 of Lemma 5.2 in the previous section.

Augmentation algorithm. We are given a tricolored dag H that is not a blocker. We implement the proof of section 7 to solve the strong connectivity augmentation problem on H. Recall that the proof orders the blocker types as 2, 3, 4, 1. The relation $\prec$ corresponds to precedence in this list, e.g., $2 \prec 3, 4, 1$. Similarly for $\preceq, \succ, \succeq$.

For $i = 2, 3, 4$ define an *i-witness set* to be a set of vertices that satisfies one of the conditions of Lemma 7.1 or Lemma 6.1(i) that preclude H from being a blocker of type $\preceq i$. For instance the six vertices of Lemma 7.1(iv) form a 3-witness set, as well as a 2-witness set. Vertices $b^+ \in B^+$, $b^- \in B^-$, $x^+ \in \overline{B}^+$, $x^- \in \overline{B}^-$ with $r(b^+, b^-)$ and $r(x^+, x^-)$ form a 2-witness set, since they satisfy the condition of Lemma 7.1(i). A 4-witness set differs from the others because it has an additional condition: There

is a distinguished color B that has $\Phi - 2$ extreme vertices. For instance, the above 3-witness set is a 4-witness set in a dag where B has $\Phi - 2$ extremes.

Each of Lemmas 7.4–7.7 is easily implemented in linear time. Lemma 7.4 reduces the graph to a bicolored dag. Lemmas 7.5–7.7 each start with a dag that has an augment giving an i-blocker ($i = 2, 3, 4$) and execute a different augment that is guaranteed not to give a blocker of type $\preceq i$. This guarantee holds because the lemma (and its corresponding algorithm) produces an i-witness set. (Lemma 7.7 produces a 4-witness set in a graph where B has $\Phi - 2$ extreme vertices.)

These observations show we can execute one augment in linear time. This implies an algorithm for the augmentation problem that runs in time $O((n + m)\Phi)$.

Now we give an algorithm with the improved time bound $O((n + m) \log \Phi)$. The algorithm consists of four stages: an initial stage and a stage for i-blockers, $i = 2, 3, 4$. We refer to the stage for i-blockers as *stage* i. The algorithm starts in the initial stage. After that it executes between zero and three blocker stages, always progressing from a stage i to a higher-numbered stage.

Each stage has as input a tricolored dag that is not a blocker. For the initial stage this is the given dag H. For subsequent stages it is an augmented version of H. Stage i is also given an i-witness set.

Each stage begins by executing a sequence of augments that end with a dag K, such that the following properties hold.

(i) Each augment decreases Φ by 1.
(ii) K is not a blocker.
(iii) No augment involves a vertex in the given witness set. (This set is $\emptyset$ in the initial stage.)
(iv) In stage 4 each augment involves a vertex of color B (so B always has $\Phi - 2$ extreme vertices).
(v) K is either (a) rooted, (b) bicolored, (c) has $\Phi = |B^+| + |B^-|$ for some color B, (d) has $\Phi = O(1)$, or (e) an augment for K has been found whose execution gives a blocker.

In cases (v(a))–(v(d)) complete the solution to the augmentation problem in time $O(n + m)$ using the procedure for Lemmas 2.3, 3.4, 4.5, or the above $O((n + m)\Phi)$ algorithm, respectively. (Note K may be strongly connected, in which case it satisfies (v(b)).)

Suppose case (v(e)) holds. If some color has no source or no sink, execute the procedure of Lemma 6.3 to get a bicolored dag and complete the solution as in the previous paragraph. Otherwise execute the procedure of Lemmas 7.4–7.7 to get a dag L that is not a blocker. (For example, suppose K has an augment giving a 2-blocker. Lemma 7.5 may give an augment that produces a nonblocker or perhaps a 4-blocker. In the first case we take the nonblocker as L. In the second case we execute the procedure for Lemma 7.7. Eventually we get a nonblocker L.) If L satisfies any of conditions (v(a))–(v(c)), complete the solution as in the previous paragraph. Otherwise the last lemma whose procedure is executed gives a j-witness set for L, where $j \in \{2, 3, 4\}$. Proceed to stage j. (Note that if Lemma 7.4 for 1-blockers is executed, it gives a graph satisfying (v(b)), so the solution gets completed.)

This algorithm executes each of the four stages at most once. In proof suppose stage i ends with (v(e)) satisfied. The augment of (v(e)) gives a blocker of type $\succ i$. This follows since the i-witness set is preserved (by (iii)–(iv)). Hence the procedure of Lemmas 7.4–7.7 gives a nonblocker L that, if not bicolored, has a j-witness set for some $j \succ i$. In this case the algorithm advances to stage j, so no stage gets repeated.

Now we show how to implement a stage. We need only describe the procedure that executes the sequence of augments satisfying (i)–(v).

Let X be the given set of i-witnesses ($X = \emptyset$ for the initial stage). The initial stage and stages 2 and 3 begin by executing MAX-SEQUENCE($V - X$). Stage 4 executes MAX-SEQUENCE($V - X$) in the dag with colors taken to be B and $\overline{B}$ (for the distinguished color B).

Let S be the sequence of augments that get executed in MAX-SEQUENCE. Undo these augments. Take the final dag K to be the result of executing augments of S, in order, until one of conditions (v(a))–(v(d)) holds, or executing the next augment in S would give a blocker (so (v(e)) holds).

Correctness of this procedure depends on two facts. First note that the final sequence of augments satisfies (i). This holds since an augment that does not decrease Φ is executed in a dag that has a color B with Φ extreme vertices, so (v(c)) holds.

The second fact is that (v) is satisfied when the entire sequence S gets executed to produce K. We show that in this case (v(d)) holds, $\Phi = O(1)$. Note that any i-witness set has $O(1)$ vertices (specifically it has at most 6 vertices). Hence the following lemma is sufficient.

LEMMA 8.5. *Let H be a tricolored graph with $X \subseteq V$.*

(i) *If* MAX-SEQUENCE($V - X$) *results in a dag K with $\Phi = |V^+| = |V^-|$, then K has $\Phi \leq |X| + 3$.*

(ii) *Suppose executing* MAX-SEQUENCE($V - X$) *in the dag with colors taken to be B and $\overline{B}$ results in a dag K that (using the original colors) has $\Phi = |V^+| = |V^-| = |B^+| + |B^-| + 2$. Then K has $\Phi \leq |X| + 4$.*

Proof. (i) Consider K. If every color B has $|(B - X)^+| \leq 1$, then $\Phi = |V^+| \leq |X| + |V^+ - X| \leq |X| + 3$. Hence assume some color B has $|(B - X)^+| > 1$. This implies $|(\overline{B} - X)^-| \leq 1$ (Lemma 2.5). If $|(B - X)^-| \leq 1$, then $\Phi = |V^-| \leq |X| + |V^- - X| \leq |X| + 2$. So assume $|(B - X)^-| > 1$. This implies $|(\overline{B} - X)^+| \leq 1$ (Lemma 2.5). Hence $\overline{B}$ contains at most $|X| + 2$ extreme vertices. But $\overline{B}$ contains at least Φ extreme vertices (since B contains at most Φ extremes). We have deduced $\Phi \leq |X| + 2$.

(ii) If $|(B - X)^+|, |(B - X)^-| \leq 1$, then $\Phi = |B^+| + |B^-| + 2 \leq |X| + 4$. So without loss of generality assume $|(B - X)^+| > 1$. The rest of the argument proceeds as in part (i). □

The efficiency of a stage depends on how fast we can find the subsequence of augments of S that get executed. We start by finding the first augment in S that results in a dag satisfying any of the conditions (v(a))–(v(c)). This is easy to do in linear time. Let S' be the sequence S up to and including the augment just found. Now we must find the first augment in S' that produces a blocker. This can be done by a binary search of S', where each probe tests if the corresponding dag is a blocker.

To see that this binary search is correct, note that the first augment in S' that produces a blocker changes a dag with $OPT = \Phi$ into a dag with $OPT = \Phi + 1$. Every subsequent dag has $OPT = \Phi + 1$ (since each augment decreases Φ by 1, and an augment decreases OPT by at most 1). Hence every subsequent dag is a blocker.

Each probe of the binary search takes time $O(n + m)$ (section 8.3). Thus the time for the binary search is $O((n + m) \log \Phi)$.

THEOREM 8.6. *The problem of strong-connectivity augmentation with partition constraints can be solved in time $O((n + m) \log n)$ (the value of OPT can be found in time $O(n + m)$).* □

9. Related problems. The weighted version of bipartite strong-connectivity augmentation is NP-hard, by reduction from the bipartite Hamiltonian cycle problem.

[3] solves the problem of making an undirected bipartite graph k-edge-connected while preserving bipartiteness, i.e., bipartite k-edge-connectivity augmentation. This paper has solved the $k = 1$ case of bipartite directed k-edge-connectivity augmentation. We believe this directed problem is harder than the undirected problem of [3]. As evidence first recall that the directed problem for $k = 1$ already has a richer set of blockers than the undirected problem for arbitrary k. Also for $k = 1$ the directed problem can have $OPT > \Phi$ for graphs with arbitrarily large values of OPT. In contrast [3] shows that for the undirected problem, $OPT \geq 2k + 1$ implies $OPT = \Phi$. Finally [8] shows that in the directed problem with arbitrary k, OPT takes on values between Φ and $\Phi + k$ (for an appropriate definition of Φ). In contrast [3] shows any undirected bipartite graph has OPT equal to Φ or $\Phi + 1$.

We close by noting that if our operation is not adding new cables but replacing existing cables by rods, the rigidity problem can be solved by the Lucchesi–Younger Theorem [7].

REFERENCES

[1] J.A. Baglivo and J.E. Graver, *Incidence and Symmetry in Design and Architecture*, Cambridge University Press, Cambridge, UK, 1983.

[2] J. Bang-Jensen and T. Jordán, *Edge-connectivity augmentation preserving simplicity*, SIAM J. Discrete Math., 11 (1998), pp. 603–623.

[3] J. Bang-Jensen, H.N. Gabow, T. Jordán, and Z. Szigeti, *Edge-connectivity augmentation with partition constraints*, SIAM J. Discrete Math., 12 (1999), pp. 160–207.

[4] E.D. Bolker and H. Crapo, *How to brace a one-story building*, Environ. Plan. B, 4 (1977), pp. 125–152.

[5] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York, 1990.

[6] K.P. Eswaran and R.E. Tarjan, *Augmentation problems*, SIAM J. Comput., 5 (1976), pp. 653–665.

[7] A. Frank, *Connectivity augmentation problems in network design*, in Mathematical Programming: State of the Art 1994, J.R. Birge and K.G. Murty, eds., University of Michigan, Ann Arbor, MI, 1994, pp. 34–63.

[8] H. N. Gabow and T. Jordán, *Bipartition constrained edge-splitting in directed graphs*, in Proceedings of the 1st Japanese–Hungarian Symp. on Discrete Math. and its Applications, Kyoto, Japan, 1999, pp. 225–232.

[9] M. Grötschel, C.L. Monma, and M. Stoer, *Design of survivable networks*, in Handbook in Operations Research and Management Science, M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, eds., Network Models 7, North-Holland, Amsterdam, The Netherlands, 1994, pp. 617–672.

[10] T-S. Hsu and M-Y. Kao, *Optimal augmentation for bipartite componentwise biconnectivity in linear time*, in Algorithms and Computation (Proc. ISAAC '96), Lecture Notes in Comput. Sci. 1178, Springer-Verlag, New York, 1996, pp. 213–222.

[11] T. Ishii, H. Nagamochi, and T. Ibaraki, *Optimal augmentation to make a graph k-edge-connected and triconnected*, in Proceedings of the 9th Annual ACM-SIAM Symp. on Discrete Algorithms, ACM, New York, 1998, pp. 280–289.

[12] T. Jordán, *Two NP-Complete Augmentation Problems*, Preprint 8, Department of Mathematics and Computer Science, Odense University, 1997.

[13] G. Kant and H.L. Bodlaender, *Planar graph augmentation problems*, in Algorithms and Data Structures (Proc. WADS '91), Lecture Notes in Comput. Sci. 519, Springer-Verlag, New York, 1991, pp. 286–298.

[14] H. Nagamochi and P. Eades, *Edge-splitting and edge-connectivity augmentation in planar graphs*, in Integer Programming and Combinatorial Optimization, Lecture Notes in Comput. Sci. 1412, Springer-Verlag, New York, 1998, pp. 96–111.

[15] A. Recski, *Matroid Theory and its Applications in Electric Network Theory and in Statics*, Akadémiai Kiadó, Budapest, 1989.

[16] M. Stoer, *Design of Survivable Networks*, Lecture Notes in Math. 1531, Springer-Verlag, New York, 1992.

SIAM J. COMPUT.
Vol. 30, No. 2, pp. 681–687

ON A CONJECTURE BY COFFMAN, FLATTO, AND WRIGHT ON STOCHASTIC MACHINE MINIMIZATION*

NAH-OAK SONG† AND DEMOSTHENIS TENEKETZIS‡

Abstract. We investigate a conjecture stated by Coffman, Flatto, and Wright within the context of a stochastic machine minimization problem with a hard deadline. We prove that the conjecture is true.

Key words. stochastic allocation, hard deadlines, machine minimization, Bellman equation, time thresholds

AMS subject classifications. 93E20, 93E03, 90C39

PII. S0097539797324643

1. Introduction.

Problems of optimal stochastic allocation of machines under waiting-time constraints have recently received considerable attention, as they are important in the design of computer and communication networks and in stochastic real-time scheduling problems (see [1], [2], [3]). From a theoretical point of view these problems are complementary to makespan minimization problems (see, for example, [4], [6], [7], [8]).

The general class of problems of optimal stochastic allocation of machines under waiting-time constraints (also known as stochastic machine minimization problems) can be formulated as follows: There are N jobs with processing times $T_1, T_2, \ldots, T_N$, deterministic or random with known distributions. Job waiting times are bounded by a time W that is independent of $T_1, T_2, \ldots, T_N$, and may be deterministic or random with a known distribution. There is an unlimited number of machines (processors) initially available to process these jobs. At time 0, a timer is started with initial value W and job scheduling begins. When the timer expires, all jobs not running at that time and still waiting to be processed are assigned to available machines. The objective is to determine, within the class of nonpreemptive policies, a policy that minimizes the expected cost, with cost defined as the number of distinct machines used throughout the schedule.

Several cases within the above class of problems have been considered so far in the literature [1], [2], [3]. A problem of practical importance is the hard deadline case where W is a constant. A partial analysis of this problem has been presented in [1], where the structure of an optimal policy was proved under a conjecture that remains so far an open problem. In this paper we prove that the conjecture stated in [1] is true; consequently, the structure of the optimal policy proposed in [1] is correct.

The paper is organized as follows: the conjecture stated in [1] is precisely formulated in section 2; an outline of the conjecture's proof is given in section 3 and

*Received by the editors July 23, 1997; accepted for publication (in revised form) January 5, 2000; published electronically June 27, 2000. This research was supported in part by NSF grant NCR-9204419, AFOSR grant F49620-96-1-0028, ARO grant DAAH04-96-1-0377, and NSF grant ECS-9979347.

http://www.siam.org/journals/sicomp/30-2/32464.html

†Samsung Electronics Co., Ltd, 11th Floor, Samsung Plaza Bldg., 263, Seohyeon-Dong, Bundang-Gu, Sungnam-Si, Kyungki-Do, Korea 463-050 (sno@telecom.samsung.co.kr). This author's work was also supported in part by a Barbour Scholarship from the University of Michigan.

‡Department of Electrical Engineering and Computer Science, University of Michigan, 1301 Beal Avenue, Ann Arbor, MI 48109-2122 (teneketzis@eecs.umich.edu).

the technical details of the proof appear in section 4. Discussion of the results and suggestions for further research appear in section 5.

2. The conjecture. Coffman, Flatto, and Wright [1] considered a stochastic machine minimization problem (described in section 1) where the job processing times $T_1, T_2, \ldots, T_N$ are independent samples of an exponentially distributed random variable T with $E(T) = 1$, and the job waiting times are bounded by a constant W.

For this particular stochastic machine minimization problem it is possible to define a Markov process on the set of states (n, k, s), where n is the number of unfinished jobs, k is the number of jobs currently assigned to processors, and s is the time remaining on the timer (see [1]). Let $V(n, k, s)$ denote the expected cost incurred by an optimal allocation policy when the initial state is (n, k, s), and define

$$V(n, s) = \min_{1 \le k \le n} V(n, k, s) \tag{1}$$

as the expected cost incurred by an optimal allocation policy when there are n jobs to be processed with s units of time remaining on the timer, and there is an unlimited number of machines initially available. $V(n, k, s)$ can be computed by the Bellman equation,

(2)

$$V(n, k, s) = \inf_{0 \le t \le s} \left\{ e^{-k(s-t)} V(n, k+1, t) + \int_0^{s-t} k e^{-ku} V(n-1, k, s-u) du \right\}, \; k < n,$$

$$V(n, n, s) = n. \tag{3}$$

Coffman, Flatto, and Wright [1] proved that $V(n, k, s)$ has the following properties.

LEMMA 2.1. (i) $V(n, k, 0) = n$, $V(n, k, \infty) = k$. (ii) $V(n, k, s)$ *is nondecreasing in* k *for fixed* n *and* s, *strictly increasing in* n *for fixed* k *and* s, *and strictly decreasing in* s *for fixed* n *and* $k < n$.

Furthermore, the following assertion was made in [1].

CONJECTURE. *For each* $n > 1$, *there exist nonnegative numbers* l_{nk}, $1 \le k \le n-1$, *such that*

$$V(n, k+1, s) > V(n, k, s) \quad \text{for } s > l_{nk}, \tag{4}$$

$$V(n, k+1, s) = V(n, k, s) \quad \text{for } s \le l_{nk}, \tag{5}$$

$$l_{n(n-1)} \le l_{n(n-2)} \le \cdots \le l_{n1}. \tag{6}$$

Based on Lemma 2.1 and the above conjecture, Coffman, Flatto, and Wright proved the following result in [1].

THEOREM 2.2. *For all* k, $1 \le k \le n-2$,

$$l_{(n-1)k} < l_{nk}. \tag{7}$$

The above theorem and the conjecture imply the following structure of an optimal policy.

THEOREM 2.3 (see [1, Theorem 5.3]). *Let the initial state have $n > 1$ jobs to be scheduled, an unlimited number of machines available, and a clock time s such that $l_{nk} < s \leq l_{n(k-1)}$ for some $1 \leq k \leq n-1$. Then assign jobs to k machines and start running jobs in state (n, k, s). Continue running jobs in any given state until one of the following three events occurs:* (1) *The clock expires; in this case assign the remaining $n - k \geq 0$ waiting jobs to new machines.* (2) *A machine completes a job in some state $(n', k', s'), k' < n'$; in this case replenish the machine and continue in state $(n' - 1, k', s')$.* (3) *A state $(n', k', s'), k' < n'$, is reached in which the remaining time has reduced to $s' = l_{n'k'}$; in this case assign a waiting job to a new machine and continue in state $(n', k' + 1, s')$.*

In the remainder of this paper, we prove that the above stated conjecture is true, thus completing the proof of the optimality of the policy described in Theorem 2.3. We proceed as follows: First we briefly outline the main ideas of the proof and then we present all the technical details.

3. Outline of the proof of the conjecture. We establish, via the Bellman equation, the existence of "time thresholds" $l_{nk}, n = 1, 2, \ldots$ and $k = 1, 2, \ldots, n-1$ that have the following features:

(F1) When the initial state is $(n, k, s), s > l_{nk}$, then along sample path realizations where there are no job completions it is optimal not to assign any new (i.e., previously unused) machines to jobs waiting to be processed as long as the time remaining in the timer is strictly larger than l_{nk}. (Along the same sample path realizations it is optimal to assign a new machine to a job waiting to be processed when the time remaining in the timer is equal to l_{nk}.)

(F2) If the initial state is (n, k, t) and $t \leq l_{nk}$, then it is optimal to assign right away at least one new machine to a job waiting to be processed.

(F3) For any $n, n = 1, 2, \ldots$ and $k = 1, 2, \ldots, n-1$ the "time thresholds" l_{nk} are ordered as follows:

$$l_{n(n-1)} \leq l_{n(n-2)} \leq \cdots \leq l_{n1}.$$

To establish (F1) we use the Bellman equation and consider an initial state (n, k, s) with the following characteristic: The earliest time l_{nks} (according to the Bellman equation) at which it is optimal to add a new machine along sample path realizations where there are no job completions is such that $l_{nks} < s$. We prove that for all states (n, k, s') such that $s' > l_{nks}$ we have $l_{nks} = l_{nks'} := l_{nk}$. This result and the definition of l_{nks} (see (8)–(9)) lead to the proof of the first part of the conjecture (see (4)).

To establish (F2) we consider an instance where the initial state is $(n, k, s), s > l_{nk}$, and an optimal allocation policy satisfying the Bellman equation is used. We show that along sample path realizations where there are no job completions until t units of time remain on the timer ($t < l_{nk}$ by assumption) it is optimal to have at least $k + 1$ jobs under processing at t. This implies that if the initial state is (n, k, t) and an optimal allocation policy (satisfying the Bellman equation) is used, at least one new machine must be allocated to a job waiting to be processed at t. This feature together with a property of the cost function $V(n, k, s)$, described by Lemma 2.1(ii), leads to the proof of the second part of the conjecture (Eq. (5)).

To establish (F3) we use the first and second parts of the conjecture (already proved), a property of the cost function $V(n, k, s)$ described by Lemma 2.1 (ii), the Bellman equation, and a contradiction argument. Feature (F3) of the "time thresholds" l_{nk} describes the last part of the conjecture (see (6)).

Thus, the validity of the conjecture is established.

4. Proof of the conjecture. Consider the Bellman equation and for each state (n,k,s) define

$$A^s_{n,k} := \left\{ l : l = \arg \inf_{0\le t\le s} \left[e^{-k(s-t)} V(n,k+1,t) + \int_0^{s-t} k e^{-ku} V(n-1,k,s-u)du \right] \right\} \tag{8}$$

and

$$l_{nks} := \max \left\{ l : l \in A^s_{n,k} \right\}. \tag{9}$$

Let π^* denote the allocation policy that satisfies the Bellman equation and has the following characteristic: Along sample paths that originate at any state (n,k,s), with $n > k$, $s > l_{nks}$, and have no job completions until l_{nks} units of time remain on the timer, π^* adds a new machine at that point.

Proof of (4). Consider an initial state (n,k,s) such that $s > l_{nks}$. We prove that for all $s' > l_{nks}$ we have

$$l_{nks} = l_{nks'} := l_{nk}. \tag{10}$$

This fact together with the definition of l_{nks} leads to the proof of the first part of the conjecture, namely, (4). To prove (10) we proceed in two steps.

Step (i). Take s' such that $l_{nks} < s' < s$. We prove that $A^s_{n,k} = A^{s'}_{n,k}$. Pick $l \in A^{s'}_{n,k}$; then

$$V(n,k,s') = e^{-k(s'-l)} V(n,k+1,l) + \int_0^{s'-l} k e^{-ku} V(n-1,k,s'-u)du. \tag{11}$$

Furthermore, since $s > s' > l_{nks}$,

$$V(n,k,s) = e^{-k(s-s')} V(n,k,s') + \int_0^{s-s'} k e^{-ku} V(n-1,k,s-u)du. \tag{12}$$

Substituting (11) into (12), we get

$$V(n,k,s) = e^{-k(s-s')} \left[e^{-k(s'-l)} V(n,k+1,l) + \int_0^{s'-l} k e^{-ku} V(n-1,k,s'-u)du \right] + \int_0^{s-s'} k e^{-ku} V(n-1,k,s-u)du. \tag{13}$$

Letting

$$v = s - s' + u \tag{14}$$

in the second term of (13), we obtain

$$V(n,k,s) = e^{-k(s-l)} V(n,k+1,l) + \int_0^{s-l} k e^{-ku} V(n-1,k,s-u)du. \tag{15}$$

Equation (15) proves that $l \in A^s_{n,k}$. Hence,

$$A^{s'}_{n,k} \subset A^s_{n,k}. \tag{16}$$

Next, take $l \in A^s_{n,k}$; suppose that $l \notin A^{s'}_{n,k}$. Then, since by assumption $s > s' > l_{nks} \geq l$, it follows that

$$\begin{aligned} V(n,k,s) &= e^{-k(s-s')}V(n,k,s') + \int_0^{s-s'} ke^{-ku}V(n-1,k,s-u)du \\ &< e^{-k(s-s')}\left[e^{-k(s'-l)}V(n,k+1,l) + \int_0^{s'-l} ke^{-ku}V(n-1,k,s'-u)du\right] \\ &\quad + \int_0^{s-s'} ke^{-ku}V(n-1,k,s-u)du \\ &= e^{-k(s-l)}V(n,k+1,l) + \int_0^{s-l} ke^{-ku}V(n-1,k,s-u)du; \end{aligned} \tag{17}$$

the inequality in (17) results because $l \notin A^{s'}_{n,k}$. According to (17), $l \notin A^s_{n,k}$ and this is a contradiction. Hence, $l \in A^{s'}_{n,k}$; therefore,

$$A^s_{n,k} \subset A^{s'}_{n,k}. \tag{18}$$

From (16) and (18), we conclude that

$$A^s_{n,k} = A^{s'}_{n,k} \qquad \text{for all } s' \text{ such that } l_{nks} < s' < s. \tag{19}$$

Step (ii). Consider s' such that $\ell_{nks} < s < s'$. We prove that there is no $\hat{\ell} \in A^{s'}_{n,k}$ such that $\hat{\ell} \geq s$. The proof is by contradiction.

Suppose there exists $\hat{\ell} \in A^{s'}_{n,k}$ such that $\hat{\ell} \geq s$; then $\ell_{nks'} \geq s$. Start at (n,k,s') and use the optimal allocation policy π^*. Let Ω' be the set of sample paths that start at (n,k,s') and along which there are no job completions until s units of time remain on the timer. Since $\ell_{nks'} \geq s$, the expected cost incurred by the optimal policy π^* along Ω' is $V(n,k+\hat{k},s)$, where $\hat{k} \geq 1$. Furthermore, since $\hat{k}$ machines are added by the optimal policy π^* along Ω', we conclude that when there are n unfinished jobs and s units of time remain on the timer it is optimal to use $k+\hat{k}$ machines ($\hat{k} \geq 1$). That is,

$$V(n,s) = V(n,k+\hat{k},s). \tag{20}$$

On the other hand, since $s > \ell_{nks}$, (1), Lemma 2.1(ii), the Bellman equation, and the definition of ℓ_{nks} imply that

$$V(n,s) < V(n,k+1,s) \leq V(n,k+\hat{k},s), \tag{21}$$

which, in turn, implies that when there are n unfinished jobs and s units of time remain on the timer it is optimal to use less than $k+1$ machines. Thus, under the assumption that there exists $\hat{\ell} \in A^{s'}_{n,k}$ such that $\hat{\ell} \geq s$, a contradiction (see (20) and (21)) is reached. Consequently, every $\hat{\ell} \in A^{s'}_{n,k}$ satisfies $\hat{\ell} < s$. Furthermore, since $s' > s > l_{nks'}$, by arguments similar to those of step (i), we conclude that

$$A^{s'}_{n,k} = A^s_{n,k} \qquad \text{for all } s' \text{ such that } l_{nks} < s < s'. \tag{22}$$

Hence, for all states $(n,k,s),(n,k,s')$ such that $s > l_{nks}, s' > l_{nks}$, we have

$$l_{nks} = l_{nks'} := l_{nk}. \tag{23}$$

From (2), (3), (8), (9), (19), (22), and (23), we conclude that there exists l_{nk} such that

$$V(n,k,s) < V(n,k+1,s) \qquad \text{for all } s > l_{nk}. \tag{24}$$

Proof of (5). Consider any $t \leq l_{nk}$. We want to prove that for such a t,

$$V(n,k,t) = V(n,k+1,t). \tag{25}$$

To do this we formalize the idea outlining the proof of (F2) in section 3. Take (n,k,s), $s > l_{nk}$, as the initial state and use the allocation policy π^*. Let $\overline{\Omega}$ be the set of sample paths starting at (n,k,s) along which there are no job completions until t time units remain on the timer. Then, because of (2), (8), (9), and the definition of policy π^*, the expected cost incurred by the optimal policy π^* along $\overline{\Omega}$ is $V(n,k+r,s)$ for some $r \geq 1$. Moreover, since r machines are added by the optimal policy π^* along $\overline{\Omega}$, it follows that when there are n unfinished jobs and t units of time remain on the timer it is optimal to use $k+r$ machines ($r \geq 1$). Hence,

$$V(n,k+r,t) = V(n,t) \leq V(n,k,t). \tag{26}$$

On the other hand, by Lemma 2.1(ii)

$$V(n,k+r,t) \geq V(n,k,t). \tag{27}$$

From (26) and (27) we conclude that

$$V(n,k,t) = V(n,k+r,t). \tag{28}$$

From (28) and Lemma 2.1(ii) we obtain

$$V(n,k,t) = V(n,k+1,t) \qquad \text{for } t \leq l_{nk}. \tag{29}$$

Proof of (6). We use (4) and (5) and formalize the idea outlining the proof of (F3) in section 3. Suppose that $l_{nk} > l_{n(k-1)}$. Consider t such that $l_{nk} > t > l_{n(k-1)}$ and assume that there are n jobs to be processed at t. Then $t < l_{nk}$ and (5) imply that

$$V(n,k,t) = V(n,k+1,t). \tag{30}$$

From (1), (30), Lemma 2.1(ii), and the Bellman equation it follows that it is optimal to use at least $k+1$ machines when there are n jobs to be processed with t units of time remaining on the timer. On the other hand, $t > l_{n(k-1)}$ and (4) imply that

$$V(n,k,t) > V(n,k-1,t). \tag{31}$$

From (1), (31), Lemma 2.1(ii), and the Bellman equation we conclude that it is optimal to use less than k machines when there are n jobs to be processed with t units of time left on the timer. Thus, under the assumption $l_{nk} > l_{n(k-1)}$ we reach a contradiction. Consequently, $l_{nk} \leq l_{n(k-1)}$. By arguments similar to the above we obtain the remaining inequalities in (6).

The proof of the conjecture is now complete. □

5. Discussion. The proof of the conjecture stated in [1] and described in section 2 completes the proof of optimality of the policy described in Theorem 2.3. Theorem 2.3 describes the nature of an optimal policy for the stochastic machine minimization problem formulated in [1] and described in sections 1 and 2 of this paper. According to this optimal policy allocation decisions are made at job completions, when the timer expires, or when the "time thresholds" l_{nk} are reached. These "time thresholds" can be computed for all $n, k, k = 1, 2, \ldots, n-1$ by (8) and (9). The computation of l_{nk} is a challenging and formidable task that will not be further pursued in this paper.

The stochastic machine minimization problems investigated by Coffman, Flatto, and Wright in [1], [2], [3] consider only the scheduling of jobs (tasks, projects) that are initially available (i.e., they are available at time 0) in the system. Stochastic machine minimization problems with arrivals are interesting extensions of the problems considered in [1], [2], [3], as they arise in wireless communication networks and in automated target recognition systems (see [5]). A stochastic machine minimization problem with Poisson arrivals, exponential service times, and hard deadlines has been investigated in [5].

A class of interesting technical questions is to determine whether the structure of the optimal policies described in [1], [2], [3] remains unaltered when arrivals are included in the problem formulation.

Acknowledgments. The authors are grateful to the anonymous reviewers, whose comments helped to significantly improve the presentation of the results of this paper.

REFERENCES

[1] E. G. Coffman, Jr., L. Flatto, and P. E. Wright, *Optimal stochastic allocation of machines under waiting-time constraints*, SIAM J. Comput., 22 (1993), pp. 332–348.

[2] E. G. Coffman, Jr., L. Flatto, and P. E. Wright, *Stochastic machine minimization with constant service times*, Math. Oper. Res., 18 (1993), pp. 300–316.

[3] E. G. Coffman, Jr., L. Flatto, B. Poonen, and P. E. Wright, *The processor minimization problem with independent waiting-time constraints*, Theoret. Comput. Sci., 125 (1994), pp. 3–16.

[4] M. Pinedo and L. Schrage, *Stochastic shop scheduling: A survey*, in Deterministic and Stochastic Scheduling, M. Dempster, J. K. Lenstra, and A. Rinooy-Kan, eds., D. Reidel, Dordrecht, The Netherlands, 1982.

[5] N. Song and D. Teneketzis, *The Stochastic Machine Minimization Problem with Hard Deadlines and Arrivals*, Control Group Report CGR-96-13, Department of EECS, University of Michigan, 1996 (under revision).

[6] R. R. Weber, *Scheduling jobs with stochastic processing requirements on parallel machines to minimize makespan or flowtime*, J. Appl. Probab., 19 (1982), pp. 167–182.

[7] G. Weiss, *Multiserver stochastic scheduling*, in Deterministic and Stochastic Scheduling, M. Dempster, J. K. Lenstra, and A. Rinooy-Kan, eds., D. Reidel, Dordrecht, The Netherlands, 1982.

[8] G. Weiss and M. Pinedo, *Scheduling tasks with exponential services times on nonidentical processors to minimize various cost functions*, J. Appl. Probab., 17 (1980), pp. 187–202.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 689–728

ALL OF US ARE SMARTER THAN ANY OF US: NONDETERMINISTIC WAIT-FREE HIERARCHIES ARE NOT ROBUST*

WAI-KAU LO† AND VASSOS HADZILACOS†

Abstract. A wait-free hierarchy [*ACM Transactions on Programming Languages and Systems*, 11 (1991), pp. 124–149; *Proceedings of the* 12th *ACM Symposium on Principles of Distributed Computing*, 1993, pp. 145–158] classifies object types on the basis of their strength in supporting wait-free implementations of other types. Such a hierarchy is robust if it is impossible to implement objects of types that it classifies as "strong" by combining objects of types that it classifies as "weak." We prove that if nondeterministic types are allowed, the only wait-free hierarchy that is robust is the trivial one, which lumps all types into a single level. In particular, the consensus hierarchy (the most closely studied wait-free hierarchy) is not robust. Our result implies that, in general, it is not possible to determine the power of a concurrent system that supports a given set of primitive object types by reasoning about the power of each primitive type in isolation.

Key words. wait-free, consensus number, robustness

AMS subject classifications. 68Q10, 68Q22

PII. S0097539798335766

1. Introduction.

1.1. Background and overview of the result. Interest in asynchronous shared-memory distributed computing dates back at least to Dijkstra's work on mutual exclusion [4]. In this model of computation, a set of processes cooperate toward some common task (such as excluding each other from a critical section) by asynchronously taking steps, some of which access shared registers. The operations that the processes can apply to the shared registers may be simple read and write operations, or more complex ones such as test-and-set, fetch-and-add, and so forth.

In a more general version of this model of computation, processes communicate by accessing shared objects. An object is accessed by applying an operation to it; the operation changes the state of the object and returns a value to the process that invoked it. Each object belongs to a type. A type consists of a set of states, a set of operations, a set of responses to operations, and a state transition relation that describes the effect of applying each operation to an object of that type. More specifically, the state transition relation specifies the responses the object may return and the states it may enter if an operation is applied when the object is in a particular state. Examples of types include register (with read and write operations only), test-and-set-register (with test-and-set operation), stack, queue, and priority-queue, among many others. (We use the sans serif font to indicate object types.)

*Received by the editors March 18, 1998; accepted for publication (in revised form) September 8, 1999; published electronically July 13, 2000. A preliminary version of the results in this paper appeared in *Proceedings of the* 29th *ACM SIGACT Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 579–588.

http://www.siam.org/journals/sicomp/30-3/33576.html

†Department of Computer Science, University of Toronto, 6 King's College Rd., Toronto, Ontario, Canada M5S 3H5 (wklo@cs.utoronto.ca, vassos@cs.utoronto.ca). The research of the first author was supported by a Canadian Commonwealth scholarship. The research of the second author was supported by a grant from the Natural Sciences and Engineering Research Council of Canada and a fellowship from the Engineering and Physical Sciences Research Council of the UK.

A distributed system supports, in hardware or software, a certain fixed set of "base" object types. To enhance the usability of such a system, a programmer may wish to enrich the set of available object types by implementing new types not directly supported by the system. To implement a type T from a set of base types $\mathcal{B}$ in a system of n processes, the programmer must provide n procedures for each operation *op* of T; these are the procedures that may be invoked by the n processes to apply *op* to an object of type T. These procedures may use any number of objects, all belonging to types in $\mathcal{B} \cup \{\mathsf{register}\}$. Note that this notion of implementation takes registers for granted. (For convenience, in the rest of the paper we say simply "registers" instead of "objects of type register.") This is reasonable since any realistic shared-memory system will provide at least this much.[1]

An implementation is required to be *linearizable* [7]: Any concurrent execution E consisting of invocations of the procedures written by the programmer must be equivalent to a *sequential* execution E_s consisting of the same procedure invocations, where each invocation returns a response and changes the state of the object it accesses in accordance with the state transition relation of the implemented type. Moreover, the sequential execution E_s must respect the order of procedure invocations that did not overlap in E; it may serialize in any way procedure invocations that were active concurrently in E.

Of special interest are *wait-free* implementations. In such implementations, a process is guaranteed to complete any procedure it invokes within a finite number of its own steps, regardless of whether other processes are fast, slow, or have even crashed (i.e., halted before completing). The requirement of wait freedom precludes the use of critical sections and busy waiting. In this paper, we consider only wait-free and linearizable implementations.

A fundamental question is whether, in a system of n processes, it is possible to implement a type T from a given set of base types $\mathcal{B}$. In a seminal paper [6], Herlihy established an important link between a type's ability to support implementations of other types and its ability to solve the well-known consensus problem [5], in which processes must reach agreement on one of their input values. (This problem is formally defined in section 3.1.) More specifically, he showed that if objects that belong to a set of types $\mathcal{B}$ (together with registers) can be used to solve consensus among n processes, then objects that belong to types in $\mathcal{B}$ (together with registers) can be used to implement *any* type in a system of n processes. This fact is referred to as "the universality of consensus."

Define the *consensus number* of a type T, as the maximum number n such that there is an algorithm that solves consensus among n processes using only objects of type T (and registers)—or ∞, if no such maximum exists. We can now place each type in the *consensus hierarchy* according to its consensus number. By the universality of consensus, the relative strength of two types T and T' is captured by their levels, say, n and n', respectively, in the consensus hierarchy. If $n \geq n'$, then T is at least as strong as T': in a system of n processes, T can be used to implement T'. If, further, $n > n'$, then T is strictly stronger than T': T' cannot be used to implement T in a system of n processes.

Although consensus numbers allow us to make *pairwise* comparisons on the

[1] The notion of implementation that we use in this paper is the most prevalent in the literature, although more restricted ones have also been studied. For instance, in some definitions registers are not taken for granted, while in others only one object of each base type may be used in the implementation. For further discussion on such more restricted definitions, see [8] and [10].

strength of types, it is not clear whether they can also be used to answer the more general implementability question: Can we implement T from a *set* of base types $\mathcal{B}$, in a system of n processes? Jayanti [8] observed that consensus numbers could be used to answer this question if the consensus hierarchy is *robust*; i.e., no type at level n can be implemented from types at levels strictly below n in a system of n processes.

If the consensus hierarchy is robust, a type T of consensus number n can be implemented from the set of base types $\mathcal{B}$ in a system of n processes if and only if $\mathcal{B}$ contains a type with consensus number n or more. Thus, we can answer the implementability question by looking at the consensus numbers of the base types and the desired target type. Another way of thinking about robustness is that if the consensus hierarchy has this property, a set of types $\mathcal{B}$ is as strong as the strongest of the types it contains. Thus, we can determine the power of a given set of types by reasoning about the power of each type in isolation. If, on the other hand, the consensus hierarchy is not robust, then to determine the power of a set of types we must, in general, take into account the interactions between the types in the set—a more formidable task. In view of the importance of the consensus hierarchy and the salutary consequences of robustness, it is useful to know whether the consensus hierarchy is robust.

Borowsky, Gafni, and Afek [1] and Peterson, Bazzi, and Neiger [15] proved that if we restrict our attention to *deterministic* types, the consensus hierarchy is robust. A type is deterministic if whenever we apply an operation to the type, the current state uniquely determines the response of the operation and the new state of the type. Although most of the common types are deterministic, some useful types are most naturally specified as nondeterministic ones. For example, consider a priority queue where keys are not unique. If several elements are in the queue with the same maximum priority, it is natural to allow the priority queue to remove and return any one of these elements. It is, of course, always possible to turn a nondeterministic type into a deterministic one by restricting the allowable behavior of the type. This, however, results in unnecessary restrictions for the implementor of the type. As pointed out in [11], by arbitrarily and artificially excluding legitimate behavior, we may be inadvertently ruling out efficient implementations of the type. Thus, the investigation of properties of nondeterministic types is of practical relevance.

In this paper we prove that if we allow nondeterministic types, then the consensus hierarchy is *not* robust. In view of this, a natural question is whether there is some other hierarchy of types which, like the consensus hierarchy, classifies types according to their strength in supporting wait-free implementations, but, unlike the consensus hierarchy, is robust. Jayanti proved that only a coarsening of the consensus hierarchy can have these properties [8]. In this paper we show that, in fact, the only hierarchy that has these properties is the trivial one in which all types are lumped into level one!

1.2. Overview of the proof. To prove our result we demonstrate two nondeterministic (but finite) object types, negation and booster; each of these is at level one of the consensus hierarchy, but, when combined, they can implement any type in a system of n processes, for any n. The central idea behind our proof is the formulation of *equality negation*, a weaker variant of the consensus problem which is formally defined in section 3.2. An object of type negation enables processes to directly solve this problem. An object of type booster allows processes to solve consensus provided they can prove that they can solve equality negation among them using initial values specified by the booster object.

We give a simple algorithm that allows n processes to solve consensus using a negation object, a booster object, and registers. (This algorithm suggests itself almost immediately from the definition of the booster type.) By the universality of consensus, this implies that the two types together can implement any type in a system of n processes. The difficult part of our result is in proving that each of negation and booster has consensus number one. The usual technique for proving upper bounds on the consensus number of a type, known as the "bivalence argument" [5], works well for deterministic types but runs into difficulties in the context of nondeterministic types. The bulk of the technical work in the paper (sections 7 and 8) seeks ways to circumvent these difficulties.

1.3. Related work. Jayanti was the first to raise the question of robustness of the consensus hierarchy [8]. Before his work, it had been widely, though tacitly, assumed that the hierarchy is robust. Jayanti showed that various natural variants of the consensus hierarchy (including the one used in [6]) are not robust. He also proved that if a robust hierarchy that classifies types according to their ability to implement other types exists, it must be either the consensus hierarchy or a coarsening of it—i.e., a hierarchy obtained by collapsing together adjacent levels of the consensus hierarchy.

As mentioned earlier, Borowsky, Gafni, and Afek [1] and Peterson, Bazzi, and Neiger [15] proved that the consensus hierarchy, restricted to deterministic types, is robust. In recent work, Ruppert [18] gave simple proofs of robustness if we restrict our attention to certain interesting classes of deterministic types.

In the next paragraph we describe some results concerning the (non)robustness of the consensus hierarchy in a model of objects that differs from the "standard" model introduced in [6] (and used in the present paper). To explain the different models we recall that, in a system of n processes, each object of type T has n procedures for each operation *op* of T. Processes trying to apply *op* to an object concurrently must invoke different procedures. In the "standard" model of objects a process that wishes to apply *op* to the object invokes one of these n procedures and remains bound to it only for the duration of the invocation. Other processes may later use that procedure. In the alternative model, each process permanently binds itself to one procedure. Objects in this model are sometimes called *hardwired*, since the binding of processes to procedures is fixed.

Chandra et al. [2] prove that in the model of hardwired objects, the consensus hierarchy is not robust if nondeterministic types are allowed. Moran and Rappoport [14] strengthen this result by proving that it applies even if only deterministic types are allowed. We borrowed from [2] the idea of a booster type, which "boosts" the power of a weak type T if it is given satisfactory evidence that processes have access to T. The booster of [2] promotes a type that solves consensus between two processes to a type that solves consensus among three processes. In our case, the booster promotes a type that solves equality negation for n processes to a type that solves consensus for n processes.

Closest to our results is the work of Schenk [19, 20], which precedes ours. He also exhibits two nondeterministic object types each of which (together with registers) is individually too weak to solve consensus between two processes, but which, taken together, can solve consensus for any number of processes. Our result strengthens this in two related ways: First, Schenk's result requires a type that exhibits infinite nondeterminism, while our types exhibit only bounded nondeterminism. Second, and perhaps more important, his result requires a definition of wait freedom that is different from the usual one. Specifically, he defines an algorithm to be wait-free if,

for any input, there is a bound on the number of steps a process may take before it halts. In the context of objects with infinite nondeterminism this definition is strictly stronger than the usual definition of wait freedom, which requires only that all executions of the algorithm be finite.

In [17], Rachman proved that for each n, there is a (nondeterministic) type with consensus number n that cannot implement some type with consensus number smaller than n in a system of $2n+1$ processes. This result addresses a point of implementability between types, but it has no bearing on the nonrobustness of the consensus hierarchy. Informally, nonrobustness asserts that some "strong" type (i.e., a type at level n) *can be implemented by* a combination of "weak" types (i.e., types at levels below n) *in a system of n processes*. In contrast, Rachman's result says that some "strong" type (i.e., a type at level n) *cannot implement* some "weak" type (i.e., a type at a level below n) *in a system with sufficiently many processes*.

1.4. Organization of the paper. In section 2 we define the model of computation used in this paper. In section 3 we review the consensus problem and introduce the equality negation problem. In section 4 we define the types negation and booster. In section 5 we show how to solve consensus for any number of processes using registers along with a negation and a booster object. In section 6 we define some concepts and derive technical results needed in the next two sections. In sections 7 and 8 we prove that registers and negation objects alone, as well as registers and booster objects alone, cannot be used to solve consensus for two processes. Section 9 concludes the paper.

2. The model of computation. In this section we describe the model of computation used in this paper.

2.1. Types and objects. An *object type* T is a tuple (Q, OP, RES, δ), where Q is a set of *states*, OP is a set of *operations*, RES is a set of *responses* to operations, and $\delta \subseteq Q \times OP \times RES \times Q$ is a *state transition relation*, which describes the possible behaviors of objects of type T, and is sometimes called the type's *sequential specification*. More specifically, $(q, op, res, q') \in \delta$ means that if the current state of an object of type T is q and operation op is applied to it, then it is possible that response res will be returned to the operation and the object will enter state q'. We say that T is *deterministic* if for any $q \in Q$ and $op \in OP$ there is at most one $res \in RES$ and $q' \in Q$ so that $(q, op, res, q') \in \delta$; otherwise, we say that T is *nondeterministic*.

For convenience, in this paper we require each type (Q, OP, RES, δ) to be *total*: For any $q \in Q$ and $op \in OP$, there is at least one $res \in RES$ and $q' \in Q$ such that $(q, op, res, q') \in \delta$. That is, the type can do *something* in response to any operation in any state. Further, we assume that for any $q \in Q$, $op \in OP$, and $res \in RES$ there is at most one $q' \in Q$ so that $(q, op, res, q') \in \delta$. In other words, if an operation is applied in a given state, the type does not lead to different states unless it returns different responses. We remark that this assumption is not necessary for the results in this paper, but it simplifies the presentation of their proofs. All types considered in this paper have this property, so we can make this assumption without loss of generality.

An *object* is an instance of an object type. For the purposes of this paper, we can think of an object O of type T as an automaton whose states and state transition relation are as in T—except that states are labeled with O, to distinguish them from the states of other objects of the same type.

2.2. Processes and algorithms. A *process* is a deterministic automaton that interacts with objects. More precisely, let $\mathcal{O}$ be a set of objects, and let OP and

RES be the set of all operations and responses, respectively, of the types to which the objects in $\mathcal{O}$ belong. We define a *process that uses* $\mathcal{O}$ as a tuple $P = (\Sigma, \Sigma_0, \nu, \tau)$, where Σ is a set of *states*, $\Sigma_0 \subseteq \Sigma$ is a set of *initial states*, and $\nu : \Sigma \to OP \times \mathcal{O}$ and $\tau : \Sigma \times RES \to \Sigma$ are functions that describe the interaction of the process with the objects. Intuitively, if P is in a state $\sigma \in \Sigma$ and $\nu(\sigma) = \langle op, O \rangle$, then in its next step P will apply operation *op* to object O. (If $\nu(\sigma) = \langle op, O \rangle$, we require that *op* be an operation of O's type.) Based on its current state, O will return a response *res* to P and will enter a new state, in accordance with the state transition relation of the type to which O belongs. Finally, P will enter state $\tau(\sigma, res)$, as a result of the response it received from O. For convenience, we assume that the function ν is total. In other words, it is always possible for a process to apply another operation. Note that we can easily accommodate processes that terminate in this framework by imagining that a process that reaches a final state applies infinitely many "DO-NOTHING" operations to a local dummy object which always returns *ack* to such operations.

An *algorithm* **A** consists of a set of processes Π, a set of objects $\mathcal{O}$ so that each $P \in \Pi$ uses a subset of $\mathcal{O}$, and an initial state for each object in $\mathcal{O}$. The designated initial state of an object is one of the states of the type to which the object belongs. If $\mathcal{S}$ is a set of types, we say that **A** *uses* $\mathcal{S}$ if every object that **A** uses belongs to a type in $\mathcal{S}$ and **A** uses at least one object of every type in $\mathcal{S}$. A *configuration* C of **A** is a tuple consisting of a state for each process in Π and each object in $\mathcal{O}$. C is an *initial* configuration of **A** if each process is in one of its initial states, and each object is in the state designated as the initial state by **A**.

A *step* of process P is a tuple (P, op, O, res); this indicates that P has applied operation *op* to object O and received response *res*. Let $P = (\Sigma, \Sigma_0, \nu, \tau)$ and let C be a configuration, where the state of P in C is σ. If $\nu(\sigma) = \langle op, O \rangle$, we say that P has operation *op* to O *pending* in C. If, in addition, the state of O in C is q, and *res* is a legitimate response of O to *op* in state q (i.e., for some state q' of O, (q, op, res, q') is in the state transition relation of O's type), then we say that the step $e = (P, op, O, res)$ is *applicable* to C. If $e = (P, op, O, res)$ is applicable to C, $e(C)$ denotes the configuration resulting from C after step e. More precisely, if in C the state of P is σ and the state of O is q, then $e(C)$ is the configuration in which P has state $\tau(\sigma, res)$, O is in state q' such that (q, op, res, q') is in the state transition relation of O's type, and all other processes and objects are in the same state as in C.[2]

A *schedule* S *of algorithm* **A** is a (finite or infinite) sequence of steps of **A**'s processes. $S = e_1, e_2, \ldots, e_i, \ldots$ is *applicable* to a configuration C if e_1 is applicable to C, and e_{i+1} is applicable to $e_i(e_{i-1}(\ldots(e_1(C))\ldots))$ for all i. If S is finite and has k steps, $S(C)$ denotes $e_k(e_{k-1}(\ldots(e_1(C))\ldots))$, i.e., the configuration that results after applying the steps in S one at a time, starting with configuration C. If S and S' are schedules such that S is finite, then $S \cdot S'$ denotes their concatenation. If every step in schedule S is a step of the same process P, then S is called a *solo* schedule of P. If S is an infinite schedule and process P has infinitely many steps in S, then P is *correct* in S.

2.3. Wait-free hierarchies and robustness. Recall from section 1.1 that the consensus number of a type T is the maximum number n so that there is an algorithm that solves consensus among n processes and uses only objects of type T and registers;

[2]It is here we use the assumption that, to go to different states, the object must return different responses. Without this assumption, we have to define $e(C)$ as a *set* of configurations, rather than as a single one.

if there is no such maximum, the consensus number of T is ∞. By the universality of consensus, the consensus number of a type is a measure of its strength, i.e., of its ability to support (wait-free and linearizable) implementations of other types.

A *hierarchy* (of types) is a function that assigns to each type a positive integer or ∞, called the type's *level*. The intention is that a type's level reflects its strength: the higher the level, the stronger the type. A hierarchy is *wait-free* if no type is assigned to a level higher than the type's consensus number. Intuitively, a wait-free hierarchy does not overstate any type's strength. Thus, a hierarchy must be wait-free in order to comply with the intention that its levels somehow reflect the strength of types. The *consensus hierarchy* is the hierarchy that maps each type exactly to its consensus number. Clearly, the consensus hierarchy is wait-free.

A hierarchy h is *robust* if for any $n \geq 1$ such that there is a type at level n and any set of types $\mathcal{B}$, there is no algorithm that solves consensus among n processes and uses only objects that belong to types in $\mathcal{B}$ and registers, unless $\mathcal{B}$ contains a type of level n or higher. Thus, if a wait-free hierarchy is robust, a type that is classified as "strong" by the hierarchy cannot be implemented by types that are classified as "weak." To see this assume the contrary: a type T at level n can be implemented using only objects of types below n (and registers). Since the hierarchy is wait-free and T is at level n, we can solve consensus among n processes using only objects of type T (and registers). Since T can be implemented from types in levels below n and registers, we can solve consensus among n processes using only objects of types in levels below n and registers, which contradicts the definition of robustness.

A hierarchy is *nontrivial* if it assigns types to at least two different levels. Jayanti and Toueg [9] showed that the consensus hierarchy assigns types to all levels, and so the consensus hierarchy is nontrivial.

3. The consensus and equality negation problems. In this section we review the consensus problem and define equality negation, a problem that plays a central role in our results.

3.1. The consensus problem. In the consensus problem, each of n processes starts with a private initial value, drawn from the set $\{0, 1\}$. Each correct process must eventually irrevocably decide one of these initial values, so that no two processes decide different values. Thus, a *consensus algorithm* $\mathbf{A}_c$ *for* n *processes* satisfies the following properties. Each process in $\mathbf{A}_c$ has two initial states associated with input values 0 and 1, respectively, and two disjoint sets of states associated with decisions 0 and 1, respectively. Since a decision is irrevocable, we require that once a process has entered a state associated with decision $d \in \{0, 1\}$, all states that it may subsequently enter are also associated with d. In addition, for any schedule S that is applicable to an initial configuration I of $\mathbf{A}_c$, the following three properties hold.

Termination: If S is infinite, then for every correct process in S there is a prefix S' of S such that the process has decided in $S'(I)$.

Validity: If a process has decided in $S(I)$, then its decision must be the initial value of some process in I.

Agreement: If two processes have decided in $S(I)$, then their decisions are the same.

The termination property requires correct processes to eventually decide in all executions, regardless of the number of processes that are correct. Thus, this formulation of this property commits us to so-called *wait-free* algorithms, where correct processes satisfy their liveness properties (in this case, to eventually decide) regardless of the progress made by other processes.

3.2. The equality negation problem. We now define the equality negation problem for $n \geq 2$ processes, $P_0, \ldots, P_{n-1}$. To understand the definition it is best to first consider the special case of two processes ($n = 2$). In this special case, each of P_0 and P_1 has a private initial value, drawn from the set $\{0, 1, 2\}$ (instead of $\{0, 1\}$ in consensus). Each correct process must irrevocably decide either 0 or 1 so that the decisions of the processes are the same if and only if the initial values of the processes are different.

We now generalize the problem for arbitrary $n \geq 2$; the idea is to replace the role of P_1 by a *group* of $n - 1$ processes $\mathcal{P}_1 = \{P_1, \ldots, P_{n-1}\}$. More specifically, each process has an initial value drawn from the set $\{0, 1, 2\}$. Additionally, we assume that all processes in $\mathcal{P}_1$ have the same initial value; if not, the problem imposes no constraints whatsoever on the behavior of processes. Every process must eventually decide irrevocably either 0 or 1, so that no two processes in $\mathcal{P}_1$ decide different values; and if P_0 and any process in $\mathcal{P}_1$ decide, then their decisions are the same if and only if their initial values are different. Thus, an *equality negation algorithm* $\mathbf{A}_{en}$ *for processes* $P_0, \ldots, P_{n-1}$ satisfies the following properties. Each process in $\mathbf{A}_{en}$ has three initial states associated with input values 0, 1, and 2, respectively. As in a consensus algorithm, each process has two disjoint sets of states associated with decisions 0 and 1, respectively. Since a decision for equality negation is also irrevocable, once a process has entered a state associated with decision $d \in \{0, 1\}$, all states that it may subsequently enter must be associated with d as well. In addition, for any schedule S applicable to an initial configuration I of $\mathbf{A}_{en}$ in which all processes in $\mathcal{P}_1$ have the same initial value, the following three properties hold.

Termination: If S is infinite, then for every correct process in S, there is a prefix S' of S such that the process has decided in $S'(I)$.

Negation: If process P_0 and any process in $\mathcal{P}_1$ have decided in $S(I)$, then their decisions are the same if and only if their initial values are different.

Agreement: If any two processes in $\mathcal{P}_1$ have decided in $S(I)$, then their decisions are the same.

It is well known that there is no consensus algorithm for even two processes that uses only registers [3, 13]. In the appendix we prove that the same is true for equality negation.

THEOREM 3.1. *There is no algorithm that solves equality negation for two processes and uses only registers.*

4. Specification of types negation and booster. We define two nondeterministic object types, named negation and booster, that will be used later to establish the result that no nontrivial wait-free hierarchy is robust. Both of these types have bounded nondeterminism; indeed, each has only finitely many states.

4.1. Type negation. The specification of negation appears in Figure 4.1.[3] To understand the specification, it is useful to keep in mind the following intuitive explanation of the type. This type supports only one kind of operation, called NEGATE, with two parameters $i \in \{0, 1\}$ and $v \in \{0, 1, 2\}$. For reasons that will be clear later, we shall refer to i as the *process* parameter and to v as the *initial-value* parameter. A negation object will become "upset" if and only if two NEGATE operations with process parameter 0 are applied or two NEGATE operations with process parameter 1, but with different initial-value parameters, are applied. If the negation object is

[3] In this figure, and throughout this paper, for any variable $v \in \{0, 1\}$, $\overline{v}$ denotes the complement of v; i.e., $\overline{v} = 1 - v$.

Let $\mathcal{A}$ be the set of four-tuples of the form $\langle v_0, v_1, u_0, u_1\rangle$, where $v_0, v_1 \in \{\perp, 0, 1, 2\}$, $u_0, u_1 \in \{\perp, 0, 1\}$, and the following conditions apply:

- $v_0 = \perp$ if and only if $u_0 = \perp$;
- $v_1 = \perp$ if and only if $u_1 = \perp$;
- at least one of v_0, v_1 is not $\perp$; and
- if $v_0, v_1 \neq \perp$ then $v_0 = v_1$ if and only if $u_0 \neq u_1$.

Set of states: $\{☺, ☹\} \cup \mathcal{A}$.

Set of operations: $\{\,\text{NEGATE}(i, v) : i \in \{0, 1\}, v \in \{0, 1, 2\}\,\}$.

Set of responses: $\{0, 1\}$.

```
NEGATE(i, v), i ∈ {0,1}, v ∈ {0,1,2}, return either 0 or 1
  case current_state
    ☹: return Choose({0,1})
    ☺: u ← Choose({0,1})
       if i = 0 then current_state ← ⟨v, ⊥, u, ⊥⟩
       else current_state ← ⟨⊥, v, ⊥, u⟩
       return u
    ⟨v0, v1, u0, u1⟩:
       if i = 0 then
          if v0 ≠ ⊥ then current_state ← ☹; return Choose({0,1})
          else (* v0 = ⊥ and, thus, v1, u1 ≠ ⊥ *)
             if v ≠ v1 then u ← u1
             else u ← ū1
             current_state ← ⟨v, v1, u, u1⟩; return u
       else (* i = 1 *)
          if v1 ≠ ⊥ then
             if v ≠ v1 then current_state ← ☹; return Choose({0,1})
             else return u1
          else (* v1 = ⊥ and, thus, v0, u0 ≠ ⊥ *)
             if v ≠ v0 then u ← u0
             else u ← ū0
             current_state ← ⟨v0, v, u0, u⟩; return u
```

FIG. 4.1. *Specification of type* negation.

upset, it will arbitrarily return either 0 or 1 to any operation, and will remain upset forever. As long as the negation object is not upset, it responds to invocations with values that satisfy the following: All NEGATE operations with process parameter 1 receive the same response; any two NEGATE operations with different process parameters receive the same response if and only if their initial-value parameters are different. Thus, an object O of type negation can be used to solve equality negation for processes $P_0, \ldots, P_{n-1}$ in a direct way: P_0 applies NEGATE$(0, v)$ to O, where v is P_0's initial value; each process in $\{P_1, \ldots, P_{n-1}\}$ applies NEGATE$(1, v')$ to O, where v' is the process's initial value; each process decides the value returned by object O.

To achieve this behavior, type negation has two distinguished states: the *fresh* state, denoted by ☺, and the *upset* state, denoted by ☹. The remaining states are of the form $\langle v_0, v_1, u_0, u_1\rangle$, where $v_0, v_1 \in \{\perp, 0, 1, 2\}$ and $u_0, u_1 \in \{\perp, 0, 1\}$. The intended meaning of state $\langle v_0, v_1, u_0, u_1\rangle$, assuming the object is initialized to the fresh state, is as follows: v_0 is the initial-value parameter of the first NEGATE operation

with process parameter 0; u_0 is the value returned to that operation; v_1 and u_1 have corresponding interpretations for the first NEGATE operation with process parameter 1. By maintaining this information, the object knows all it needs in order to respond appropriately to the NEGATE operations applied to it.

The pseudocode in Figure 4.1 describes the state transition relation of negation. In addition to standard programming constructs, we use the function "Choose," which takes as a parameter a set V of integers and returns an arbitrary value in V. We also assume that the present state of the type is recorded in variable *current_state*.

4.2. Type booster. The specification of type booster is shown in Figure 4.2. To understand the specification, it is useful to keep in mind the following intuitive explanation of the type. There are two kinds of operations that can be applied to a booster object: ENROLL and REVEAL. An ENROLL operation requires a parameter $i \in \{0,1\}$ and arbitrarily returns a value in $\{0,1,2\}$. A REVEAL operation requires three parameters $i \in \{0,1\}$, $v \in \{0,1,2\}$, and $u \in \{0,1\}$ and returns a value in $\{0,1\}$; we shall refer to these three parameters as the *process*, *challenge*, and *decision* parameters, respectively. A booster object will be "upset" if (and only if) any one of the following holds:

- Two ENROLL(0) operations are applied.
- For some $i \in \{0,1\}$ and $v \in \{0,1,2\}$, a REVEAL(i,v,u) operation is applied such that no ENROLL(i) operation has previously been applied or v is not the response of any such ENROLL(i) operation.
- A REVEAL(i_0,v_0,u_0) and a REVEAL(i_1,v_1,u_1) operation are applied such that one of the following is true: (a) $i_0 = i_1 = 0$, or (b) $i_0 = i_1 = 1$ and either $v_0 \neq v_1$ or $u_0 \neq u_1$, or (c) $i_0 \neq i_1$ and $v_0 = v_1 \Leftrightarrow u_0 = u_1$.

Once upset, the booster object remains upset forever. If not upset, the object maintains a secret: the parameter of the first ENROLL operation applied. The object will reveal this secret to every process that subsequently accesses it with a REVEAL operation. If upset, however, the object arbitrarily returns either 0 or 1 to every REVEAL operation.

The intended use of a booster object by processes $P_0, \ldots, P_{n-1}$ is as follows: The first thing process P_0 does to a booster object is to apply ENROLL(0) to "enroll" itself in the object. Similarly, the first thing any process in $\mathcal{P}_1 = \{P_1, \ldots, P_{n-1}\}$ does to a booster object is to apply ENROLL(1) to enroll its group in the object. In response to the ENROLL operation it applies, every process receives a value in $\{0,1,2\}$ from the booster object. This value is a "challenge" given by the object with which the process is required to solve equality negation. More specifically, process P_0 uses the challenge, say, v_0, received from the booster object as its initial value to solve equality negation. The different processes in $\mathcal{P}_1$, however, may receive different challenges from the booster object. Thus, before embarking upon solving equality negation, processes in $\mathcal{P}_1$ must first choose a unique initial value, say, v_1, from the set of challenges they received. After solving equality negation and deciding a value $u \in \{0,1\}$, process P_0 (respectively, every process in $\mathcal{P}_1$) applies the operation REVEAL$(0,v_0,u)$ (respectively, REVEAL$(1,v_1,u)$) to the booster object to obtain the secret.

To achieve the behavior described above, type booster (like type negation) has two distinguished states: the *fresh* state, denoted ☺, and the *upset* state, denoted ☹. The remaining states are of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$, where $V \subseteq \{0,1,2\}$, $v_0, v_1 \in \{\perp, 0, 1, 2\}$, $u_0, u_1 \in \{\perp, 0, 1\}$, and $d \in \{0,1\}$. The intended meaning of state $\langle V, v_0, v_1, u_0, u_1, d\rangle$ is as follows (assuming the object is initialized to the fresh state):

- d is the "secret"—the parameter the first ENROLL operation applied, i.e.,

Let $\mathcal{A}$ be the set of six-tuples of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$, where $V \subseteq \{0,1,2\}$, $v_0, v_1 \in \{\perp, 0, 1, 2\}$, $u_0, u_1 \in \{\perp, 0, 1\}$, $d \in \{0,1\}$, and the following conditions apply:

- if $d = 0$ then $v_0 \neq \perp$; if $d = 1$ then $V \neq \emptyset$;
- if $v_0 = \perp$ then $u_0 = \perp$;
- $v_1 = \perp$ if and only if $u_1 = \perp$, and if $v_1 \neq \perp$ then $v_1 \in V$; and
- if $v_0, v_1 \neq \perp$ then $v_0 = v_1$ if and only if $u_0 \neq u_1$.

Set of states: $\{☺, ☹\} \cup \mathcal{A}$.
Set of operations: $\{\,\text{ENROLL}(i) : i \in \{0,1\}\,\} \cup \{\,\text{REVEAL}(i, v, u) : i, u \in \{0,1\}, v \in \{0,1,2\}\,\}$.
Set of responses: $\{0,1,2\}$.

```
ENROLL(i), i ∈ {0,1}, return a value in {0,1,2}
    v ← Choose({0,1,2})
    case current_state
        ☹: no state change
        ☺: if i = 0 then current_state ← ⟨∅, v, ⊥, ⊥, ⊥, 0⟩
           else current_state ← ⟨{v}, ⊥, ⊥, ⊥, ⊥, 1⟩
        ⟨V, v0, v1, u0, u1, d⟩:
           if i = 0 then
              if v0 ≠ ⊥ then current_state ← ☹
              else current_state ← ⟨V, v, v1, u0, u1, d⟩
           else current_state ← ⟨V ∪ {v}, v0, v1, u0, u1, d⟩
    return v

REVEAL(i, v, u), i, u ∈ {0,1}, v ∈ {0,1,2}, return a value in {0,1}
    case current_state
        ☺: current_state ← ☹; return Choose({0,1})
        ☹: return Choose({0,1})
        ⟨V, v0, v1, u0, u1, d⟩:
           if i = 0 then
              if v = v0 and u0 = ⊥ and (u1 = ⊥ or (v = v1 ⇔ u ≠ u1)) then
                 current_state ← ⟨V, v0, v1, u, u1, d⟩; return d
              else current_state ← ☹; return Choose({0,1})
           else (* i = 1 *)
              if v ∈ V and ((u1 = ⊥ and (u0 = ⊥ or (v = v0 ⇔ u ≠ u0)))
                            or (v = v1 and u = u1)) then
                 current_state ← ⟨V, v0, v, u0, u, d⟩; return d
              else current_state ← ☹; return Choose({0,1})
```

FIG. 4.2. *Specification of type* booster.

whether P_0 or some process in $\mathcal{P}_1$ is the process that applied the first ENROLL. More accurately, d is the parameter of the first ENROLL operation.

- V is the set of challenges given to processes in $\mathcal{P}_1$, one of which these processes must use to solve equality negation. More accurately, V is the set of values returned to ENROLL(1) operations.
- v_0 is the challenge with which P_0 must solve equality negation. More accurately, the value returned to the first ENROLL(0) operation.
- v_1 is the challenge chosen from V by processes in $\mathcal{P}_1$, with which these pro-

cesses solve equality negation. More accurately, v_1 is the challenge parameter of the first REVEAL$(1, *, *)$ operation.[4]

- u_0 is the decision reached for equality negation by P_0. More accurately, u_0 is the decision parameter of the first REVEAL$(0, *, *)$ operation.
- u_1 is the decision reached for equality negation by the first process in $\mathcal{P}_1$ that tries to get the secret. More accurately, u_1 is the decision parameter of the first REVEAL$(1, *, *)$ operation.

With this information, the object knows all it needs in order to respond appropriately to operations.

5. The proof of nonrobustness. Later in the paper we shall prove that negation and booster each has consensus number one. Taking this for granted, in this section we prove our main result: in the context of nondeterministic types, no nontrivial wait-free hierarchy can be robust. The key to this proof is the following lemma.

LEMMA 5.1. *For any $n \geq 1$, if consensus among n processes can be solved using only* negation *and* booster *objects and registers, then consensus among $n+1$ processes can also be solved using only* negation *and* booster *objects and registers.*

Proof. Suppose that consensus among n processes can be solved using only negation and booster objects and registers. Let $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ be algorithms that solve consensus among n processes, $P_1, P_2, \ldots, P_n$, and use only negation and booster objects and registers. Without loss of generality, we may assume that $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ solve consensus with initial values drawn from the set {0,1,2}.[5]

We shall show that the algorithm in Figure 5.1 uses $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ to solve consensus among $n+1$ processes, $P_0, P_1, \ldots, P_n$. The following are some conventions used in the pseudocode in Figure 5.1 and other algorithms described later. We use capital letters for names of shared objects (including registers) and lower-case letters for names of local variables. Local variables with the same name that belong to different processes are distinct. The notation "Apply(P, op, O)" denotes the procedure by which process P applies operation op to object O; the procedure returns the response of the object to that operation. We use a special (but familiar) notation for the application of operations to an object R of type register: we place R on the left-hand side of an assignment statement to indicate a WRITE operation applied to R, and we use the name R in any other context to indicate a READ operation applied to R. If $\mathbf{A}$ is an equality negation or consensus algorithm and P is a process of $\mathbf{A}$, the procedure Execute$(P, \mathbf{A}, k)$ causes the caller to execute the steps of process P in algorithm $\mathbf{A}$ using k as its initial value and returns the decision of P in the simulated execution of that algorithm.

Before proving that the algorithm in Figure 5.1 does, in fact, solve consensus, we informally explain how it works.

- P_0 first writes its initial value into register R_0. It then enrolls in the booster object O_b by applying ENROLL(0) and obtains a challenge, say, v, with which it is required to solve equality negation. To do so, P_0 applies NEGATE$(0, v)$ to the negation object O_n. Let u be the value returned by O_n. By applying REVEAL$(0, v, u)$ to O_b, P_0 determines whether it or one of $\{P_1, \ldots, P_n\}$ was the first to enroll in O_b.

[4]Here, and in the rest of the paper, we use "$*$" to indicate a "don't care" entry.

[5]In our definition of consensus (sometimes called *binary* consensus for emphasis), the initial values of processes are drawn from the set $\{0, 1\}$. It is well known [16] that if a set of types that includes register can be used to solve (binary) consensus, it can also be used to solve consensus with initial values drawn from an arbitrary set.

Shared: all booster and negation objects and registers used in $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$, each initialized as specified by $\mathbf{A}_c^1$ or $\mathbf{A}_c^2$
O_b: booster, initialized to ☺
O_n: negation, initialized to ☺
R_0, R_1: register, each initialized to $\perp$

Code for process P_0

1 $R_0 :=$ initial value of P_0
2 $v := \text{Apply}(P_0, \text{ENROLL}(0), O_b)$
3 $u := \text{Apply}(P_0, \text{NEGATE}(0, v), O_n)$
4 $d := \text{Apply}(P_0, \text{REVEAL}(0, v, u), O_b)$
5 **decide** R_d

Code for process P_k, $k \in \{1, \ldots, n\}$

1 *init* := initial value of P_k
2 $R_1 := \text{Execute}(P_k, \mathbf{A}_c^1, \mathit{init})$
3 $w := \text{Apply}(P_k, \text{ENROLL}(1), O_b)$
4 $v := \text{Execute}(P_k, \mathbf{A}_c^2, w)$
5 $u := \text{Apply}(P_k, \text{NEGATE}(1, v), O_n)$
6 $d := \text{Apply}(P_k, \text{REVEAL}(1, v, u), O_b)$
7 **decide** R_d

FIG. 5.1. *A consensus algorithm for $n+1$ processes using only objects of types* negation *and* booster *and registers.*

- Processes $P_1, \ldots, P_n$ first agree on one of their initial values; they use $\mathbf{A}_c^1$ for that purpose. They then write the agreed-upon initial value into register R_1 (multiple processes may write into R_1, but they all write the same value). Each P_k, $k \in \{1, \ldots, n\}$, then enrolls in O_b by applying ENROLL(1) and obtains a challenge. Since, object O_b may hand out different challenges to processes $P_1, \ldots, P_n$, these processes use the consensus algorithm $\mathbf{A}_c^2$ to agree upon one of these challenges, say, v. Each P_k, $k \in \{1, \ldots, n\}$, uses v as its initial value to solve equality negation. To do so, P_k applies NEGATE$(1, v)$ to the negation object O_n. Let u be the value returned by O_n. By applying REVEAL$(1, v, u)$ to O_b, P_k determines whether P_0 or one of the other processes was the first to enroll in O_b.

Since all processes agree on whether P_0 or one of the other processes was the first to enroll in O_b, they can agree on the initial value of some process: If P_0 was the first to enroll, all processes agree on the value they read from R_0; otherwise, they all agree on the value they read from R_1.

We now prove that the algorithm in Figure 5.1 solves consensus. It is clear that the algorithm satisfies termination, since both $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$ do. Consider any execution of the algorithm. From the specification of booster and negation, it is easy to verify that the algorithm accesses O_b and O_n in such a way that these objects never become upset. Furthermore, if d is the parameter of the first ENROLL operation applied to O_b, then that ENROLL operation will cause O_b to enter a state of the form $\langle *, *, *, *, *, d \rangle$. The last component of the state is never changed once it is set. Furthermore, any REVEAL operation applied to O_b that does not cause the object to become upset returns the last component of the state. Thus,

(1) every REVEAL operation applied to O_b returns d, where d is the parameter of the first ENROLL operation applied to O_b.

Next we claim that, for each $k \in \{0, 1\}$, at most one value is written into register R_k and that value is an initial value of some process. This is clearly true for $k = 0$, since only P_0 writes into R_0, and the value it writes there is its initial value. For $k = 1$, this follows from the agreement and validity properties of the consensus algorithm $\mathbf{A}_c^1$ used by the processes in $\{P_1, \ldots, P_n\}$ to determine the value they each write into R_1.

Thus, to prove the algorithm in Figure 5.1 satisfies validity, it suffices to show that if a process decides the value in register R_d, then some process has previously written a value into R_d. This follows from (1), since in the algorithm a process writes into R_k before applying ENROLL(k), for each $k \in \{0, 1\}$.

It remains to show that the algorithm satisfies agreement. Consider any execution in which two distinct processes P and P' have decided. By (1), P and P' decide the value in the same register R_d, where d is the parameter of the first ENROLL operation applied to O_b. By validity, a process decides the value in R_d only if some process has written a value into R_d. As argued earlier, only one value is written into R_d in any given execution. Therefore, P and P' decide the same value, the one written in R_d. Therefore, the algorithm satisfies agreement.

Thus the algorithm in Figure 5.1 solves consensus for $n + 1$ processes. It uses only registers and booster and negation objects because, in addition to the (registers, booster and negation) objects used by $\mathbf{A}_c^1$ and $\mathbf{A}_c^2$, the algorithm uses only two registers, one booster object, and one negation object. □

THEOREM 5.2. *For any $n \geq 1$, consensus among n processes can be solved using only registers and objects of types* negation *and* booster.

Proof. The theorem is proved by induction on n. The basis, $n = 1$ is trivial (because consensus for one process is trivial). The induction step is Lemma 5.1. □

COROLLARY 5.3. *There is no nontrivial wait-free hierarchy that is robust.*

Proof. Suppose h is a nontrivial wait-free hierarchy of types. We shall prove that h is not robust. By Theorems 7.10 and 8.15 (see sections 7 and 8, respectively) the consensus number of negation and booster is 1. Since h is wait-free, $h(\text{negation}) \leq 1$ and $h(\text{booster}) \leq 1$, and thus $h(\text{negation}) = h(\text{booster}) = 1$. Since h is nontrivial, there is some level $n > 1$ and some type T such that $h(T) = n$. Let $\mathcal{B} = \{\text{negation}, \text{booster}\}$. By Theorem 5.2, there is an algorithm that solves consensus among n processes and uses only objects that belong to types in $\mathcal{B}$ and registers. Since $h(\text{negation}) = h(\text{booster}) = 1$ and $n > 1$, $\mathcal{B}$ contains no type that is at level n or higher. Thus, h is not robust, as wanted. □

6. Some technical preliminaries. In this section we define some concepts and prove some technical results that are needed in our proofs that each of the types negation and booster has consensus number one.

6.1. Computation trees. Let $\mathbf{A}$ be an algorithm, and let I be an initial configuration of $\mathbf{A}$. Consider a tree $\mathcal{T}$ whose nodes are finite schedules of $\mathbf{A}$ that are applicable to I (not necessarily all of them), so that the root of $\mathcal{T}$ is the empty schedule, denoted $S_\perp$, and there is an edge from node S to node S' only if $S' = S \cdot e$ for some step e. We say that $\mathcal{T}$ is a *computation tree of* $\mathbf{A}$ *from* I if for any node S of $\mathcal{T}$ and any process P of $\mathbf{A}$ S has at least one child $S \cdot e$, where e is a step of P. In what follows we shall not distinguish between a node in a computation tree and the (finite) schedule corresponding to the node. Similarly, we shall not distinguish between a (finite or infinite) path in a computation tree and the (finite or infinite) schedule corresponding to the path.

Informally, a computation tree of $\mathbf{A}$ from I represents executions of $\mathbf{A}$ starting from I for all possible interleavings of the processes. A node S represents a point in some execution; at that point the algorithm is in configuration $S(I)$. It is easy to see that if all object types used by $\mathbf{A}$ are deterministic, there is a unique computation tree of $\mathbf{A}$ from I. If, however, some object types are nondeterministic, there may be more than one computation tree of $\mathbf{A}$ from I.

A computation tree of **A** from I is *full* if its set of nodes is the entire set of finite schedules of **A** that are applicable to I. The full computation tree of **A** from I is unique, regardless of whether the object types used by **A** are deterministic. If **A** uses nondeterministic object types, we can think of the different computation trees of **A** from I as obtained from the full computation tree of **A** from I by (repeatedly) applying the following pruning rule: If there are multiple edges out of a node S corresponding to steps of a single process P (because of the nondeterministic responses of the object accessed by P in the operation pending in $S(I)$), then some, but not all, of these edges may be pruned.

We use the following notation and terminology for computation trees. Let $\mathcal{T}$ be a computation tree of **A** from an initial configuration I, let S be a node in $\mathcal{T}$, and let P be a process of **A**. The subtree of $\mathcal{T}$ that is rooted at S is denoted *subtree*$(S, \mathcal{T})$, and we say that it is *full* if it is equal to *subtree*$(S, \mathcal{F})$, where $\mathcal{F}$ is the full computation tree of **A** from I. The set of nodes $\{ S \cdot e : S \cdot e \text{ is a node in } \mathcal{T} \text{ and } e \text{ is a step of } P \}$ is denoted *children*$(P, S, \mathcal{T})$; these are the nodes reached from S by a single step of process P. We say that S_1 is a *P-sibling* of S_2 in $\mathcal{T}$ if there is a node S in $\mathcal{T}$ so that $S_1, S_2 \in$ *children*$(P, S, \mathcal{T})$.

LEMMA 6.1. *Let* **A** *be an algorithm, and let $\mathcal{T}$ and $\mathcal{T}'$ be any computation trees of* **A** *from (not necessarily distinct) initial configurations I and I', respectively. Consider any nodes S_0 and $S_0 \cdot S$ in $\mathcal{T}$ and node S_1 in $\mathcal{T}'$ such that subtree$(S_1, \mathcal{T}')$ is full and every process that takes a step in S and every object is in the same state in $S_0(I)$ as in $S_1(I')$. Then $S_1 \cdot S$ is a node in $\mathcal{T}'$, and every process that takes a step in S and every object is in the same state in $S_0 \cdot S(I)$ as in $S_1 \cdot S(I')$.*

Proof. Let S^i be the schedule consisting of the first i steps of S for all $0 \leq i \leq |S|$. Using the fact that *subtree*$(S_1, \mathcal{T}')$ is full and every process that takes a step in S and every object is in the same state in $S_0(I)$ as in $S_1(I')$, a straightforward induction on i proves that S^i is applicable to $S_1(I')$, and every process that takes a step in S and every object is in the same state in $S_0 \cdot S^i(I)$ as in $S_1 \cdot S^i(I')$. The lemma follows from this, since $S = S^\ell$, where $\ell = |S|$. □

6.2. Valence of nodes. Let **A** be a consensus or equality negation algorithm for processes $P_0, P_1, \ldots, P_{n-1}$, where $n \geq 2$. Let I be an initial configuration of **A**, $\mathcal{T}$ be a computation tree of **A** from I, and S be a node in $\mathcal{T}$. For any $v \in \{0, 1\}$, we say that S is *v-valent in $\mathcal{T}$* if there is no descendant S' of S in $\mathcal{T}$ such that process P_0 decides $\overline{v}$ in $S'(I)$. S is *univalent in $\mathcal{T}$* if it is v-valent in $\mathcal{T}$ for some $v \in \{0, 1\}$. S is *bivalent in $\mathcal{T}$* if there are descendants S_0 and S_1 of S in $\mathcal{T}$ so that process P_0 decides 0 in $S_0(I)$ and it decides 1 in $S_1(I)$. (Note that in this definition it is the same process, P_0, whose decisions are 0 and 1 in $S_0(I)$ and $S_1(I)$, respectively.) Informally, a univalent node represents a point in an execution of **A** where the outcome (of consensus or equality negation) has been "sealed"—even if no process has actually decided yet. A bivalent node represents a point in an execution where both outcomes are still possible.

The *valence* of a node S in $\mathcal{T}$ refers to whether S is bivalent or univalent in $\mathcal{T}$ and, in the latter case, whether it is 0-valent or 1-valent. As noted in section 6.1, if **A** uses nondeterministic types, there may be many computation trees of **A** from I. Therefore, a node S that appears in two different computation trees may have a different valence in one than in the other, because the valence of S in a computation tree $\mathcal{T}$ depends on the subset of S's descendants in the *full* computation tree that are actually present in $\mathcal{T}$.

Let S_0 and S_1 be two univalent nodes in $\mathcal{T}$. We say that S_0 and S_1 *have the same valence* in $\mathcal{T}$ if they are both 0-valent or both 1-valent in $\mathcal{T}$. We say that they *have*

opposite valence in $\mathcal{T}$ if one is 0-valent and the other is 1-valent in $\mathcal{T}$.

The notions of v-valence, univalence, and bivalence for *configurations* of *consensus* algorithms were originally defined in [5] and subsequently used extensively in the literature. Although the above definitions are very much in the spirit of those in [5], there are some minor differences. We now explain the reasons for these differences. First, we define valence with respect to nodes of computation trees rather than configurations. This is because some of our results (in particular, those in section 7) are more conveniently proved using computation trees. A second, and more important, difference is the fact that we now wish to have a more general definition that encompasses both consensus and equality negation algorithms; in the past, valence was defined only in the context of consensus algorithms. The need for this greater generality underlies the other difference of our definition relative to the earlier ones: In the "classical" definition, a node is v-valent if there is no descendant of it in which *any* process (not P_0 in particular) decides $\overline{v}$. An analogous comment applies to the definition of bivalence. In the case of consensus, there is no difference between the two definitions, since the agreement property requires that any two processes that decide, decide the same value. In the case of equality negation, however, if the initial values of all processes are the same, then P_0 is supposed to decide a different value than the other processes. Thus, in this case, we should specify whether it is the decision of P_0 or the decision of the other processes that is taken to be the "outcome" of the computation. It is not important *which* of the two is chosen. In our definition we arbitrarily chose P_0.

6.3. Properties of computation trees. In this section we prove several properties of computation trees of any consensus or equality negation algorithm for two processes. These properties are used in sections 7 and 8. For the rest of this section, let $\mathbf{A}$ be a consensus or equality negation algorithm for processes P_0 and P_1, and I be any initial configuration of $\mathbf{A}$.

LEMMA 6.2. *Let $\mathcal{T}$ be any computation tree of $\mathbf{A}$ from initial configuration I, and let S be any node in $\mathcal{T}$. For each $k \in \{0,1\}$, there exists a solo schedule S_k of process P_k such that $S \cdot S_k$ is a node in $\mathcal{T}$ and P_k has decided in $S \cdot S_k(I)$.*

Proof. The proof is immediate from the termination requirement (of consensus or equality negation) and the definition of computation tree. □

By Lemma 6.2, every node S in $\mathcal{T}$ has a descendant S' such that process P_0 has decided in $S'(I)$. From the definition of valence it follows that S has exactly one of the three possible attributes: 0-valent, 1-valent, or bivalent. Therefore, the valence of each node in $\mathcal{T}$ is well defined.

LEMMA 6.3. *Let $\mathcal{T}$ be any computation tree of $\mathbf{A}$ from initial configuration I. Let S_0 and S_1 be any univalent nodes in $\mathcal{T}$ such that some process has decided the same value in $S_0(I)$ as in $S_1(I)$. Then S_0 and S_1 have the same valence in $\mathcal{T}$.*

Proof. Observe that if a process decides a value, say, v, in a node S of $\mathcal{T}$, then the initial configuration I and v uniquely determine the value w that P_0 can decide in any descendant of S in $\mathcal{T}$. (If $\mathbf{A}$ is a consensus algorithm, then $w = v$ regardless of the initial configuration. If $\mathbf{A}$ is an equality negation algorithm, then $w = v$ (respectively, $w = \overline{v}$) if the initial values of P_0 and P_1 in I are different (respectively, the same).)

Let P_k for some $k \in \{0,1\}$ be the process that has decided the same value, say, v, in $S_0(I)$ as in $S_1(I)$. Let S_0' and S_1' be descendants of S_0 and S_1, respectively, such that P_0 has decided in $S_0'(I)$ and $S_1'(I)$ (S_0' and S_1' exist by Lemma 6.2). By the preceding observation, P_0 decides the same value, say, w, in $S_0'(I)$ as in $S_1'(I)$. Recalling that S_0 and S_1 are both univalent, from the definition of w-valent node,

it follows that S_0 and S_1 are both w-valent in $\mathcal{T}$. Thus, S_0 and S_1 have the same valence in $\mathcal{T}$. □

A computation tree $\mathcal{T}$ of $\mathbf{A}$ is 1-*full* if every subtree rooted at a 1-valent node in $\mathcal{T}$ is full; in particular, every full computation tree of $\mathbf{A}$ is 1-full.

LEMMA 6.4. *Let $\mathcal{T}$ be any 1-full computation tree of $\mathbf{A}$ from initial configuration I. Let S_0 and S_1 be any univalent nodes in $\mathcal{T}$ such that the state of some process and each object is the same in $S_0(I)$ as in $S_1(I)$. Then, S_0 and S_1 have the same valence in $\mathcal{T}$.*

Proof. Suppose, for contradiction, that S_0 and S_1 have opposite valence. Without loss of generality, assume that S_0 is 0-valent and S_1 is 1-valent in $\mathcal{T}$. Let P be the process whose state in $S_0(I)$ is the same as in $S_1(I)$. By Lemma 6.2 (applied to node S_0 in $\mathcal{T}$), there exists a solo schedule S of P such that $S_0 \cdot S$ is a node in $\mathcal{T}$ and P has decided in $S_0 \cdot S(I)$. Since $\mathcal{T}$ is 1-full and S_1 is 1-valent in $\mathcal{T}$, it follows that *subtree*$(S_1, \mathcal{T})$ is full. By hypothesis, the state of process P and each object is the same in $S_0(I)$ as in $S_1(I)$. Since S is a solo schedule of P, by Lemma 6.1 (applied for $I' = I$), $S_1 \cdot S$ is a node in $\mathcal{T}$ and P has the same state in $S_0 \cdot S(I)$ as in $S_1 \cdot S(I)$. Thus, P decides the same value in $S_0 \cdot S(I)$ as in $S_1 \cdot S(I)$. By Lemma 6.3, $S_0 \cdot S$ and $S_1 \cdot S$ have the same valence in $\mathcal{T}$. This contradicts our assumption that S_0 and S_1 have opposite valence in $\mathcal{T}$. □

Let $\mathcal{T}$ be any computation tree of $\mathbf{A}$ from initial configuration I, let S be any node in $\mathcal{T}$, and let e_0 and e_1 be any steps of $\mathbf{A}$. We say e_0 *and* e_1 *commute at* S *in* $\mathcal{T}$ if both $S \cdot e_0 \cdot e_1$ and $S \cdot e_1 \cdot e_0$ are nodes in $\mathcal{T}$ and the state of each process and each object is the same in $S \cdot e_0 \cdot e_1(I)$ as in $S \cdot e_1 \cdot e_0(I)$. We say that e_k *overwrites* $e_{\overline{k}}$ *at* S *in* $\mathcal{T}$ for some $k \in \{0, 1\}$ if both $S \cdot e_k$ and $S \cdot e_{\overline{k}} \cdot e_k$ are nodes in $\mathcal{T}$ and the state of P_k and each object is the same in $S \cdot e_k(I)$ as in $S \cdot e_{\overline{k}} \cdot e_k(I)$.

LEMMA 6.5. *Let $\mathcal{T}$ be any 1-full computation tree of $\mathbf{A}$ from initial configuration I. Let S be any node in $\mathcal{T}$, and let e_0 and e_1 be any steps of $\mathbf{A}$ such that $S \cdot e_0$ and $S \cdot e_1$ are univalent nodes of opposite valence in $\mathcal{T}$. Then*

(a) *e_0 and e_1 do not commute at S in $\mathcal{T}$; and*

(b) *neither e_0 nor e_1 overwrites the other at S in $\mathcal{T}$.*

Proof. Without loss of generality, assume that $S \cdot e_0$ is 0-valent and $S \cdot e_1$ is 1-valent in $\mathcal{T}$.

(a) Suppose, for contradiction, that e_0 and e_1 commute at S in $\mathcal{T}$. Then, by definition, $S \cdot e_0 \cdot e_1$ and $S \cdot e_1 \cdot e_0$ are nodes in $\mathcal{T}$ and the state of each process and object is the same in $S \cdot e_0 \cdot e_1(I)$ as in $S \cdot e_1 \cdot e_0(I)$. By Lemma 6.4, $S \cdot e_0 \cdot e_1$ and $S \cdot e_1 \cdot e_0$ have the same valence in $\mathcal{T}$. This contradicts the fact that $S \cdot e_0 \cdot e_1$ and $S \cdot e_1 \cdot e_0$ have opposite valence in $\mathcal{T}$ (because $S \cdot e_0$ and $S \cdot e_1$ do).

(b) Suppose, for contradiction, that e_k overwrites $e_{\overline{k}}$ at S in $\mathcal{T}$ for some $k \in \{0, 1\}$. Then, by definition, $S \cdot e_k$ and $S \cdot e_{\overline{k}} \cdot e_k$ are nodes in $\mathcal{T}$ and the state of P_k and each object is the same in $S \cdot e_k(I)$ as in $S \cdot e_{\overline{k}} \cdot e_k(I)$. By Lemma 6.4, $S \cdot e_k$ and $S \cdot e_{\overline{k}} \cdot e_k$ have the same valence in $\mathcal{T}$. This contradicts the fact that $S \cdot e_k$ is k-valent and $S \cdot e_{\overline{k}} \cdot e_k$ is $\overline{k}$-valent in $\mathcal{T}$ (because $S \cdot e_{\overline{k}}$ is). □

LEMMA 6.6. *Let $\mathcal{F}$ be any full computation tree of $\mathbf{A}$. Let S be any node in $\mathcal{F}$, and let e_0 and e_1 be any steps of processes P_0 and P_1, respectively, such that $S \cdot e_0$ and $S \cdot e_1$ are univalent nodes of opposite valence in $\mathcal{F}$. Then, e_0 and e_1 access the same object.*

Proof. Suppose, for contradiction, that e_0 and e_1 access different objects. Then e_1 is applicable to $S \cdot e_0(I)$ and e_0 is applicable to $S \cdot e_1(I)$. Since $\mathcal{F}$ is a full computation tree, $S \cdot e_0 \cdot e_1$ and $S \cdot e_1 \cdot e_0$ are nodes in $\mathcal{F}$. Furthermore, each process and object has

the same state in $S \cdot e_0 \cdot e_1(I)$ as in $S \cdot e_1 \cdot e_0(I)$. Thus, e_0 and e_1 commute at S in $\mathcal{F}$. This contradicts Lemma 6.5(a) (because $\mathcal{F}$ is full and hence 1-full). □

Let $\mathcal{T}$ be any computation tree of $\mathbf{A}$, and S be any node in $\mathcal{T}$. We say that S is a *critical node* in $\mathcal{T}$ if S is bivalent and all of its children are univalent in $\mathcal{T}$.

LEMMA 6.7. *Let $\mathcal{T}$ be any computation tree of $\mathbf{A}$ with a bivalent root. Then $\mathcal{T}$ has a critical node.*

Proof. Suppose, for contradiction, that $\mathcal{T}$ has no critical node. Then every bivalent node in $\mathcal{T}$ has at least one child that is also bivalent in $\mathcal{T}$. Thus, there is an *infinite* path S of $\mathcal{T}$ that starts at the (bivalent) root of $\mathcal{T}$ and consists entirely of bivalent nodes. Consider any node S' on this path; since S' is bivalent, no process has decided in $S'(I)$. Thus, there is an infinite schedule S of $\mathbf{A}$ that is applicable to I such that, for any prefix S' of S, no process has decided in $S'(I)$. Since S is infinite, at least one process takes infinitely many steps in S and therefore is correct in S. As we have just shown, however, in every prefix S' of S, no process—and, in particular, no correct process in S—decides in $S'(I)$. This contradicts the fact that $\mathbf{A}$ is a consensus algorithm or an equality negation algorithm (in particular, it contradicts the termination property of $\mathbf{A}$). □

7. Type negation has consensus number one. In this section we prove that type negation has consensus number one—i.e., that consensus for two processes cannot be solved using only negation objects and registers.

7.1. The bivalence argument and some limitations. There is a standard technique for proving that there is no consensus algorithm for n processes using a certain set of object types, known as "the bivalence argument." This technique goes back to the ground-breaking result of Fischer, Lynch, and Paterson [5] for the asynchronous message-passing model and was further adapted by others, e.g., [3, 6, 13], for the asynchronous read/write-memory model. In this subsection, we sketch the main ingredients of this technique for two processes and call attention to a difficulty that arises when it is applied to nondeterministic object types.

The bivalence argument employs proof by contradiction. It begins by assuming that there is a consensus algorithm $\mathbf{A}_c$ for processes P_0 and P_1 that uses only objects of some given types. Consider the full computation tree $\mathcal{F}$ of this algorithm from an initial configuration I in which the two processes have different initial values. The root of $\mathcal{F}$ is bivalent because, by validity, in a solo schedule a process can decide only its own initial value. Thus, by Lemma 6.7, $\mathcal{F}$ has a critical node S. By definition, some of the (univalent) children of the critical node S must be 0-valent and some must be 1-valent in $\mathcal{F}$. If the object types used by $\mathbf{A}_c$ are deterministic, then S has exactly two children in $\mathcal{F}$, say, $S \cdot e_0$ and $S \cdot e_1$, where e_k is a step of process P_k for $k \in \{0, 1\}$. Thus, in this case, we can say that *the outcome of consensus after S is determined by which one of the two processes takes the next step.* By Lemma 6.6, e_0 and e_1 both access the same object. Finally, by a case analysis involving the types of operations applied in steps e_0 and e_1, the following is shown: There exist schedules S_0 and S_1 such that (i) both $S \cdot e_0 \cdot S_0$ and $S \cdot e_1 \cdot S_1$ are nodes in $\mathcal{F}$, and (ii) some process has decided the same value in $S \cdot e_0 \cdot S_0(I)$ and $S \cdot e_1 \cdot S_1(I)$. This contradicts the fact that the outcome of consensus is determined by which of the two processes takes the next step after S.

This type of argument runs into difficulties when applied to algorithms that use *nondeterministic* types. Specifically, we can no longer assert that at the critical node S, the outcome of consensus is determined by which of the two processes takes the next step. Since the object types are nondeterministic, it is possible that *each* of the

For $k, v \in \{0,1\}$, $e_k^v = (P_k, \text{NEGATE}(k,k), O, v)$.

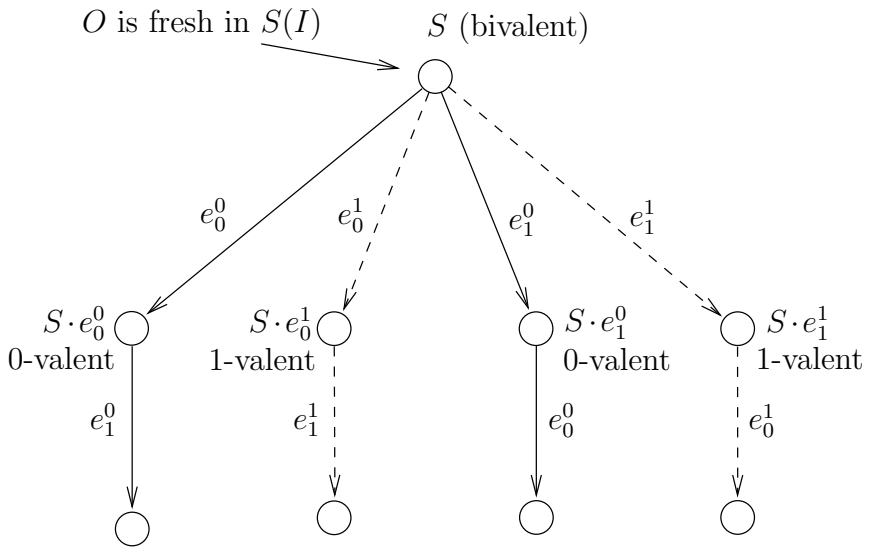


FIG. 7.1. *A critical node in* $\mathcal{F}$.

two processes can lead to both a 0-valent node and a 1-valent node, depending on the response of the object accessed.

To appreciate this point in more concrete terms, suppose that the consensus algorithm $\mathbf{A}_c$ uses negation objects and that the critical node S is such that, in configuration $S(I)$, each process P_k has a NEGATE(k,k) operation pending to a negation object O in the fresh state. Recall that a NEGATE operation applied to a fresh negation object can return either 0 or 1. Suppose, further, that each process P_k decides the value returned by the NEGATE(k,k) operation (see Figure 7.1). Now the step of process P_k that causes it to decide v, namely, e_k^v, is not applicable after the other process, $P_{\overline{k}}$, has applied the step that causes it to decide the opposite value, $\overline{v}$, for all $k \in \{0,1\}$ and all $v \in \{0,1\}$. This means that the we cannot derive a contradiction in the style of the bivalence argument.

Despite this, it turns out that the negation type is not powerful enough to solve consensus between two processes. To prove this, we use a modified version of the bivalence argument, based on a restricted computation tree. This is described in the following subsection.

7.2. The pruned computation tree. In the remainder of section 7, we assume, for contradiction, that $\mathbf{A}_c$ is a consensus algorithm for processes P_0 and P_1 that uses only negation objects and registers. Let I be an initial configuration of $\mathbf{A}_c$ in which P_0 and P_1 have different initial values, $\mathcal{T}$ be any computation tree of $\mathbf{A}_c$ from I, and $\mathcal{F}$ be the full computation tree of $\mathbf{A}_c$ from I. By the definition of type negation and the definition of computation tree, it is easy to see that $1 \leq |\mathit{children}(P, S, \mathcal{T})| \leq 2$ for every node S in $\mathcal{T}$ and each process P.

Consider a nonroot node S in $\mathcal{T}$ and let P be the process that takes the last step in S. We say that S is an *avoidable* 1*-valent node* in $\mathcal{T}$ if S is 1-valent in $\mathcal{F}$ and has a P-sibling S' in $\mathcal{F}$ such that S' is not 1-valent in $\mathcal{F}$. Intuitively, an avoidable 1-valent

node S in $\mathcal{F}$ is reached from its (bivalent) parent, $\hat{S}$, by a step of some process P; that step seals the outcome of consensus to be 1. Furthermore, S has a P-sibling S' that is not 1-valent; this means that the object accessed by the operation of P that is pending in $\hat{S}(I)$ *could* have returned a value that would leave open the possibility of the outcome of consensus being 0. In other words, starting from configuration $\hat{S}(I)$, outcome 1 could be avoided by a fortuitous choice of the value returned by the object accessed by the operation of process P that is pending in $\hat{S}(I)$. As a concrete illustration, the computation tree shown in Figure 7.1 contains two avoidable 1-valent nodes, namely, $S \cdot e_0^1$ and $S \cdot e_1^1$.

The *pruned computation tree of* $\mathbf{A}_c$ *from* I, denoted $\mathcal{L}$, is the tree obtained from the *full* computation tree $\mathcal{F}$ of $\mathbf{A}_c$ from I by pruning all its avoidable 1-valent nodes—i.e., removing them and their descendants. $\mathcal{L}$ is, in fact, a computation tree: If a node S in $children(P, \hat{S}, \mathcal{F})$ is pruned, then some P-sibling of S is not pruned. In other words, $\hat{S}$ has at least one child $\hat{S} \cdot e$ in $\mathcal{L}$ where e is a step of P. Thus, the pruning of S will not inhibit P from taking the next step after $\hat{S}$. Intuitively, pruning the avoidable 1-valent nodes from the full computation tree amounts to restricting the nondeterministic behavior of objects: Whenever an object has two responses to an operation, one of which forecloses the possibility of outcome 0 while the other leaves the possibility of outcome 0 open, we restrict the behavior of the object by choosing the latter response.

The following lemmas establish some basic properties of the pruned computation tree $\mathcal{L}$.

LEMMA 7.1. *The root of* $\mathcal{L}$ *is bivalent.*

Proof. Recall that $\mathcal{L}$ is a computation tree of $\mathbf{A}_c$ from initial configuration I in which the two processes have different initial values. Without loss of generality, assume that P_0 has initial value 0 and P_1 has initial value 1 in I. By Lemma 6.2 (applied to the root, $S_\perp$, of $\mathcal{L}$), for each $k \in \{0, 1\}$ there exists a solo schedule S_k of P_k such that $S_\perp \cdot S_k = S_k$ is a node in $\mathcal{L}$ and P_k has decided in $S_k(I)$. Since P_k has initial value k in I and S_k is a solo schedule of P_k, by validity, P_k must decide k in $S_k(I)$. By Lemma 6.2 (applied to node S_1 of $\mathcal{L}$), there is a solo schedule S_0' of P_0 such that $S_1 \cdot S_0'$ is a node in $\mathcal{L}$ and P_0 has decided in $S_1 \cdot S_0'(I)$. Since P_1 has decided 1 in $S_1(I)$ and the decision is irrevocable, P_1 has decided 1 in $S_1 \cdot S_0'(I)$. By agreement, P_0 has also decided 1 in $S_1 \cdot S_0'(I)$. Thus, the root of $\mathcal{L}$ has two descendants, S_0 and $S_1 \cdot S_0'$ so that P_0 has decided 0 and 1, respectively, in $S_0(I)$ and $S_1 \cdot S_0'(I)$. Thus, the root of $\mathcal{L}$ is bivalent. □

LEMMA 7.2. *Let* S *be any node in the full computation tree* $\mathcal{F}$. *Then*

(a) S *is not a node in* $\mathcal{L}$ *if and only if* S *is a descendant of an avoidable* 1*-valent node in* $\mathcal{F}$. *(By convention, a node is a descendant of itself.)*
(b) *If* S *is* 1*-valent in* $\mathcal{F}$ *and is also in* $\mathcal{L}$, *then* S *is* 1*-valent in* $\mathcal{L}$.
(c) *If* S *is* 0*-valent in* $\mathcal{F}$, *then it is in* $\mathcal{L}$ *and is* 0*-valent in* $\mathcal{L}$.
(d) *If* S *is in* $\mathcal{L}$ *and* $children(P, S, \mathcal{F}) = \{S'\}$, *then* S' *is in* $\mathcal{L}$.

Proof. Immediate consequences of the definition of $\mathcal{L}$. □

LEMMA 7.3. $\mathcal{L}$ *is* 1*-full.*

Proof. We must show that every subtree rooted at a 1-valent node in $\mathcal{L}$ is full. Consider any 1-valent node S in $\mathcal{L}$. S cannot have any 0-valent descendants in $\mathcal{F}$: by Lemma 7.2(c), any such descendants would exist in $\mathcal{L}$ and be 0-valent in $\mathcal{L}$, contradicting that S is 1-valent in $\mathcal{L}$. Thus, S is 1-valent in $\mathcal{F}$. Consequently, all nodes in $subtree(S, \mathcal{F})$ are 1-valent, and therefore none of them can be an avoidable 1-valent node in $\mathcal{F}$. Hence, by Lemma 7.2(a), all nodes in $subtree(S, \mathcal{F})$ are in $\mathcal{L}$, proving that

subtree$(S, \mathcal{L}) =$ *subtree*$(S, \mathcal{F})$. Therefore, *subtree*$(S, \mathcal{L})$ is full. □

LEMMA 7.4. *Let S be any node in $\mathcal{L}$ and P be any process such that* $|$*children*$(P, S, \mathcal{F})| = 2$, *each node in* *children*$(P, S, \mathcal{L})$ *is univalent, and one of them is* 1*-valent in* $\mathcal{L}$. *Then* *children*$(P, S, \mathcal{L}) =$ *children*$(P, S, \mathcal{F})$ *and both nodes in* *children*$(P, S, \mathcal{L})$ *are* 1*-valent in* $\mathcal{L}$.

Proof. Let *children*$(P, S, \mathcal{F}) = \{S', S''\}$. By the hypothesis of the lemma we may assume, without loss of generality, that $S' \in$ *children*$(P, S, \mathcal{L})$ and S' is 1-valent in $\mathcal{L}$. It remains to show that S'' is also in $\mathcal{L}$ and is 1-valent in $\mathcal{L}$.

By Lemma 7.3, $\mathcal{L}$ is 1-full. Since S' is 1-valent in $\mathcal{L}$, *subtree*$(S', \mathcal{L}) =$ *subtree*$(S', \mathcal{F})$ and thus S' is 1-valent in $\mathcal{F}$. Since S' is in $\mathcal{L}$, by Lemma 7.2(a) S' is not a proper descendant of an avoidable 1-valent node in $\mathcal{F}$. Therefore, as S'' is a P-sibling of S' in $\mathcal{F}$, S'' is not a proper descendant of an avoidable 1-valent node in $\mathcal{F}$ either. Also, S'' is not an avoidable 1-valent node in $\mathcal{F}$, because its only P-sibling in $\mathcal{F}$, namely, S', is a 1-valent node in $\mathcal{F}$. By Lemma 7.2(a), S'' is in $\mathcal{L}$. Furthermore, S'' must be a 1-valent node in $\mathcal{F}$. If not, then by definition S' would be an avoidable 1-valent node in $\mathcal{F}$ and, by Lemma 7.2(a), S' would not be in $\mathcal{L}$—a contradiction. Since S'' is 1-valent in $\mathcal{F}$ and is also in $\mathcal{L}$, by Lemma 7.2(b), S'' is 1-valent in $\mathcal{L}$. □

7.3. The impossibility result. The proof that no algorithm can solve consensus between two processes using only negation objects and registers is a bivalence argument based on the pruned computation tree $\mathcal{L}$. The only remaining tricky point is to ensure that the computation tree nodes that exhibit the contradictory behavior required by the bivalence argument have not been pruned away!

By Lemmas 6.7 and 7.1, $\mathcal{L}$ has a critical node E. By definition of a critical node, there are steps e_0 and e_1 of processes P_0 and P_1, respectively, such that $E \cdot e_0$ and $E \cdot e_1$ are univalent nodes of opposite valence in $\mathcal{L}$. Without loss of generality, suppose that $E \cdot e_0$ is 0-valent and $E \cdot e_1$ is 1-valent in $\mathcal{L}$. (Otherwise, we rename e_0 and e_1 and apply the same argument.) To derive a contradiction, we first show that e_0 and e_1 access the same object (Lemma 7.5). Then we prove that the object accessed in e_0 and e_1 cannot be a register (Lemma 7.6). Finally, we prove that the object cannot be a negation object (Corollary 7.8 and Lemma 7.9).

LEMMA 7.5. *Steps e_0 and e_1 access the same object.*

Proof. Suppose, for contradiction, that e_0 and e_1 access, respectively, objects O_0 and O_1, where $O_0 \neq O_1$. Then, e_1 is applicable to $E \cdot e_0(I)$ and e_0 is applicable to $E \cdot e_1(I)$ and, in fact, $E \cdot e_0 \cdot e_1(I) = E \cdot e_1 \cdot e_0(I)$. Thus $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are nodes in $\mathcal{F}$. Since $\mathcal{L}$ is 1-full (by Lemma 7.3) and $E \cdot e_1$ is 1-valent in $\mathcal{L}$, *subtree*$(E \cdot e_1, \mathcal{L}) =$ *subtree*$(E \cdot e_1, \mathcal{F})$. Since $E \cdot e_1 \cdot e_0$ is in $\mathcal{F}$, it follows that it is also in $\mathcal{L}$. We shall prove that $E \cdot e_0 \cdot e_1$ is also a node in $\mathcal{L}$. This fact implies that e_0 and e_1 commute at E in $\mathcal{L}$, contrary to Lemma 6.5(a).

To show that $E \cdot e_0 \cdot e_1$ is in $\mathcal{L}$, we consider two cases, depending on whether *children*$(P_1, E, \mathcal{F})$ has one or two nodes.

Case 1. $|$*children*$(P_1, E, \mathcal{F})| = 1$: Then that unique child has to be $E \cdot e_1$ (since $E \cdot e_1$ is a node in $\mathcal{F}$). Since $O_0 \neq O_1$, each of P_1 and O_1 is in the same state in $E(I)$ as in $E \cdot e_0(I)$. It follows that e_1 is the only step of P_1 that is applicable to $E \cdot e_0(I)$; thus, *children*$(P_1, E \cdot e_0, \mathcal{F}) = \{E \cdot e_0 \cdot e_1\}$. By Lemma 7.2(d), $E \cdot e_0 \cdot e_1$ is in $\mathcal{L}$.

Case 2. $|$*children*$(P_1, E, \mathcal{F})| = 2$: Recall that all nodes in *children*$(P_1, E, \mathcal{L})$ are univalent. Recall also that, by assumption, $E \cdot e_1$ is in *children*$(P_1, E, \mathcal{L})$ and is 1-valent in $\mathcal{L}$. Since $|$*children*$(P_1, E, \mathcal{F})| = 2$, by Lemma 7.4, *children*$(P_1, E, \mathcal{F}) =$ *children*$(P_1, E, \mathcal{L})$ and both nodes in *children*$(P_1, E, \mathcal{L})$ are actually 1-valent in $\mathcal{L}$. Let the two nodes in *children*$(P_1, E, \mathcal{L})$ be $E \cdot e_1$ and $E \cdot e_1'$. Because each of P_1 and O_1 is

in the same state in $E(I)$ as in $E \cdot e_0(I)$, both e_1 and e_1' are applicable to $E \cdot e_0(I)$ and access the same object. Therefore, it must be that $E \cdot e_0 \cdot e_1$ or $E \cdot e_0 \cdot e_1'$ is in $\mathcal{L}$. Since $E \cdot e_1$ and $E \cdot e_1'$ are 1-valent in $\mathcal{L}$ and since e_1 was chosen arbitrarily as a 1-valent node in *children*$(P_1, E, \mathcal{L})$, we may assume without loss of generality that $E \cdot e_0 \cdot e_1$ is in $\mathcal{L}$. (Otherwise, we repeat the argument with e_1' instead of e_1.) □

In the remainder of this section, let $\mathcal{O}$ denote the object accessed in both e_0 and e_1.

LEMMA 7.6. *$\mathcal{O}$ is not a register.*

Proof. Assume, by way of contradiction, that $\mathcal{O}$ is a register. Since register is a deterministic type, *children*$(P_0, E, \mathcal{F})$ and *children*$(P_1, E, \mathcal{F})$ contain a single node each. Let op_0 and op_1 be the (READ or WRITE) operations applied to $\mathcal{O}$ in steps e_0 and e_1, respectively. There are two cases.

Case 1. *op_k is a* WRITE *operation for some $k \in \{0, 1\}$*: Let v be the value written into $\mathcal{O}$ by process P_k in step e_k. Since P_k has the same state in $E(I)$ as in $E \cdot e_{\overline{k}}(I)$, e_k is also applicable to $E \cdot e_{\overline{k}}(I)$, and therefore *children*$(P_k, E \cdot e_{\overline{k}}, \mathcal{F}) = \{E \cdot e_{\overline{k}} \cdot e_k\}$. By Lemma 7.2(d), $E \cdot e_{\overline{k}} \cdot e_k$ is in $\mathcal{L}$. Clearly, $\mathcal{O}$ has value v in both $E \cdot e_k(I)$ and $E \cdot e_{\overline{k}} \cdot e_k(I)$. In addition, each other object and process P_k has the same state in $E \cdot e_k(I)$ as in $E \cdot e_{\overline{k}} \cdot e_k(I)$. But then e_k overwrites $e_{\overline{k}}$ at E in $\mathcal{L}$. This contradicts Lemma 6.5(b).

Case 2. *op_0 and op_1 are both* READ *operations:* Let v be the value of $\mathcal{O}$ in configuration $E(I)$. Since the value of a register is not changed by READ operations, $\mathcal{O}$ has the same value v in $E \cdot e_0(I)$ and $E \cdot e_1(I)$. It follows that e_0 and e_1 are applicable to $E \cdot e_1(I)$ and $E \cdot e_0(I)$, respectively. Thus *children*$(P_0, E \cdot e_0, \mathcal{F}) = \{E \cdot e_0 \cdot e_1\}$ and *children*$(P_1, E \cdot e_1, \mathcal{F}) = \{E \cdot e_1 \cdot e_0\}$. By Lemma 7.2(d), $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are nodes in $\mathcal{L}$ as well. Clearly, the state of each process and each object in $E \cdot e_0 \cdot e_1(I)$ is the same as in $E \cdot e_1 \cdot e_0(I)$. But then e_0 and e_1 commute at E in $\mathcal{L}$. This contradicts Lemma 6.5(a). □

Since $\mathcal{O}$ is not a register and the assumed consensus algorithm $\mathbf{A}_c$ uses only registers and negation objects, $\mathcal{O}$ must be of type negation. We shall now prove that this is not the case by proving that the state of $\mathcal{O}$ in $E(I)$ is not a legitimate state of type negation. We say that $\mathcal{O}$ is *fresh* in a configuration C of $\mathbf{A}_c$ if its state in C is ☺ and *upset* in C if its state in C is ☹.

For each $k \in \{0, 1\}$, let $j_k \in \{0, 1\}$ and $w_k \in \{0, 1, 2\}$ be the process and initial-value parameters, respectively, of the NEGATE operation to $\mathcal{O}$ of process P_k that is pending in $E(I)$. In other words, the operation to $\mathcal{O}$ of P_k pending in $E(I)$ is NEGATE(j_k, w_k). Recall that the response of a NEGATE operation is 0 or 1. For each $k, u \in \{0, 1\}$, let $d_k^u = (P_k, \text{NEGATE}(j_k, w_k), \mathcal{O}, u)$—i.e., d_k^u is a step in which P_k applies NEGATE(j_k, w_k) to $\mathcal{O}$ and receives u as response. Thus, $e_k \in \{d_k^0, d_k^1\}$ for each $k \in \{0, 1\}$.

The next lemma states that the application of the two NEGATE operations pending in $E(I)$ does not upset $\mathcal{O}$.

LEMMA 7.7. *For each $k \in \{0, 1\}$, let d_k be any step in $\{d_k^0, d_k^1\}$ such that $E \cdot e_{\overline{k}} \cdot d_k$ is a node in $\mathcal{F}$. Then $\mathcal{O}$ is not upset in $E \cdot e_{\overline{k}} \cdot d_k(I)$.*

Proof. Suppose, for contradiction, that $\mathcal{O}$ is upset in $E \cdot e_{\overline{k}} \cdot d_k(I)$ for some $k \in \{0, 1\}$. We first show that $\mathcal{O}$ is also upset in $E \cdot e_k \cdot d_{\overline{k}}(I)$. Since $\mathcal{O}$ is upset in $E \cdot e_{\overline{k}} \cdot d_k(I)$, by the specification of negation, it is easy to verify that one of the following conditions must hold (in verifying this, recall that e_0 and d_0 are NEGATE(j_0, w_0) steps to $\mathcal{O}$, and e_1 and d_1 are NEGATE(j_1, w_1) steps to $\mathcal{O}$):

- $\mathcal{O}$ is upset in $E(I)$.

- $\mathcal{O}$ is fresh in $E(I)$, and either
  (a) $j_0 = j_1 = 0$, or
  (b) $j_0 = j_1 = 1$ and $w_0 \neq w_1$.
- The state of $\mathcal{O}$ in $E(I)$ is of the form $\langle v_0, v_1, u_0, u_1\rangle$ and either
  (a) $v_0 \neq \perp$ and, for some $\ell \in \{0,1\}$, $j_\ell = 0$, or
  (b) $v_1 \neq \perp$ and, for some $\ell \in \{0,1\}$, $j_\ell = 1$ and $w_\ell \neq v_1$.

In each of these cases, it can be verified that $\mathcal{O}$ is also upset in $E\cdot e_k \cdot d_{\overline{k}}(I)$.

Since $\mathcal{O}$ is upset in $E\cdot e_1\cdot d_0(I)$, it is easy to verify that the step $d_0' \in \{d_0^0, d_0^1\}\setminus\{d_0\}$ is also applicable to $E\cdot e_1(I)$ and $\mathcal{O}$ is upset in $E\cdot e_1\cdot d_0'(I)$ as well. Therefore, *children*$(P_0, E\cdot e_1, \mathcal{F}) = \{E\cdot e_1\cdot d_0^0, E\cdot e_1\cdot d_0^1\}$, and $\mathcal{O}$ is upset in both $E\cdot e_1\cdot d_0^0(I)$ and $E\cdot e_1\cdot d_0^1(I)$. Likewise, as $\mathcal{O}$ is upset in $E\cdot e_0\cdot d_1(I)$, *children*$(P_1, E\cdot e_0, \mathcal{F}) = \{E\cdot e_0\cdot d_1^0, E\cdot e_0\cdot d_1^1\}$, and $\mathcal{O}$ is upset in both $E\cdot e_0\cdot d_1^0(I)$ and $E\cdot e_0\cdot d_1^1(I)$. Since $\mathcal{L}$ is 1-full (by Lemma 7.3) and $E\cdot e_1$ is 1-valent in $\mathcal{L}$, it follows that both $E\cdot e_1\cdot d_0^0$ and $E\cdot e_1\cdot d_0^1$ are nodes in $\mathcal{L}$. Furthermore, they are 1-valent in $\mathcal{L}$ because $E\cdot e_1$ is. Therefore, as $e_0 \in \{d_0^0, d_0^1\}$, $E\cdot e_1\cdot e_0$ is a 1-valent node in $\mathcal{L}$. Since $\mathcal{L}$ is a computation tree, there exists $d \in \{d_1^0, d_1^1\}$ such that $E\cdot e_0\cdot d$ is a node in $\mathcal{L}$. This node is 0-valent in $\mathcal{L}$ because $E\cdot e_0$ is. As we have just shown, $\mathcal{O}$ is upset in $E\cdot e_0\cdot d(I)$. Therefore, the state of P_0 and each object is the same in $E\cdot e_1\cdot e_0(I)$ as in $E\cdot e_0\cdot d(I)$. By Lemma 6.4, $E\cdot e_1\cdot e_0$ and $E\cdot e_0\cdot d$ have the same valence in $\mathcal{L}$. This contradicts the fact that $E\cdot e_1\cdot e_0$ is 1-valent and $E\cdot e_0\cdot d$ is 0-valent in $\mathcal{L}$.

We reached this contradiction by assuming $\mathcal{O}$ is upset in $E\cdot e_{\overline{k}}\cdot d_k$ for some $k \in \{0,1\}$. Therefore, $\mathcal{O}$ is not upset in $E\cdot e_0\cdot d_1$ or in $E\cdot e_1\cdot d_0$. □

Since once it has become upset, a negation object remains upset forever, we have the following.

COROLLARY 7.8. *$\mathcal{O}$ is not upset in $E(I)$, $E\cdot e_0(I)$, or $E\cdot e_1(I)$.*

We now rule out the remaining possibilities for the state of $\mathcal{O}$ in $E(I)$.

LEMMA 7.9. *$\mathcal{O}$ is neither fresh nor in a state of the form $\langle v_0, v_1, u_0, u_1\rangle$ in $E(I)$.*

Proof. Suppose, for contradiction, that $\mathcal{O}$ is either fresh or in a state of the form $\langle v_0, v_1, u_0, u_1\rangle$ in $E(I)$. There are two cases, depending on the values of j_0 and j_1 (the process parameters of the two NEGATE operations pending in $E(I)$).

Case 1. *For some $k \in \{0,1\}$, $j_k = 0$*: If $j_{\overline{k}} = 0$, then both $d_{\overline{k}}^0$ and $d_{\overline{k}}^1$ are applicable to $E\cdot e_k(I)$, and $\mathcal{O}$ is upset in both $E\cdot e_k\cdot d_{\overline{k}}^0(I)$ and $E\cdot e_k\cdot d_{\overline{k}}^1(I)$, contrary to Lemma 7.7. Thus, $j_{\overline{k}} = 1$. We distinguish two subcases.

Subcase 1(a). *$\mathcal{O}$ is fresh in $E(I)$*: Since $\mathcal{O}$ is fresh in $E(I)$, by the specification of negation, both d_1^0 and d_1^1 are applicable to $E(I)$. Thus, $E\cdot d_1^0$ and $E\cdot d_1^1$ are in $\mathcal{F}$. Since E is a critical node in $\mathcal{L}$, by definition, every node in *children*$(P_1, E, \mathcal{L})$ is univalent. Recall that $E\cdot e_1$ is 1-valent. By Lemma 7.4, both $E\cdot d_1^0$ and $E\cdot d_1^1$ are nodes in $\mathcal{L}$ and are 1-valent in $\mathcal{L}$.

Let $u \in \{0,1\}$ be such that $e_0 = d_0^u$. We further distinguish two cases, depending on whether the initial-value parameters, w_0 and w_1, of the two pending NEGATE operations in $E(I)$ are equal or not:

(i) $w_0 \neq w_1$: By the specification of negation, e_0 is the only step of P_0 that is applicable to $E\cdot d_1^u(I)$, and d_1^u is the only step of P_1 that is applicable to $E\cdot e_0(I)$. Therefore, by Lemma 7.2(d), $E\cdot d_1^u\cdot e_0$ and $E\cdot e_0\cdot d_1^u$ are in $\mathcal{L}$. Furthermore, $E\cdot d_1^u\cdot e_0$ is 1-valent in $\mathcal{L}$ (because, as we just showed, $E\cdot d_1^u$ is); and $E\cdot e_0\cdot d_1^u$ is 0-valent in $\mathcal{L}$ (because $E\cdot e_0$ is). By the specification of negation, it can be checked that $\mathcal{O}$ is in the same state in $E\cdot e_0\cdot d_1^u(I)$ as in $E\cdot d_1^u\cdot e_0(I)$ (namely, in state $\langle w_0, w_1, u, u\rangle$ if $j_0 = 0$, and in state $\langle w_1, w_0, u, u\rangle$ if $j_0 = 1$). Therefore, $E\cdot e_0\cdot d_1^u$ and $E\cdot d_1^u\cdot e_0$ are both nodes in $\mathcal{L}$, and each

process and each object is in the same state in $E \cdot e_0 \cdot d_1^u(I)$ as in $E \cdot d_1^u \cdot e_0(I)$. This implies that e_0 and d_1^u commute at E in $\mathcal{L}$, contrary to Lemma 6.5(a).

(ii) $w_0 = w_1$: The argument of Subcase 1(a)(i) also applies in this subcase by replacing d_1^u with $d_1^{\overline{u}}$ throughout. (But now the state of $\mathcal{O}$ in $E \cdot e_0 \cdot d_1^{\overline{u}}(I)$ and $E \cdot d_1^{\overline{u}} \cdot e_0(I)$ is $\langle w_0, w_0, u, \overline{u} \rangle$ if $j_0 = 0$ and $\langle w_0, w_0, \overline{u}, u \rangle$ if $j_0 = 1$.)

Subcase 1(b). *$\mathcal{O}$ is in a state of the form $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$*: First we claim that $v_0 = \perp$. If not, then by the specification of negation, $\mathcal{O}$ is upset in $E \cdot e_k(I)$ (since e_k is a NEGATE$(0, w_k)$ step to $\mathcal{O}$). This contradicts Corollary 7.8. Thus, $v_0 = \perp$. Given this, by the definition of the states of negation, it must be that $u_0 = \perp$ and $v_1, u_1 \neq \perp$. Recall that, in this case, $e_{\overline{k}}$ is a NEGATE$(1, w_{\overline{k}})$ step to $\mathcal{O}$. Since $v_1 \neq \perp$ and by Corollary 7.8 $\mathcal{O}$ is not upset in $E \cdot e_{\overline{k}}(I)$, it must be that $w_{\overline{k}} = v_1$. Then, it is easy to verify that $d_{\overline{k}}^{u_1}$ is the only step of $P_{\overline{k}}$ that is applicable to $E(I)$. Thus, $e_{\overline{k}} = d_{\overline{k}}^{u_1}$. We further distinguish two cases:

(i) $w_k \neq w_{\overline{k}}$: From the specification of negation, we can check that $d_k^{u_1}$ is the only step of P_k that is applicable to $E(I)$ and $E \cdot e_{\overline{k}}(I)$. Thus, $e_k = d_k^{u_1}$, and by Lemma 7.2(d), $E \cdot e_{\overline{k}} \cdot e_k$ is in $\mathcal{L}$. Also, $e_{\overline{k}}$ is the only step of $P_{\overline{k}}$ that is applicable to $E \cdot e_k$ (since $\mathcal{O}$ is in state $\langle w_k, v_1, u_1, u_1 \rangle$ in $E \cdot e_k(I)$). By Lemma 7.2(d), $E \cdot e_k \cdot e_{\overline{k}}$ is a node in $\mathcal{L}$.
By the specification of negation, it can be checked that $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$ (namely, in state $\langle w_k, v_1, u_1, u_1 \rangle$). Therefore, each process and each object is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. But then e_0 and e_1 commute at E in $\mathcal{L}$, contrary to Lemma 6.5(a).

(ii) $w_k = w_{\overline{k}}$: The argument in Subcase 1(b)(i) also applies in this case by replacing $d_k^{u_1}$ with $d_k^{\overline{u}_1}$ throughout. (But now $\mathcal{O}$ is in state $\langle w_k, v_1, \overline{u}_1, u_1 \rangle$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.)

Case 2. *For both $k \in \{0, 1\}$, $j_k = 1$*: In this case, it must be that $w_0 = w_1 = w$, for some $w \in \{0, 1\}$. (Otherwise, for each $k \in \{0, 1\}$, both $d_{\overline{k}}^0$ and $d_{\overline{k}}^1$ are applicable to $E \cdot e_k(I)$ and $\mathcal{O}$ is upset in $E \cdot e_k \cdot d_{\overline{k}}^0(I)$ and $E \cdot e_k \cdot d_{\overline{k}}^1(I)$, contrary to Lemma 7.7.) We distinguish two subcases.

Subcase 2(a). *$\mathcal{O}$ is fresh in $E(I)$*: The argument of Subcase 1(a)(i) applies here as well (but now the state of $\mathcal{O}$ in $E \cdot e_0 \cdot d_1^u(I)$ and $E \cdot d_1^u \cdot e_0(I)$ is $\langle \perp, w, \perp, u \rangle$).

Subcase 2(b). *$\mathcal{O}$ is in a state of the form $\langle v_0, v_1, u_0, u_1 \rangle$ in $E(I)$*: We further distinguish two cases:

(i) $v_1 = \perp$: By the definition of the states of negation, we have $u_1 = \perp$ and $v_0, u_0 \neq \perp$. Define u as

$$u = \begin{cases} u_0 & \text{if } w \neq v_0, \\ \overline{u}_0 & \text{if } w = v_0. \end{cases}$$

From the specification of negation, for each $k \in \{0, 1\}$, d_k^u is the only step of P_k that is applicable to $E(I)$. Hence, $e_k = d_k^u$, for each $k \in \{0, 1\}$. Also, e_k is the only step of P_k that is applicable to $E \cdot e_{\overline{k}}(I)$. Thus, by Lemma 7.2(d), both $E \cdot e_0 \cdot e_1$ and $E \cdot e_1 \cdot e_0$ are in $\mathcal{L}$. By the specification of negation, $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$ (namely, in state $\langle v_0, w, u_0, u \rangle$). Therefore, each process and each object is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. But then e_0 and e_1 commute at E in $\mathcal{L}$, contrary to Lemma 6.5(a).

(ii) $v_1 \neq \perp$: Since $v_1 \neq \perp$, by the definition of the states of negation, $u_1 \neq \perp$. The argument of Subcase 2(b)(i) applies in this case by taking u to be u_1 (instead of defining it as above).

Since all cases lead to a contradiction, the lemma follows. □

By Corollary 7.8 and Lemma 7.9, the state of $\mathcal{O}$ in $E(I)$ is not a legitimate state of type negation. Thus, $\mathcal{O}$ is not a negation object. This, together with Lemma 7.6 which shows that $\mathcal{O}$ is not a register, contradicts our hypothesis that $\mathbf{A}_c$ uses only negation objects and registers. Thus, we have the following.

THEOREM 7.10. *Type* negation *has consensus number one.*

8. Type booster has consensus number one. In this section, we show that type booster has consensus number one. Unfortunately, even the modified version of the bivalence argument of section 7 does not appear to be helpful in proving that consensus between two processes cannot be solved using only booster objects and registers. Instead, we use a different technique, originally used (in a somewhat different context) in [12]. Here is the outline of the proof.

Suppose, for contradiction, that type booster has consensus number at least two. By the universality of consensus [6], we can implement a negation object for two processes using only booster objects and registers. Thus, the equality negation problem for two processes is solvable using only booster objects and registers. Let $\mathbf{A}_{en}$ be an equality negation algorithm for two processes that uses a minimal number of booster objects (together with any number of registers).[6] By Theorem 3.1, $\mathbf{A}_{en}$ must use at least one booster object. We will show that $\mathbf{A}_{en}$ can be modified into an equality negation algorithm that uses one booster object less than $\mathbf{A}_{en}$, thus contradicting the definition of $\mathbf{A}_{en}$.

In this section we deal exclusively with *full* computation trees of $\mathbf{A}_{en}$. We first establish some properties of such trees that will be needed later on.

LEMMA 8.1. *Let I be any initial configuration of $\mathbf{A}_{en}$, and let $\mathcal{F}$ be the full computation tree of $\mathbf{A}_{en}$ from I. Let S_0, $S_0 \cdot S$, and S_1 be any nodes in $\mathcal{F}$ such that every process that takes a step in S and every object, except a* booster *object O, is in the same state in $S_0(I)$ as in $S_1(I)$. Suppose further that*

- *the state of O in $S_1(I)$ is ☹, or*
- *the state of O in $S_1(I)$ is of the form $\langle *, *, *, *, *, d\rangle$ for some $d \in \{0, 1\}$ and S contains no* REVEAL *step to O with return value $\overline{d}$ (i.e., S contains no step of the form $(*, \text{REVEAL}(*, *, *), O, \overline{d})$).*

Then, $S_1 \cdot S$ is a node in $\mathcal{F}$, and every process that takes a step in S and every object except O is in the same state in $S_0 \cdot S(I)$ as in $S_1 \cdot S(I)$.

Proof. From the specification of booster, it is easy to verify that

- the only state that is reachable from ☹ is itself, and a REVEAL operation can return either 0 or 1 when applied to a booster object in state ☹;
- every state reachable from a state of the form $\langle *, *, *, *, *, d\rangle$ is either of the same form or ☹, and a REVEAL operation can return d when applied to a booster object in a state of the form $\langle *, *, *, *, *, d\rangle$ or in state ☹.

Let S^i be the schedule consisting of the first i steps of S for $0 \le i \le |S|$. Using the above two properties, by a straightforward induction on i, we can show that for all i, $0 \le i \le |S|$, S^i is applicable to $S_1(I)$, and every process that takes a step in S and every object except O is in the same state in $S_0 \cdot S^i(I)$ as in $S_1 \cdot S^i(I)$. The lemma follows immediately from these facts, since $S = S^\ell$, where $\ell = |S|$. □

If the root of a full computation tree of $\mathbf{A}_{en}$ is univalent, the following lemma tells how the valence of the root relates to the possible decisions of the processes.

[6] This number must be finite by the termination property of equality negation and the fact that type booster is bounded nondeterministic.

LEMMA 8.2. *Let $k, u, v \in \{0,1\}$, I be any initial configuration of $\mathbf{A}_{en}$, $\mathcal{F}$ be the full computation tree of $\mathbf{A}_{en}$ from I, and S be any node of $\mathcal{F}$. If the root of $\mathcal{F}$ is v-valent and P_k has decided u in $S(I)$, then*

$$u = \begin{cases} v & \text{if } k = 0, \\ v & \text{if } k = 1 \text{ and } P_0, P_1 \text{ have different initial values in } I, \\ \overline{v} & \text{if } k = 1 \text{ and } P_0, P_1 \text{ have the same initial value in } I. \end{cases}$$

Proof. Let S_0 be a solo schedule of P_0 such that $S \cdot S_0$ is in $\mathcal{F}$ and P_0 has decided in $S \cdot S_0(I)$ (S_0 exists by Lemma 6.2). Since (by hypothesis) the root of $\mathcal{F}$ is v-valent, P_0 has decided v in $S \cdot S_0(I)$. By hypothesis, P_k has decided u in $S(I)$; since the decision of a process is irrevocable, P_k has decided u in $S \cdot S_0(I)$. If $k = 0$, clearly $u = v$, as wanted. If $k = 1$, then P_0 has decided v in $S \cdot S_0(I)$, and P_1 has decided u in $S \cdot S_0(I)$. By the negation property, $u = v$ if P_0 and P_1 have different initial values in I; and $u = \overline{v}$ if P_0 and P_1 have the same initial value in I—as wanted. □

LEMMA 8.3. *For some initial configuration I of algorithm $\mathbf{A}_{en}$, the full computation tree $\mathcal{F}$ of $\mathbf{A}_{en}$ from I has a bivalent root.*

Proof. Suppose, for contradiction, that the root of every full computation tree of $\mathbf{A}_{en}$ is univalent. For each $i, j \in \{0,1,2\}$, let $I_{i,j}$ be the initial configuration of $\mathbf{A}_{en}$ in which P_0 and P_1 have initial values i and j, respectively, and let $\mathcal{F}_{i,j}$ be the full computation tree of $\mathbf{A}_{en}$ from $I_{i,j}$.

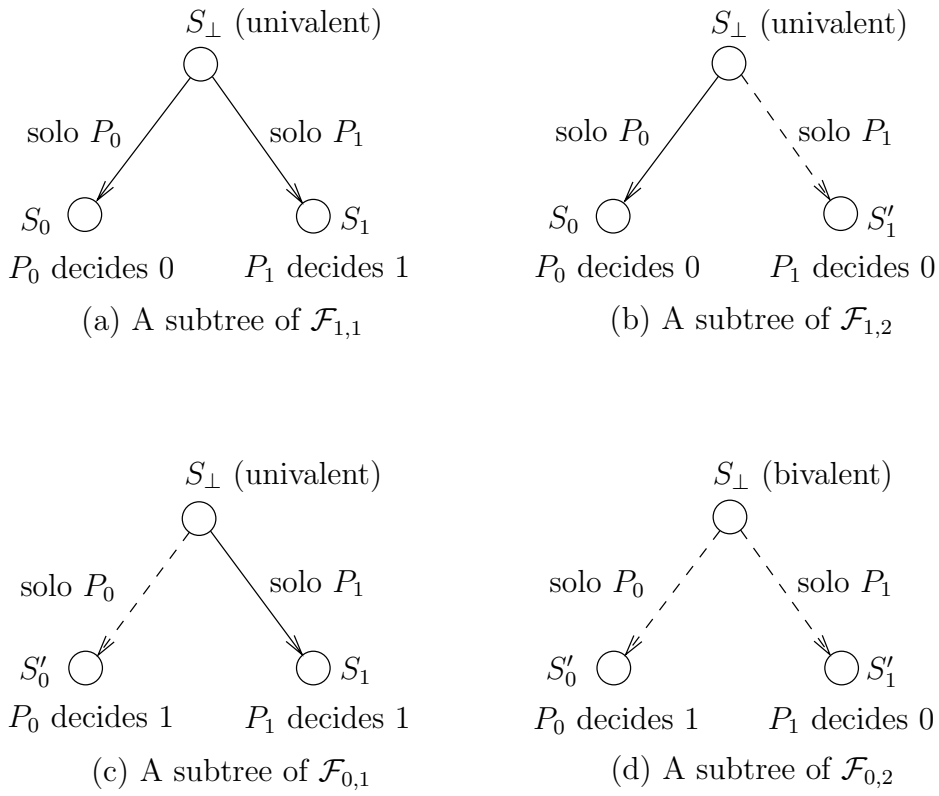
Without loss of generality, assume that the root of $\mathcal{F}_{1,1}$ is 0-valent. By Lemma 6.2 (applied to the root, $S_\perp$, of $\mathcal{F}_{1,1}$), for each $k \in \{0,1\}$ there is a solo schedule S_k of P_k such that $S_\perp \cdot S_k = S_k$ is a node in $\mathcal{F}_{1,1}$ and P_k has decided in $S_k(I_{1,1})$. By Lemma 8.2, P_k decides k in $S_k(I_{1,1})$; see Figure 8.1(a).

Since $I_{1,1}$ and $I_{1,2}$ differ only in the state of process P_1 and since S_0 is a solo schedule of P_0, by Lemma 6.1, S_0 is node in $\mathcal{F}_{1,2}$ and P_0 has the same state in $S_0(I_{1,2})$ as in $S_0(I_{1,1})$; see Figure 8.1(b). Thus, P_0 decides 0 in $S_0(I_{1,2})$. By the hypothesis that the root of $\mathcal{F}_{1,2}$ is univalent, it follows that the root of $\mathcal{F}_{1,2}$ is 0-valent. By Lemma 6.2 (applied to the root of $\mathcal{F}_{1,2}$), there is a solo schedule S_1' of P_1 such that S_1' is a node in $\mathcal{F}_{1,2}$ and P_1 has decided in $S_1'(I_{1,2})$. By Lemma 8.2, P_1 decides 0 in $S_1'(I_{1,2})$. Repeating the argument in this paragraph with $I_{1,2}$ replaced by $I_{0,1}$ and with the roles of 0 and 1 interchanged, we can prove that there exists a solo schedule S_0' of P_0 such that S_0' is a node in $\mathcal{F}_{0,1}$ and P_0 decides 1 in $S_0'(I_{0,1})$; see Figure 8.1(c).

Since $I_{0,1}$ and $I_{0,2}$ differ only in the state of P_1 and S_0' is a solo schedule of P_0, by Lemma 6.1, S_0' is a node in $\mathcal{F}_{0,2}$ and P_0 has the same state in $S_0'(I_{0,1})$ as in $S_0'(I_{0,2})$. Since P_0 decides 1 in $S_0'(I_{0,1})$, it also decides 1 in $S_0'(I_{0,2})$. By a similar argument, replacing $I_{0,1}$ and S_0' by $I_{1,2}$ and S_1', respectively, and interchanging the roles of P_0 and P_1, we can show that S_1' is a node in $\mathcal{F}_{0,2}$ and P_1 decides 0 in $S_1'(I_{0,2})$; see Figure 8.1(d). Thus, the root of $\mathcal{F}_{0,2}$ has descendants, namely, S_0' and S_1', in which processes have decided different values in the corresponding configurations. By Lemma 8.2, the root of $\mathcal{F}_{0,2}$ cannot be univalent. This contradicts our hypothesis that the root of every full computation tree of $\mathbf{A}_{en}$ is univalent. □

In the remainder of the section we focus exclusively on the initial configuration I and the full computation tree $\mathcal{F}$ of $\mathbf{A}_{en}$ from I whose existence is asserted by Lemma 8.3. Thus, from now on, we refer to the valence of a node without explicitly specifying the computation tree, since that will always be $\mathcal{F}$.

Since the root of $\mathcal{F}$ is bivalent, by Lemma 6.7, $\mathcal{F}$ has a critical node E. By the definition of critical node, there are steps e_0 and e_1 of processes P_0 and P_1, respectively,



(a) A subtree of $\mathcal{F}_{1,1}$ (b) A subtree of $\mathcal{F}_{1,2}$

(c) A subtree of $\mathcal{F}_{0,1}$ (d) A subtree of $\mathcal{F}_{0,2}$

FIG. 8.1. *Subtrees of four full computation trees of* $\mathbf{A}_{en}$.

such that $E \cdot e_0$ and $E \cdot e_1$ are univalent nodes of opposite valence. Without loss of generality, assume that $E \cdot e_0$ is 0-valent and $E \cdot e_1$ is 1-valent (if not, we simply rename e_0 and e_1). By Lemma 6.6, e_0 and e_1 access the same object, say, $\mathcal{O}$. We can show that $\mathcal{O}$ is not a register by using an argument similar to the one used in the proof of Lemma 7.6 (except simpler, since we are now working with the full computation tree and need not worry about whether certain schedules applicable to I have been pruned). Thus, $\mathcal{O}$ is a booster object. In what follows, we say that $\mathcal{O}$ is *fresh* in a configuration C of $\mathbf{A}_{en}$ if its state in C is ☺ and *upset* in C if its state in C is ☹.

We now outline how the proof proceeds: To show that the assumed equality negation algorithm $\mathbf{A}_{en}$ that uses only booster objects and registers does not exist, we derive a contradiction in two stages. In the first stage, we prove certain facts about the configuration $E(I)$ and the booster object $\mathcal{O}$. Specifically, we show that $\mathcal{O}$ is fresh in $E(I)$ (Corollary 8.5 and Lemma 8.7), that each process has an ENROLL operation (with a different parameter) to $\mathcal{O}$ pending in $E(I)$, and that neither process can decide before one of them has applied a REVEAL operation to $\mathcal{O}$ (Lemma 8.11). Furthermore, if neither process has yet applied a REVEAL operation to $\mathcal{O}$ but each is about to, then the parameters of the two pending REVEAL operations must be such as to ensure that $\mathcal{O}$ does not become upset (Lemma 8.13). These facts are then exploited in the second stage of the proof to show how to solve equality negation using only registers and the booster objects of $\mathbf{A}_{en}$ *except* $\mathcal{O}$. We have thus reached a contradiction, as $\mathbf{A}_{en}$ was assumed to use the smallest possible number of booster objects. The details of these

two stages are carried out in the next two subsections.

8.1. Stage 1 of the proof. We now show that $\mathcal{O}$ is fresh in $E(I)$. We do this by proving that, in $E(I)$, $\mathcal{O}$ is neither upset (Corollary 8.5) nor in a state of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$ (Lemma 8.7).

LEMMA 8.4. *Object $\mathcal{O}$ is not upset in $E\cdot e_0(I)$ or in $E\cdot e_1(I)$.*

Proof. Suppose, for contradiction, that $\mathcal{O}$ is upset in $E\cdot e_k(I)$ for some $k \in \{0,1\}$. By Lemma 6.2 (applied to node $E\cdot e_{\overline{k}}$ in $\mathcal{F}$), there is a solo schedule $S_{\overline{k}}$ of $P_{\overline{k}}$ such that $E\cdot e_{\overline{k}}\cdot S_{\overline{k}}$ is a node in $\mathcal{F}$ and $P_{\overline{k}}$ has decided in $E\cdot e_{\overline{k}}\cdot S_{\overline{k}}(I)$. Comparing configurations $E(I)$ and $E\cdot e_k(I)$, we note that $P_{\overline{k}}$ and each object other than $\mathcal{O}$ is in the same state in both. Since, by assumption, $\mathcal{O}$ is upset in $E\cdot e_k(I)$ and $e_{\overline{k}}\cdot S_{\overline{k}}$ is a solo schedule of $P_{\overline{k}}$, by Lemma 8.1, $E\cdot e_k\cdot e_{\overline{k}}\cdot S_{\overline{k}}$ is a node in $\mathcal{F}$, and $P_{\overline{k}}$ is in the same state in $E\cdot e_{\overline{k}}\cdot S_{\overline{k}}(I)$ as in $E\cdot e_k\cdot e_{\overline{k}}\cdot S_{\overline{k}}(I)$. Thus, $P_{\overline{k}}$ decides the same value in both $E\cdot e_{\overline{k}}\cdot S_{\overline{k}}(I)$ and $E\cdot e_k\cdot e_{\overline{k}}\cdot S_{\overline{k}}(I)$. By Lemma 6.3, $E\cdot e_{\overline{k}}\cdot S_{\overline{k}}$ and $E\cdot e_k\cdot e_{\overline{k}}\cdot S_{\overline{k}}$ have the same valence. Since $E\cdot e_{\overline{k}}$ and $E\cdot e_k$ are univalent and have descendants of the same valence, they also have the same valence. This contradicts the fact that $E\cdot e_0$ is 0-valent while $E\cdot e_1$ is 1-valent. □

Once a booster object is upset, it remains upset forever. Thus, Lemma 8.4 implies the following.

COROLLARY 8.5. *Object $\mathcal{O}$ is not upset in $E(I)$.*

LEMMA 8.6. *$E\cdot e_0\cdot e_1$ and $E\cdot e_1\cdot e_0$ are nodes in $\mathcal{F}$.*

Proof. By definition of $\mathcal{F}$, it suffices to show that for each $k \in \{0,1\}$ e_k is applicable to $E\cdot e_{\overline{k}}(I)$. If e_k is an ENROLL step, the lemma follows immediately by the specification of booster.

Suppose that e_k is a REVEAL step. If $\mathcal{O}$ is fresh in $E(I)$, then $\mathcal{O}$ is upset in $E\cdot e_k(I)$, contrary to Lemma 8.4. Thus, $\mathcal{O}$ is not fresh in $E(I)$. By Corollary 8.5, $\mathcal{O}$ is not upset in $E(I)$. Thus, $\mathcal{O}$ is in a state of the form $\langle *, *, *, *, *, d\rangle$ in $E(I)$ for some $d \in \{0,1\}$. By the specification of booster, and since $\mathcal{O}$ is not upset in $E\cdot e_k(I)$, the response returned by the REVEAL operation in e_k is d. Since $\mathcal{O}$ is in a state of the form $\langle *, *, *, *, *, d\rangle$ in $E(I)$ and is not upset in $E\cdot e_{\overline{k}}(I)$, it must be in a state of the form $\langle *, *, *, *, *, d\rangle$ in $E\cdot e_{\overline{k}}(I)$ as well (because the last component of the state is never changed, once it is set). By the specification of booster, a REVEAL operation to $\mathcal{O}$ applied in such a state can return the response d. Thus, e_k is indeed applicable to $E\cdot e_{\overline{k}}(I)$. □

LEMMA 8.7. *The state of $\mathcal{O}$ in $E(I)$ is not of the form $\langle V, v_0, v_1, u_0, u_1, d\rangle$.*

Proof. Suppose, for contradiction, that the state of $\mathcal{O}$ in $E(I)$ is $\langle V, v_0, v_1, u_0, u_1, d\rangle$. By Lemma 8.6, both $E\cdot e_0\cdot e_1$ and $E\cdot e_1\cdot e_0$ are nodes in $\mathcal{F}$. We shall show that

(2) the state of $\mathcal{O}$ is the same in $E\cdot e_0\cdot e_1(I)$ as in $E\cdot e_1\cdot e_0(I)$.

Clearly, the state of each process and object except $\mathcal{O}$ is the same in $E\cdot e_1\cdot e_0(I)$ as in $E\cdot e_0\cdot e_1(I)$. By (2), we have thus shown that e_0 and e_1 commute at E in $\mathcal{F}$. This contradicts Lemma 6.5(a), as wanted.

It remains to show (2). Let op_0 and op_1 be the (ENROLL or REVEAL) operations to $\mathcal{O}$ of processes P_0 and P_1 that are pending in $E(I)$, respectively. There are three cases, depending on the types of op_0 and op_1.

Case 1. *Both operations pending in $E(I)$ are* ENROLL: Let $op_0 = \text{ENROLL}(i_0)$ and $op_1 = \text{ENROLL}(i_1)$ for some $i_0, i_1 \in \{0,1\}$. For each $k \in \{0,1\}$, let r_k be the response of $\mathcal{O}$ to the $\text{ENROLL}(i_k)$ operation in step e_k. There are three subcases.

Subcase 1(a). $i_0 = i_1 = 0$: From the specification of booster, $\mathcal{O}$ is upset in both $E\cdot e_0\cdot e_1(I)$ and $E\cdot e_1\cdot e_0(I)$.

Subcase 1(b). $i_0 = i_1 = 1$: From the specification of booster, $\mathcal{O}$ is in state $\langle V \cup \{r_0, r_1\}, v_0, v_1, u_0, u_1, d\rangle$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Subcase 1(c). *For some* $k \in \{0,1\}$, $i_k = 0$ *and* $i_{\overline{k}} = 1$: Without loss of generality we assume that $i_0 = 0$ and $i_1 = 1$. Then $v_0 = \perp$; if not, $\mathcal{O}$ would be upset in $E \cdot e_0(I)$, which contradicts Lemma 8.4. By the definition of the set of states of booster, $u_0 = \perp$. From this, it follows that $\mathcal{O}$ is in state $\langle V \cup \{r_1\}, r_0, v_1, \perp, u_1, d\rangle$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Case 2. *One of the two operations pending in* $E(I)$ *is* ENROLL *and the other is* REVEAL: Without loss of generality, we may assume that $op_0 = \text{ENROLL}(i_0)$ and $op_1 = \text{REVEAL}(i_1, v, u)$ for some $i_0, i_1, u \in \{0,1\}$ and $v \in \{0,1,2\}$. Let r be the response of the $\text{ENROLL}(i_0)$ operation that is returned in step e_0. There are three subcases.

Subcase 2(a). $i_0 = 0$: Then $v_0 = \perp$ (otherwise, $\mathcal{O}$ would be upset in $E \cdot e_0(I)$, contrary to Lemma 8.4), and hence $u_0 = \perp$ (by the definition of the set of states of booster). Also, $i_1 = 1$ (otherwise, $\mathcal{O}$ would be upset in $E \cdot e_1(I)$). Thus, $\mathcal{O}$ is in state $\langle V, \perp, v_1, \perp, u_1, d\rangle$ in $E(I)$ and the operations of P_0 and P_1 pending in $E(I)$ are $\text{ENROLL}(0)$ and $\text{REVEAL}(1, v, u)$, respectively. From the specification of booster, the state of $\mathcal{O}$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$ is

$$\begin{cases} \langle V, r, v, \perp, u, d\rangle & \text{if } v \in V \textbf{ and } (u_1 = \perp \textbf{ or } (v = v_1 \textbf{ and } u = u_1)), \\ \text{☹} & \text{otherwise.} \end{cases}$$

Subcase 2(b). $i_0 = 1$ *and* $i_1 = 0$: Since $\mathcal{O}$ is not upset in $E \cdot e_1(I)$, the Boolean expression

$$v = v_0 \textbf{ and } u_0 = \perp \textbf{ and } (u_1 = \perp \textbf{ or } (v = v_1 \Leftrightarrow u \neq u_1))$$

is true, and $\mathcal{O}$ is in state $\langle V, v_0, v_1, u, u_1, d\rangle$ in $E \cdot e_1(I)$. Thus, $\mathcal{O}$ is in state $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$ in $E \cdot e_1 \cdot e_0(I)$.

The state of $\mathcal{O}$ in $E \cdot e_0(I)$ is $\langle V \cup \{r\}, v_0, v_1, \perp, u_1, d\rangle$ (recall that $u_0 = \perp$). Since the preceding Boolean expression is true, by the specification of booster, the application of $\text{REVEAL}(0, v, u)$ to $\mathcal{O}$ in state $\langle V \cup \{r\}, v_0, v_1, \perp, u_1, d\rangle$ does not cause $\mathcal{O}$ to become upset; in fact, it causes it to enter state $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$. Thus, e_1 is applicable to $E \cdot e_0(I)$, and the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ is $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$, which is the same as in $E \cdot e_1 \cdot e_0(I)$.

Subcase 2(c). $i_0 = 1$ *and* $i_1 = 1$: Similar to Subcase 2(b), except that in this case the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$ is $\langle V \cup \{r\}, v_0, v, u_0, u, d\rangle$ instead of $\langle V \cup \{r\}, v_0, v_1, u, u_1, d\rangle$.

Case 3. *Both operations pending in* $E(I)$ *are* REVEAL: Let $op_0 = \text{REVEAL}(i_0, v, u)$ and $op_1 = \text{REVEAL}(i_1, v', u')$ for some $i_0, i_1, u, u' \in \{0,1\}$ and $v, v' \in \{0,1,2\}$. There are three subcases.

Subcase 3(a). $i_0 = i_1 = 0$: Then $\mathcal{O}$ is upset in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Subcase 3(b). $i_0 \neq i_1$: Without loss of generality, we may assume that $i_0 = 0$ and $i_1 = 1$. It must be the case that $v = v_0$ and $u_0 = \perp$ (otherwise $\mathcal{O}$ is upset in $E \cdot e_0(I)$, contrary to Lemma 8.4). Also $v' \in V$ and either $u_1 = \perp$, or $v' = v_1$ and $u' = u_1$ (otherwise $\mathcal{O}$ is upset in $E \cdot e_1(I)$). Thus, the state of $\mathcal{O}$ in $E(I)$ is either $\langle V, v, \perp, \perp, \perp, d\rangle$ or $\langle V, v, v', \perp, u', d\rangle$. In either case, it can be verified from the specification of booster that the state of $\mathcal{O}$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$ is

$$\begin{cases} \langle V, v, v', u, u', d\rangle & \text{if } v = v' \Leftrightarrow u \neq u', \\ \text{☹} & \text{otherwise.} \end{cases}$$

Subcase 3(c). $i_0 = i_1 = 1$: Since $\mathcal{O}$ is not upset in either one of $E \cdot e_0(I)$ or $E \cdot e_1(I)$, it must be that $v, v' \in V$ and $\mathcal{O}$ is in state $\langle V, v_0, v, u_0, u, d\rangle$ in $E \cdot e_0(I)$ and in $\langle V, v_0, v', u_0, u', d\rangle$ in $E \cdot e_1(I)$. From the specification of booster, it can be checked that the state of $\mathcal{O}$ in $E \cdot e_0 \cdot e_1(I)$ is

$$\begin{cases} \langle V, v_0, v', u_0, u', d\rangle & \text{if } v' = v \text{ and } u' = u, \\ \text{☹} & \text{otherwise.} \end{cases}$$

Similarly, the state of $\mathcal{O}$ in $E \cdot e_1 \cdot e_0(I)$ is

$$\begin{cases} \langle V, v_0, v, u_0, u, d\rangle & \text{if } v = v' \text{ and } u = u', \\ \text{☹} & \text{otherwise.} \end{cases}$$

Thus, $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. This completes the proof of (2), and thereby of the lemma. □

By Corollary 8.5 and Lemma 8.7, $\mathcal{O}$ must be fresh in $E(I)$. Furthermore, each P_k must have an ENROLL operation to $\mathcal{O}$ pending in $E(I)$. (Otherwise, by the specification of booster, $\mathcal{O}$ would be upset in $E \cdot e_k(I)$, contrary to Lemma 8.4.) For each $k \in \{0, 1\}$, let $j_k \in \{0, 1\}$ be the parameter of the ENROLL operation of P_k pending in $E(I)$.

The next lemma states that the parameters of the two ENROLL operations to $\mathcal{O}$ pending in $E(I)$ are different.

LEMMA 8.8. $j_0 \neq j_1$.

Proof. Suppose, for contradiction, that $j_0 = j_1$. As in the proof of Lemma 8.7, we shall show that $\mathcal{O}$ is in the same state in $E \cdot e_0 \cdot e_1(I)$ as in $E \cdot e_1 \cdot e_0(I)$. This fact, together with Lemma 8.6, implies that e_0 and e_1 commute at E in $\mathcal{F}$, thus contradicting Lemma 6.5(a).

Recall that $j_0, j_1 \in \{0, 1\}$. There are two cases.

Case 1. $j_0 = j_1 = 0$: Then $\mathcal{O}$ is upset in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$.

Case 2. $j_0 = j_1 = 1$: Let r_0 and r_1 be the responses of $\mathcal{O}$ returned to the ENROLL operations in e_0 and e_1, respectively. Then $\mathcal{O}$ is in state $\langle \{r_0, r_1\}, \perp, \perp, \perp, \perp, 1\rangle$ in both $E \cdot e_0 \cdot e_1(I)$ and $E \cdot e_1 \cdot e_0(I)$. □

We next show that after applying its ENROLL operation that is pending in $E(I)$ no process can decide unless some process has applied at least one REVEAL operation to $\mathcal{O}$ (Lemma 8.11). In the next two lemmas, we establish some properties of schedules applicable to $E(I)$ used to prove this fact.

LEMMA 8.9. *For any $k \in \{0, 1\}$ and any node $E \cdot S$ in $\mathcal{F}$, if P_k is the process that takes the first step in S, then $E \cdot S$ is k-valent.*

Proof. Later we prove that

(3) for each $k \in \{0, 1\}$, the nodes in *children*$(P_k, E, \mathcal{F})$ have the same valence.

Let e be the first step in S, which is taken by process P_k for some $k \in \{0, 1\}$. Thus, $E \cdot e$ is a node in *children*$(P_k, E, \mathcal{F})$. By (3), all nodes in *children*$(P_k, E, \mathcal{F})$ are univalent and have the same valence. One of these children, namely, $E \cdot e_k$, is k-valent. Therefore, $E \cdot e$ is also k-valent. Since $E \cdot S$ is a descendant of $E \cdot e$, it follows that $E \cdot S$ is k-valent.

It remains to prove (3). We prove it for $k = 0$. (The proof for $k = 1$ can be obtained by interchanging the roles of 0 and 1 in the argument below.) For each $v \in \{0, 1, 2\}$, let $e_0^v = (P_0, \text{ENROLL}(j_0), \mathcal{O}, v)$. Since $\mathcal{O}$ is fresh in $E(I)$ and an ENROLL operation applied to a fresh booster object can return any value in $\{0, 1, 2\}$, it follows

that $children(P_0, E, \mathcal{F}) = \{ E \cdot e_0^v : 0 \leq v \leq 2 \}$. Since E is a critical node, all nodes in $children(P_0, E, \mathcal{F})$ are univalent. To prove (3), it suffices to show that for any $v, v' \in \{0, 1, 2\}$, $E \cdot e_0^v$ and $E \cdot e_0^{v'}$ have the same valence. From the specification of booster and the fact that $\mathcal{O}$ is fresh in $E(I)$ we have the following:

(a) For any $v \in \{0, 1, 2\}$, the state of $\mathcal{O}$ in $E \cdot e_0^v(I)$ is $\langle *, *, *, *, *, j_0 \rangle$.
(b) For any $v \in \{0, 1, 2\}$, if S is a schedule applicable to $E \cdot e_0^v(I)$, then the state of $\mathcal{O}$ in $E \cdot e_0^v \cdot S(I)$ is either of the form $\langle *, *, *, *, *, j_0 \rangle$ or ☹.
(c) A REVEAL operation can return j_0 when applied to a booster object in a state of the form $\langle *, *, *, *, *, j_0 \rangle$ or in state ☹.

Let v be any element of $\{0, 1, 2\}$. By Lemma 6.2 (applied to node $E \cdot e_0^v$ in $\mathcal{F}$), there is a solo schedule S of P_1 such that $E \cdot e_0^v \cdot S$ is a node in $\mathcal{F}$ and P_1 has decided in $E \cdot e_0^v \cdot S(I)$. Since $\mathcal{F}$ is a full computation tree, by (a)–(c) we may assume, without loss of generality, that S contains no step of the form $(*, \text{REVEAL}(*, *, *), \mathcal{O}, \overline{j}_0)$.

For any $v' \in \{0, 1, 2\}$, the state of P_1 and each object except $\mathcal{O}$ is the same in $E \cdot e_0^v(I)$ as in $E \cdot e_0^{v'}(I)$. Since the state of $\mathcal{O}$ in $E \cdot e_0^{v'}(I)$ is of the form $\langle *, *, *, *, *, j_0 \rangle$ and since S does not contain a REVEAL step to $\mathcal{O}$ with return value $\overline{j}_0$, the hypothesis of Lemma 8.1 applies for $S_0 = E \cdot e_0^v$ and $S_1 = E \cdot e_0^{v'}$. Thus, $E \cdot e_0^{v'} \cdot S$ is a node in $\mathcal{F}$, and P_1 is in the same state in $E \cdot e_0^v \cdot S(I)$ as in $E \cdot e_0^{v'} \cdot S(I)$. This implies that P_1 decides the same value in these two configurations. By Lemma 6.3, nodes $E \cdot e_0^v \cdot S$ and $E \cdot e_0^{v'} \cdot S$ have the same valence. Hence, $E \cdot e_0^v$ and $E \cdot e_0^{v'}$ (which are univalent since E is critical) must also have the same valence. This completes the proof of (3) and thereby of the lemma. □

LEMMA 8.10. *Let $E \cdot S$ be any node in $\mathcal{F}$ such that S contains no* REVEAL *step to $\mathcal{O}$, and d_k be the first step of P_k in S for any $k \in \{0, 1\}$. Let $\hat{S}$ be the schedule obtained from S by moving d_k to the beginning (i.e., if $S = S' \cdot d_k \cdot S''$ for some S' and S'', then $\hat{S} = d_k \cdot S' \cdot S''$). Then, $E \cdot \hat{S}$ is a node in $\mathcal{F}$, and the state of each process and each object other than $\mathcal{O}$ is the same in $E \cdot \hat{S}(I)$ as in $E \cdot S(I)$.*

Proof. Let S' and S'' be such that $S = S' \cdot d_k \cdot S''$. Recall that process P_k has an ENROLL(j_k) to $\mathcal{O}$ pending in $E(I)$. Since S' is a solo schedule of $P_{\overline{k}}$ (because d_k is the first step of P_k in S), the operation of P_k pending in $E \cdot S'(I)$ is still ENROLL(j_k) to $\mathcal{O}$. Thus, d_k is an ENROLL step to $\mathcal{O}$. By the specification of booster, d_k is applicable to $E(I)$. Clearly, as d_k is a step of P_k that accesses (only) $\mathcal{O}$, the state of $P_{\overline{k}}$ and each object other than $\mathcal{O}$ is the same in $E(I)$ as in $E \cdot d_k(I)$. Also, the state of $\mathcal{O}$ in $E \cdot d_k(I)$ is of the form $\langle *, *, *, *, *, j_k \rangle$ (recall that d_k is an ENROLL(j_k) step to $\mathcal{O}$). Since $E \cdot S'$ is a node in $\mathcal{F}$ and S' is a solo schedule of $P_{\overline{k}}$ that does not contain a REVEAL step to $\mathcal{O}$, by Lemma 8.1 (applied for $S_0 = E$, $S_1 = E \cdot d_k$, and $S = S'$), $E \cdot d_k \cdot S'$ is a node in $\mathcal{F}$, and the state of $P_{\overline{k}}$ and each object other than $\mathcal{O}$ is the same in $E \cdot d_k \cdot S'(I)$ as in $E \cdot S'(I)$. It follows that each process and each object other than $\mathcal{O}$ is in the same state in $E \cdot d_k \cdot S'(I)$ as in $E \cdot S' \cdot d_k(I)$. By hypothesis, $E \cdot S' \cdot d_k \cdot S''$ is a node in $\mathcal{F}$ and S'' does not contain a REVEAL step to $\mathcal{O}$. Again by Lemma 8.1 (applied for $S_0 = E \cdot S' \cdot d_k$, $S_1 = E \cdot d_k \cdot S'$, and $S = S''$), $E \cdot d_k \cdot S' \cdot S''$ is a node in $\mathcal{F}$, and the state of each process and each object other than $\mathcal{O}$ is the same in $E \cdot S' \cdot d_k \cdot S''(I)$ as in $E \cdot d_k \cdot S' \cdot S''(I)$. □

LEMMA 8.11. *Let $E \cdot S$ be any node in $\mathcal{F}$ such that some process has decided in $E \cdot S(I)$. Then S has a* REVEAL *step to $\mathcal{O}$.*

Proof. Suppose, for contradiction, that some process has decided in $E \cdot S(I)$, but S contains no REVEAL step to $\mathcal{O}$. Without loss of generality, we may assume that both processes take steps in S. (If S contains only steps of process P_k for some $k \in \{0, 1\}$, we extend S by a step of $P_{\overline{k}}$, which cannot be a REVEAL step to $\mathcal{O}$ because

the operation of $P_{\bar{k}}$ pending in $E(I)$ is an ENROLL to $\mathcal{O}$, and consider the extended schedule instead.)

Let P_ℓ, $\ell \in \{0,1\}$, be the process that does *not* take the first step in S, and $\hat{S}$ be the schedule obtained from S by moving the first step of P_ℓ to the beginning. By Lemma 8.10, $E \cdot \hat{S}$ is a node in $\mathcal{F}$, and the state of each process and each object except $\mathcal{O}$ is the same in $E \cdot \hat{S}(I)$ as in $E \cdot S(I)$. In particular, the process that has decided in $E \cdot S(I)$ decides the same value in $E \cdot \hat{S}(I)$. By Lemma 6.3, $E \cdot S$ and $E \cdot \hat{S}$ have the same valence. This contradicts Lemma 8.9, since P_ℓ takes the first step in $\hat{S}$ while $P_{\bar{\ell}}$ takes the first step in S. □

To summarize, $\mathcal{O}$ is fresh in $E(I)$, each P_k has an ENROLL(j_k) operation to $\mathcal{O}$ pending in $E(I)$, and $j_0 \neq j_1$. By Lemma 8.11, no process can decide unless a REVEAL operation is applied to $\mathcal{O}$. Thus, by the termination requirement of equality negation, there is a schedule S applicable to $E(I)$ so that no process has taken a REVEAL step to $\mathcal{O}$ in S, and each process has a REVEAL operation to $\mathcal{O}$ pending in $E \cdot S(I)$. S contains ENROLL steps of both processes because each process P_k, $k \in \{0,1\}$, has an ENROLL(j_k) operation to $\mathcal{O}$ pending in $E(I)$ but has a REVEAL operation to $\mathcal{O}$ pending in $E \cdot S(I)$. By Lemma 8.8, S contains both ENROLL(0) and ENROLL(1) steps to $\mathcal{O}$.

The next lemma proves that, in any execution starting from configuration $E(I)$, if each of the two processes is about to apply its first REVEAL operation to $\mathcal{O}$, then the application of these two operations does not cause $\mathcal{O}$ to become upset.

LEMMA 8.12. *Consider any node $E \cdot S$ in $\mathcal{F}$ such that S contains no* REVEAL *step to $\mathcal{O}$ and both processes have a* REVEAL *operation to $\mathcal{O}$ pending in $E \cdot S(I)$. For each $k \in \{0,1\}$, let d_k be any step of P_k and $c_{\bar{k}}$ be any step of $P_{\bar{k}}$ such that $E \cdot S \cdot d_k \cdot c_{\bar{k}}$ is a node in $\mathcal{F}$. (Note that d_k and $c_{\bar{k}}$ are* REVEAL *steps that access $\mathcal{O}$.) Then $\mathcal{O}$ is not upset in $E \cdot S \cdot d_k \cdot c_{\bar{k}}(I)$.*

Proof. Suppose, to the contrary, that $\mathcal{O}$ is upset in $E \cdot S \cdot d_k \cdot c_{\bar{k}}(I)$ for some $k \in \{0,1\}$. Let P_ℓ, $\ell \in \{0,1\}$, be the process that does *not* take the first step in S, and $\hat{S}$ be the schedule obtained from S by moving the first step of P_ℓ to the beginning. By Lemma 8.10, $E \cdot \hat{S}$ is a node in $\mathcal{F}$, and each process and each object except $\mathcal{O}$ is in the same state in $E \cdot S(I)$ as in $E \cdot \hat{S}(I)$. Let $\hat{d}_k$ be any step of P_k such that $E \cdot \hat{S} \cdot \hat{d}_k$ is a node in $\mathcal{F}$. We shall prove that

$$E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\bar{k}} \text{ is a node in } \mathcal{F}, \text{ and } \mathcal{O} \text{ is upset in } E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\bar{k}}(I). \tag{4}$$

Given (4), process $P_{\bar{k}}$ and each object (including $\mathcal{O}$) is in the same state in $E \cdot S \cdot d_k \cdot c_{\bar{k}}(I)$ as in $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\bar{k}}(I)$. By Lemma 6.4, $E \cdot S \cdot d_k \cdot c_{\bar{k}}$ and $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\bar{k}}$ have the same valence. This contradicts Lemma 8.9, since P_ℓ takes the first step in $\hat{S}$, while $P_{\bar{\ell}}$ takes the first step in S.

It remains to show (4). There are three cases.

Case 1. *$\mathcal{O}$ is upset in $E \cdot S(I)$*: Since S contains only ENROLL operations to $\mathcal{O}$, it must contain two ENROLL(0) operations to $\mathcal{O}$ (otherwise $\mathcal{O}$ would not be upset in $E \cdot S(I)$). Then so does $\hat{S}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S}(I)$. It is immediate from the specification of booster that $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot \hat{d}_k(I)$ and $c_{\bar{k}}$ is applicable to $E \cdot \hat{S} \cdot \hat{d}_k(I)$. Therefore, $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\bar{k}}$ is a node in $\mathcal{F}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\bar{k}}(I)$ (because once a booster object becomes upset, it remains upset forever).

Case 2. *$\mathcal{O}$ is not upset in $E \cdot S(I)$ but it is upset in $E \cdot S \cdot d_k(I)$*: S contains exactly one ENROLL(0) and at least one ENROLL(1) step. (We have already argued, in the paragraph preceding Lemma 8.12, that S contains both ENROLL(0) and ENROLL(1)

steps. It cannot contain two ENROLL(0) steps because, otherwise, $\mathcal{O}$ would be upset in $E \cdot S(I)$, contrary to the hypothesis of this case.) By assumption, S does not contain any REVEAL steps. Since $\mathcal{O}$ is fresh in $E(I)$ and it is not upset in $E \cdot S(I)$, the state of $\mathcal{O}$ in $E \cdot S(I)$ is $\langle V, v, \perp, \perp, \perp, j_{\overline{\ell}} \rangle$, where V is the set of return values of the ENROLL(1) steps in S, and v is the return value of the ENROLL(0) step in S. (To see why the last component of the state is $j_{\overline{\ell}}$, recall that $P_{\overline{\ell}}$ has an ENROLL($j_{\overline{\ell}}$) operation to $\mathcal{O}$ pending in $E(I)$. Since $P_{\overline{\ell}}$ is the process that takes the first step in S, that step involves an ENROLL($j_{\overline{\ell}}$) operation to $\mathcal{O}$. Since $\mathcal{O}$ is fresh in $E(I)$, when that ENROLL($j_{\overline{\ell}}$) operation is applied to $\mathcal{O}$, the state of $\mathcal{O}$ is set to a state of the form $\langle *, *, \perp, \perp, \perp, j_{\overline{\ell}} \rangle$. The last component of the state is not changed, once it is set.) Similarly, the state of $\mathcal{O}$ in $E \cdot \hat{S}(I)$ is $\langle V, v, \perp, \perp, \perp, j_\ell \rangle$—i.e., the same as in $E \cdot S(I)$, except for the last component.

Since $\mathcal{O}$ is upset in $E \cdot S \cdot d_k(I)$ it follows, from the specification of booster, that d_k is a REVEAL(i_k, v_k, u_k) step to $\mathcal{O}$ so that one of the following holds:

(a) $i_k = 0$ and $v_k \neq v$ (i.e., v_k is not the return value of the ENROLL(0) step in S);
(b) $i_k = 1$ and $v_k \notin V$ (i.e., v_k is not the return value of some ENROLL(1) step in S).

In either case, it is immediate from the specification of booster that the application of REVEAL(i_k, v_k, u_k) to $\mathcal{O}$ in state $\langle V, v, \perp, \perp, \perp, j_\ell \rangle$ will cause $\mathcal{O}$ to become upset (and can return any value in $\{0, 1\}$). Since the state of process P_k is the same in $E \cdot S(I)$ as is $E \cdot \hat{S}(I)$, $\hat{d}_k$ is also a REVEAL(i_k, v_k, u_k) step to $\mathcal{O}$. Thus, $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot \hat{d}_k(I)$. Hence, $c_{\overline{k}}$ is applicable to $E \cdot \hat{S} \cdot \hat{d}_k(I)$. This implies that $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\overline{k}}$ is a node in $\mathcal{F}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\overline{k}}(I)$.

Case 3. *$\mathcal{O}$ is not upset in $E \cdot S \cdot d_k(I)$ but it is upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$*: Let the REVEAL operation applied to $\mathcal{O}$ in d_k be REVEAL(i_k, v_k, u_k). Since $\mathcal{O}$ is not upset in $E \cdot S \cdot d_k(I)$, and S does not contain any REVEAL steps, we conclude the following about the state of $\mathcal{O}$ in $E \cdot S \cdot d_k(I)$:

- If $i_k = 0$, then that state is $\langle V, v, \perp, u_k, \perp, j_{\overline{\ell}} \rangle$, where V is the set of return values of the ENROLL(1) steps in S, v is the return value of the ENROLL(0) step in S, and $v_k = v$.
- If $i_k = 1$, then that state is $\langle V, v, v_k, \perp, u_k, j_{\overline{\ell}} \rangle$, where V is the set of return values of the ENROLL(1) steps in S, v is the return value of the ENROLL(0) step in S, and $v_k \in V$.

As argued in the preceding case, $\hat{d}_k$ is also a REVEAL(i_k, v_k, u_k) step to $\mathcal{O}$. Thus, the state of $\mathcal{O}$ in $E \cdot \hat{S} \cdot \hat{d}_k(I)$ is as above, except that the last component is j_ℓ instead of $j_{\overline{\ell}}$. Since $\mathcal{O}$ is upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$, from the specification of booster it follows that $c_{\overline{k}}$ is a REVEAL$(i_{\overline{k}}, v_{\overline{k}}, u_{\overline{k}})$ step to $\mathcal{O}$ so that one of the following holds:

(a) $i_{\overline{k}} = 0$ and $v_{\overline{k}} \neq v$ (i.e., $v_{\overline{k}}$ is not the return value of the ENROLL(0) step in S).
(b) $i_{\overline{k}} = 1$ and $v_{\overline{k}} \notin V$ (i.e., $v_{\overline{k}}$ is not the return value of some ENROLL(1) step in S).
(c) $i_{\overline{k}} = i_k = 0$.
(d) $i_{\overline{k}} = i_k = 1$, and either $v_{\overline{k}} \neq v_k$, or $u_{\overline{k}} \neq u_k$.
(e) $i_{\overline{k}} \neq i_k$ and it is not the case that $v_{\overline{k}} = v_k \Leftrightarrow u_{\overline{k}} \neq u_k$.

In each of these cases, it can be seen that the application of a REVEAL$(i_{\overline{k}}, v_{\overline{k}}, u_{\overline{k}})$ operation to $\mathcal{O}$ when in a state of the form $\langle V, v, \perp, u_k, \perp, j_\ell \rangle$ or $\langle V, v, v_k, \perp, u_k, j_\ell \rangle$ (the only possible states of $\mathcal{O}$ in $E \cdot \hat{S} \cdot \hat{d}_k(I)$) will cause $\mathcal{O}$ to become upset and can return any value in $\{0, 1\}$. Thus, $c_{\overline{k}}$ is applicable to $E \cdot \hat{S} \cdot \hat{d}_k(I)$. Therefore, $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\overline{k}}$ is a node in $\mathcal{F}$, and $\mathcal{O}$ is upset in $E \cdot \hat{S} \cdot \hat{d}_k \cdot c_{\overline{k}}(I)$. This completes the proof of (4), and

thereby of the lemma. □

Using Lemma 8.12, we can show that starting from $E(I)$, if each of the two processes is about to apply its first REVEAL operation to $\mathcal{O}$, then the parameters of these two REVEAL operations satisfy certain properties.

LEMMA 8.13. *Consider any node $E \cdot S$ in $\mathcal{F}$ such that S contains no* REVEAL *step to $\mathcal{O}$ and both processes have a* REVEAL *operation to $\mathcal{O}$ pending in $E \cdot S(I)$. For each $k \in \{0, 1\}$, let i_k, v_k, and u_k be the parameters of the* REVEAL *operation to $\mathcal{O}$ of P_k that is pending in $E \cdot S(I)$. Then*

(a) *for each $k \in \{0, 1\}$, v_k is the response of an* ENROLL(i_k) *step to $\mathcal{O}$ contained in S;*
(b) *i_0 and i_1 cannot be both equal to 0;*
(c) *if $i_0 = i_1 = 1$, then $v_0 = v_1$ and $u_0 = u_1$;*
(d) *if $i_0 \neq i_1$, then $v_0 = v_1 \Leftrightarrow u_0 \neq u_1$.*

Proof. For each $k \in \{0, 1\}$, let d_k be a step of P_k and $c_{\overline{k}}$ be a step of $P_{\overline{k}}$ such that $E \cdot S \cdot d_k \cdot c_{\overline{k}}$ is a node in $\mathcal{F}$. By the specification of booster, if any of the properties (a)–(d) failed to hold, $\mathcal{O}$ would be upset in $E \cdot S \cdot d_k \cdot c_{\overline{k}}(I)$ for any $k \in \{0, 1\}$. This would contradict Lemma 8.12. Thus, (a)–(d) must all hold. □

8.2. Stage 2 of the proof. We now describe an algorithm that solves equality negation between two processes, Q_0 and Q_1, and uses one less booster object than $\mathbf{A}_{en}$. Intuitively, the algorithm works as follows: Each process Q_k simulates the actions of the corresponding process P_k in an execution of $\mathbf{A}_{en}$, and uses this execution to determine what value to decide. In this simulation, the processes are allowed to access all objects used by $\mathbf{A}_{en}$ except $\mathcal{O}$, each of which is initialized to the state it has in configuration $E(I)$.

The simulation of P_k by Q_k starts by pretending that P_k is in the state it has in configuration $E(I)$. Based on the current state of (the simulated) P_k, Q_k determines the operation *op* that P_k would apply next, and the object O to which *op* would be applied. If $O \neq \mathcal{O}$, then Q_k has access to O and can apply *op* to it directly to determine the response and update the (simulated) state of P_k accordingly. If, however, $O = \mathcal{O}$, this is not possible, since Q_k is not allowed to access $\mathcal{O}$. Instead, in this case, Q_k acts as follows:

- If *op* is an ENROLL(i) operation, P_k pretends that it applied this operation to $\mathcal{O}$ and received as the response the initial value for which Q_k must solve equality negation. In other words, Q_k pretends that when P_k enrolls in $\mathcal{O}$ it receives as the "challenge" with which to solve equality negation just the input value for which Q_k is required to solve equality negation! Q_k then updates the (simulated) state of P_k as if this step were executed and proceeds with the simulation of the next operation of P_k.
- If *op* is a REVEAL operation, Q_k does *not* simulate any more steps of $\mathbf{A}_{en}$. Instead, it decides a value based on the parameters of this REVEAL operation and the parameter of the first ENROLL operation to $\mathcal{O}$ that it previously simulated.

The algorithm is described more formally in Figure 8.2. A few remarks about the conventions used in the pseudocode are in order. Recall, from Figure 5.1, that Apply(P_k, op, O) is the procedure by which P_k invokes operation *op* on object O; it returns the result of this invocation (and updates the state of O accordingly). In addition, we assume that we are given two functions that describe the behavior of processes P_0 and P_1 in $\mathbf{A}_{en}$. (These correspond to the functions ν and τ discussed when we formally defined processes in section 2.2.)

Shared: every object other than $\mathcal{O}$ used in algorithm $\mathbf{A}_{en}$,
each initialized to the state it has in $E(I)$
S : auxiliary variable : schedule of $\mathbf{A}_{en}$ applicable to $E(I)$, initially empty
Code for process Q_k, $k \in \{0, 1\}$

```
1  init := initial value of Q_k
2  state := state of P_k in E(I)
3  j := parameter of the ENROLL operation to O by P_k pending in E(I)
4  while Q_k has not decided do
5      ⟨op, O⟩ := NextOp(P_k, state)
6      if O ≠ O then
7        ⟦ res := Apply(P_k, op, O); S := S·(P_k, op, O, res) ⟧
8          state := NextState(P_k, state, res)
9      else (* O = O *)
10         if op = ENROLL(i) then
11             S := S·(P_k, ENROLL(i), O, init)
12             state := NextState(P_k, state, init)
13         else
14             let i, v and u be such that op = REVEAL(i, v, u)
15             if i ≠ j then
16                 if v = init then decide ū
17                 else decide u
18             else
19                 if i = 0 then (* i = j = 0 *)
20                     decide u
21                 else (* i = j = 1 *)
22                     if v = init then decide u
23                     else decide ū
```

FIG. 8.2. *Solving equality negation for two processes using* $\mathbf{A}_{en}$ *with one less* booster *object.*

- NextOp(P_k, s): returns the pair $\langle op, O\rangle$, where the next operation that P_k executes in $\mathbf{A}_{en}$ when in state s is to apply op to object O.
- NextState(P_k, s, r): returns the state that P_k enters in $\mathbf{A}_{en}$ if it receives response r from the operation it invokes when in state s.

In the algorithm we also use an auxiliary variable S whose value, informally speaking, is the schedule of steps of $\mathbf{A}_{en}$ that have been simulated by Q_0 and Q_1 so far. This variable is not needed by the algorithm, but it is useful in proving its correctness (see Lemma 8.14). In one atomic step, besides accessing an ordinary (local or shared) variable, each process can also modify the auxiliary variable S. To emphasize this, we bracket with "$[\![\cdots]\!]$" the line that corresponds to a single atomic step affecting both an ordinary variable and the auxiliary variable S (line 7).

LEMMA 8.14. *If* $\mathbf{A}_{en}$ *is an equality negation algorithm for processes* P_0 *and* P_1*, then the algorithm in Figure* 8.2 *also solves equality negation for processes* Q_0 *and* Q_1 *using one fewer* booster *object than* $\mathbf{A}_{en}$*.*

Proof. It is obvious from Figure 8.2 that the algorithm uses one booster object less than $\mathbf{A}_{en}$. In the remainder of this proof we show that the algorithm solves equality negation for Q_0 and Q_1. To this end, fix an arbitrary execution of the algorithm. We shall prove that in this execution, the three properties of equality

negation—termination, negation, and agreement—are satisfied.

First we establish some properties of this execution. Let S_i be the value of the auxiliary variable S when the schedule it contains consists of exactly i steps. By lines 7 and 11 of Figure 8.2, it is clear that for all i, j, where $0 \leq i \leq j$, if S_j is defined, then S_i is a prefix of S_j. A straightforward induction on i shows that for all $i \geq 0$ such that S_i is defined, the following invariants hold.

(a) S_i is a schedule of $\mathbf{A}_{en}$ that is applicable to $E(I)$ (hence, $E \cdot S_i$ is a node in $\mathcal{F}$) and contains no REVEAL step to $\mathcal{O}$.

(b) Let Q_k be the process that assigns S_i to S, and $state_k$ be the value to which Q_k sets its local variable *state* after assigning S_i to S (cf. lines 8 and 12). Then, for all $j \geq i$, such that S_j is defined and there is no step of P_k in S_j after S_i, $state_k$ is the state of P_k in configuration $S_j(I)$ of $\mathbf{A}_{en}$.

With these invariants, we are now ready to prove that the execution satisfies the three properties of equality negation.

Since only two processes execute the algorithm, the agreement requirement of equality negation is trivially satisfied. We now show that the execution satisfies termination. Suppose, for contradiction, that some correct process Q_k is correct and never decides. This means that the **while** loop of Q_k does not terminate, and hence S_i is defined for all integers $i \geq 0$. By invariant (a), S_i is applicable to $E(I)$ and contains no REVEAL step to $\mathcal{O}$ for all $i \geq 0$. Thus, there is an infinite schedule S^* of $\mathbf{A}_{en}$ that is applicable to $E(I)$, contains infinitely many steps of P_k, and does not contain a REVEAL step to $\mathcal{O}$. By Lemma 8.11, P_k does not decide in $E \cdot S'(I)$ for all prefixes S' of S^*. This contradicts the fact that $\mathbf{A}_{en}$ satisfies the termination property. Thus, the execution satisfies termination.

It remains to show that the execution satisfies the negation property. Assume that both Q_0 and Q_1 decide in the execution. (Otherwise, negation is trivially satisfied.) For each $k \in \{0, 1\}$, let $init_k$ be the initial value of process Q_k (i.e., the value assigned to local variable *init* of Q_k in line 1); let j_k be the value assigned to local variable j of process Q_k in line 3; and let $state_k$ be the value of local variable *state* of process Q_k when Q_k decides. Let S^* be the value of S when Q_0 and Q_1 have both decided (i.e., S^* is the final value of the auxiliary variable S). By the algorithm, the next operation that P_k will execute when in $state_k$ is a REVEAL operation to object $\mathcal{O}$; let i_k, v_k, and u_k be the parameters of that REVEAL operation. From invariant (b), it follows that the operation of P_k pending in $E \cdot S^*(I)$ is a REVEAL(i_k, v_k, u_k) to $\mathcal{O}$. By invariant (a), $E \cdot S^*$ is a node in $\mathcal{F}$, and S^* contains no REVEAL operations to $\mathcal{O}$. Thus, the hypothesis of Lemma 8.13 holds (with $S = S^*$); we will make use of parts (a)–(d) of this lemma below. By line 11 of the algorithm, the only return values of ENROLL operations to $\mathcal{O}$ that appear in S^* are $init_0$ and $init_1$. Thus, by Lemma 8.13(a), $v_0, v_1 \in \{init_0, init_1\}$. There are three cases.

Case 1. *For both* $k \in \{0, 1\}$, $i_k \neq j_k$: In this case, each Q_k decides by line 16 or 17 (depending on whether $v_k = init_k$). Also, $i_0 \neq i_1$ (because $i_0, i_1, j_0, j_1 \in \{0, 1\}$ and, by Lemma 8.8, $j_0 \neq j_1$). Thus, by Lemma 8.13(7.2), $u_0 = u_1$ if and only if $v_0 \neq v_1$.

By a case analysis of the four possible combinations of lines by which Q_0 and Q_1 can decide, we have that Q_0 and Q_1 decide the same value if and only if

$$
\begin{array}{lll}
 & (v_0 = init_0 \textbf{ and } v_1 = init_1 \textbf{ and } u_0 = u_1) & /* \ Q_0\text{'s line 16; } Q_1\text{'s line 16 } */ \\
\textbf{or} & (v_0 = init_0 \textbf{ and } v_1 \neq init_1 \textbf{ and } u_0 \neq u_1) & /* \ Q_0\text{'s line 16; } Q_1\text{'s line 17 } */ \\
\textbf{or} & (v_0 \neq init_0 \textbf{ and } v_1 = init_1 \textbf{ and } u_0 \neq u_1) & /* \ Q_0\text{'s line 17; } Q_1\text{'s line 16 } */ \\
\textbf{or} & (v_0 \neq init_0 \textbf{ and } v_1 \neq init_1 \textbf{ and } u_0 = u_1). & /* \ Q_0\text{'s line 17; } Q_1\text{'s line 17 } */
\end{array}
$$

Since $u_0 = u_1$ if and only if $v_0 \neq v_1$, the above expression is equivalent to

$$\begin{array}{rl} & (v_0 = \mathit{init}_0 \textbf{ and } v_1 = \mathit{init}_1 \textbf{ and } v_0 \neq v_1) \\ \textbf{or} & (v_0 = \mathit{init}_0 \textbf{ and } v_1 \neq \mathit{init}_1 \textbf{ and } v_0 = v_1) \\ \textbf{or} & (v_0 \neq \mathit{init}_0 \textbf{ and } v_1 = \mathit{init}_1 \textbf{ and } v_0 = v_1) \\ \textbf{or} & (v_0 \neq \mathit{init}_0 \textbf{ and } v_1 \neq \mathit{init}_1 \textbf{ and } v_0 \neq v_1). \end{array}$$

As argued above, $v_0, v_1 \in \{\mathit{init}_0, \mathit{init}_1\}$. Thus, the above disjunction is equivalent to $\mathit{init}_0 \neq \mathit{init}_1$. In other words, Q_0 and Q_1 decide the same value if and only if they have different initial values.

Case 2. *For both* $k \in \{0,1\}$, $i_k = j_k$: By Lemma 8.8, $j_0 \neq j_1$. Thus, for some $k \in \{0,1\}$, $i_k = j_k = 0$ and $i_{\overline{k}} = j_{\overline{k}} = 1$. Then Q_k decides by line 20 and $Q_{\overline{k}}$ decides by line 22 or 23 (depending on whether $v_{\overline{k}} = \mathit{init}_{\overline{k}}$). Thus, Q_k decides u_k. So we have that Q_0 and Q_1 decide the same value if and only if

$$\begin{array}{rll} & (v_{\overline{k}} = \mathit{init}_{\overline{k}} \textbf{ and } u_k = u_{\overline{k}}) & /* \ Q_{\overline{k}}\text{'s line 22 } */ \\ \textbf{or} & (v_{\overline{k}} \neq \mathit{init}_{\overline{k}} \textbf{ and } u_k = \overline{u}_{\overline{k}}). & /* \ Q_{\overline{k}}\text{'s line 23 } */ \end{array}$$

In this case, $i_k \neq i_{\overline{k}}$. By Lemma 8.13(7.2), $u_k = u_{\overline{k}}$ if and only if $v_k \neq v_{\overline{k}}$. Thus, the above expression is equivalent to

$$(v_{\overline{k}} = \mathit{init}_{\overline{k}} \textbf{ and } v_k \neq v_{\overline{k}}) \textbf{ or } (v_{\overline{k}} \neq \mathit{init}_{\overline{k}} \textbf{ and } v_k = v_{\overline{k}}).$$

S^* does not contain two ENROLL(0) steps: Otherwise, $\mathcal{O}$ would be upset in $E \cdot S^*(I)$—and therefore still upset after the application of the two REVEAL operations pending in $E \cdot S^*(I)$—contrary to Lemma 8.12. Since $j_k = 0$, P_k has an ENROLL(0) to $\mathcal{O}$ pending in $E(I)$. Hence, the unique ENROLL(0) to $\mathcal{O}$ in S^* is a step of P_k. By line 11, the response to that unique ENROLL(0) step to $\mathcal{O}$ in S^* is the initial value of P_k, init_k. By Lemma 8.13(a) and the fact that $i_k = 0$, v_k is the response of the unique ENROLL(0) step to $\mathcal{O}$ that is contained in S^*. Therefore, $v_k = \mathit{init}_k$. Thus, the above expression is equivalent to

$$(v_{\overline{k}} = \mathit{init}_{\overline{k}} \textbf{ and } \mathit{init}_k \neq v_{\overline{k}}) \textbf{ or } (v_{\overline{k}} \neq \mathit{init}_{\overline{k}} \textbf{ and } \mathit{init}_k = v_{\overline{k}}).$$

As argued earlier, $v_{\overline{k}} \in \{\mathit{init}_0, \mathit{init}_1\}$; so the above expression is equivalent to $\mathit{init}_k \neq \mathit{init}_{\overline{k}}$. In other words, Q_0 and Q_1 decide the same value if and only if they have different initial values.

Case 3. *For some* $k \in \{0,1\}$, $i_k \neq j_k$ *and* $i_{\overline{k}} = j_{\overline{k}}$: By Lemma 8.8, $j_k \neq j_{\overline{k}}$, so $i_k = i_{\overline{k}}$. By Lemma 8.13(b), i_k and $i_{\overline{k}}$ cannot both be 0. Hence, we must have $i_k = i_{\overline{k}} = j_{\overline{k}} = 1$ and $j_k = 0$. By Lemma 8.13(c), $v_0 = v_1$ and $u_0 = u_1$. Also, since in this case $i_k \neq j_k$, while $i_{\overline{k}} = j_{\overline{k}} = 1$, Q_k decides by line 16 or 17 (depending on whether $v_k = \mathit{init}_k$), and $Q_{\overline{k}}$ decides by line 22 or 23 (depending on whether $v_{\overline{k}} = \mathit{init}_{\overline{k}}$). Keeping in mind that $u_0 = u_1$, we have that Q_0 and Q_1 decide the same value if and only if

$$\begin{array}{rll} & (v_k = \mathit{init}_k \textbf{ and } v_{\overline{k}} \neq \mathit{init}_{\overline{k}}) & /* \ Q_k\text{'s line 16; } Q_{\overline{k}}\text{'s line 23 } */ \\ \textbf{or} & (v_k \neq \mathit{init}_k \textbf{ and } v_{\overline{k}} = \mathit{init}_{\overline{k}}). & /* \ Q_k\text{'s line 17; } Q_{\overline{k}}\text{'s line 22 } */ \end{array}$$

Since $v_0 = v_1$ and, as argued earlier, $v_0, v_1 \in \{\mathit{init}_0, \mathit{init}_1\}$, the above expression is equivalent to $\mathit{init}_k \neq \mathit{init}_{\overline{k}}$. Hence, Q_0 and Q_1 decide the same value if and only if they have different initial values. In all cases the negation property is satisfied. □

We now have all the pieces needed to prove the main result of this section.

THEOREM 8.15. *Type* booster *has consensus number one.*

Proof. Suppose, to the contrary, that booster has consensus number at least two. Then we can solve consensus for two processes using only booster objects and registers. By the universality of consensus [6], we can implement a negation object shared by two processes using only booster objects and registers. Hence, we can solve equality negation for two processes using only booster objects and registers. Let $\mathbf{A}_{en}$ be an equality negation algorithm for two processes that uses a minimal number of booster objects. By Theorem 3.1, $\mathbf{A}_{en}$ uses at least one booster object. By Lemma 8.14, however, we can solve equality negation for two processes using one booster object less than $\mathbf{A}_{en}$. This contradicts the definition of $\mathbf{A}_{en}$. □

9. Conclusion. In this paper we showed that if nondeterministic object types are allowed, then the only robust wait-free hierarchy is the trivial one, which lumps all types into level one. In contrast, if only deterministic types are allowed, then the consensus hierarchy is robust [1, 15]. Our understanding of the results regarding deterministic types is far from complete. In view of the sharply contrasting results regarding the robustness of wait-free hierarchies for deterministic and nondeterministic types it would be interesting to understand the fundamental reason(s) why the restriction to determinism inherently rules out types with properties like negation and booster.

The unfortunate fact that (nontrivial) wait-free hierarchies are not robust suggests that it would be desirable to investigate other ways of classifying types according to their strength. A classification based on consensus numbers is computability-based: it depends on whether objects of a certain type can be used to solve consensus for a certain number of processes with no regard as to the efficiency of this solution. It would be interesting to investigate complexity-based classifications, where the level of a type may depend not only on its ability to solve consensus but also on doing so efficiently (in some appropriate sense). Could a complexity-based hierarchy classify types according to their strength in (efficiently) implementing other types, while being robust in the sense that "powerful" types are not (efficiently) implementable by "weak" ones?

A different line of research might proceed from the observation that the types negation and booster, although well within the set prescribed by the general definition of type given in section 2.1, are not "natural" types. Is there a sensible definition of "natural" types that excludes negation, booster, and other such "anomalous" types without excluding any useful ones?

Appendix. Unsolvability of equality negation using only registers. In this appendix we prove Theorem 3.1, which states that there is no algorithm that solves equality negation for two processes and uses only registers. The proof is by contradiction. Suppose that there is an equality negation algorithm, $\mathbf{A}_{en}$, for processes P_0 and P_1 that uses only registers. For each $k \in \{0, 1\}$ and $v \in \{0, 1, 2\}$, consider a solo execution of process P_k in algorithm $\mathbf{A}_{en}$, where the initial value of P_k is v. Since $\mathbf{A}_{en}$ uses only registers (which are deterministic), the solo execution of P_k with initial value v is uniquely defined. Let Δ_k^v be the decision of P_k in this solo execution.

LEMMA A.1. *There exist* $\alpha_0, \alpha_1 \in \{0, 1, 2\}$ *such that* $\alpha_0 = \alpha_1$ *if and only if* $\Delta_0^{\alpha_0} = \Delta_1^{\alpha_1}$.

Proof. Suppose, for contradiction, that no choice of values for $\alpha_0, \alpha_1 \in \{0, 1, 2\}$ satisfies the required property. Thus, (a) for all $v \in \{0, 1, 2\}$, $\Delta_0^v \neq \Delta_1^v$; and (b) for all $v_0, v_1 \in \{0, 1, 2\}$ such that $v_0 \neq v_1$, $\Delta_0^{v_0} = \Delta_1^{v_1}$. By (b), $\Delta_0^0 = \Delta_1^1$, $\Delta_1^1 = \Delta_0^2$, and $\Delta_0^2 = \Delta_1^0$. Therefore, $\Delta_0^0 = \Delta_1^0$, which contradicts (a). □

Shared: all registers used in $\mathbf{A}_{en}$, each initialized as specified by $\mathbf{A}_{en}$
R_0, R_1: register, each initialized to $\perp$
Code for process Q_k, $k \in \{0,1\}$
1 $R_k :=$ initial value of Q_k
2 $w :=$ Execute$(P_k, \mathbf{A}_{en}, \alpha_k)$
3 **if** $w = \Delta_k^{\alpha_k}$ **then decide** R_k
4 **else decide** $R_{\overline{k}}$

FIG. A.1. *Solving consensus for processes Q_0 and Q_1 using $\mathbf{A}_{en}$.*

As the lemma below proves, the algorithm in Figure A.1 solves consensus for processes Q_0 and Q_1. Since this algorithm uses two registers in addition to the objects used by $\mathbf{A}_{en}$, and since $\mathbf{A}_{en}$ uses only regusters, we have a consensus algorithm for two processes that uses only registers. This contradicts the fact that no such algorithm exists [3, 13]. Therefore, $\mathbf{A}_{en}$ cannot use only registers, as wanted.

LEMMA A.2. *The algorithm in Figure* A.1 *solves consensus for processes Q_0 and Q_1.*

Proof. The algorithm in Figure A.1 satisfies termination, since $\mathbf{A}_{en}$ does.

To prove that the algorithm satisfies validity, it suffices to show that if process Q_k decides the value in register R_d in an execution of the algorithm in Figure A.1, then Q_d has previously written its initial value into R_d. (Note that R_d is written only once.) This property follows immediately if Q_k decides the value in R_k in line 3. It remains to cover the case where Q_k decides in line 4. In this case, we want to prove that before Q_k decides $R_{\overline{k}}$ in line 4, $Q_{\overline{k}}$ has previously written its initial value into $R_{\overline{k}}$ (line 1). For this, it suffices to show that the execution of P_k that Q_k simulated in line 2 is not a solo P_k execution of $\mathbf{A}_{en}$. (If the simulated execution is not a solo P_k execution, Q_k and $Q_{\overline{k}}$ execute their line 2 concurrently, and therefore $Q_{\overline{k}}$ executes line 1 before Q_k executes line 4.) To see why the execution of P_k that Q_k simulated is not a solo P_k execution, suppose the contrary. Then, by definition of $\Delta_k^{\alpha_k}$, P_k would have decided $\Delta_k^{\alpha_k}$ in that execution, and the value that Q_k assigned to its local variable w in line 2 would have been equal to $\Delta_k^{\alpha_k}$. But then Q_k would be deciding in line 3, not 4.

To prove that the algorithm satisfies agreement, consider any execution in which both Q_0 and Q_1 decide. Let w_0 and w_1 be the values of the local variables w of Q_0 and Q_1, respectively, assigned in line 2 during that execution. Consider the execution of algorithm $\mathbf{A}_{en}$ by P_0 and P_1 that Q_0 and Q_1 simulated in line 2. In this simulated execution P_0 and P_1 have initial values α_0 and α_1 and decide w_0 and w_1. By the negation property, $w_0 = w_1$ if and only if $\alpha_0 \neq \alpha_1$. Combining with Lemma A.1, we have $w_0 = w_1$ if and only if $\Delta_0^{\alpha_0} \neq \Delta_1^{\alpha_1}$. Since $w_0, w_1, \Delta_0^{\alpha_0}, \Delta_1^{\alpha_1} \in \{0,1\}$, it follows that $w_0 = \Delta_0^{\alpha_0}$ if and only if $w_1 \neq \Delta_1^{\alpha_1}$. By the algorithm in Figure A.1, for any $k \in \{0,1\}$, Q_k decides the value in R_k in line 3 if and only if $w_k = \Delta_k^{\alpha_k}$ and hence if and only if $w_{\overline{k}} \neq \Delta_{\overline{k}}^{\alpha_{\overline{k}}}$. This, in turn, happens if and only if $Q_{\overline{k}}$ decides the value in $R_{\overline{k}} = R_k$ in line 4. This means that both Q_0 and Q_1 decide the value in the same register R_d for some $d \in \{0,1\}$. By validity, a process decides the value in R_d only after Q_d has written its initial value into R_d. Since only one value is written into R_d, agreement follows. □

Acknowledgments. Our thinking on shared-memory distributed computing has been deeply influenced by discussions with Tushar Chandra, Prasad Jayanti, and

Sam Toueg. Faith Fich and the anonymous referees made many helpful comments on earlier versions of this paper. Patricia O'Brien suggested the title.

REFERENCES

[1] E. Borowsky, E. Gafni, and Y. Afek, *Consensus power makes (some) sense!*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 363–372.

[2] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg, *Wait-freedom versus t-resiliency and the robustness of wait-free hierarchies*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 334–343.

[3] B. Chor, A. Israeli, and M. Li, *On processor coordination using asynchronous hardware*, in Proceedings of the 6th ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, 1987, pp. 86–97.

[4] E. Dijkstra, *Solution of a problem in concurrent programming control*, Comm. ACM, 8 (1965), p. 569.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.

[6] M. Herlihy. *Wait-free synchronization*, ACM Transactions on Programming Languages and Systems, 11 (1991), pp. 124–149.

[7] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Transactions on Programming Languages and Systems, 12 (1990), pp. 463–492.

[8] P. Jayanti, *On the robustness of Herlihy's hierarchy*, in Proceedings of the 12th ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 145–158.

[9] P. Jayanti and S. Toueg, *Some results on the impossibility, universality, and decidability of consensus*, in Proceedings of the 6th International Workshop on Distributed Algorithms, Haifa, Israel, Springer-Verlag, Berlin, 1992, pp. 69–84.

[10] J. Kleinberg and S. Mullainathan, *Resource bounds and combinations of consensus objects*, in Proceedings of the 12th ACM Symposium on Principles of Distributed Computing, Ithaca, NY, 1993, pp. 133–144.

[11] B. Liskov and J. Guttag, *Abstraction and Specification in Program Development*, MIT Press, Cambridge, MA, 1986.

[12] W.-K. Lo, *More on t-resilence vs. wait-freedom*, in Proceedings of the 14th ACM Symposium on Principles of Distributed Computing, Ottawa, Ontario, Canada, 1995, pp. 110–119.

[13] M. Loui and H. Abu-Amara, *Memory requirements for agreement among unreliable asynchronous processes*, in Advances in Computer Research 4, JAI Press, Greenwich, CT, 1987, pp. 163–183.

[14] S. Moran and L. Rappoport, *On the robustness of* h_m^r, in Proceedings of the 10th International Workshop on Distributed Algorithms, Bologna, Italy, Springer-Verlag, Berlin, 1996.

[15] G. Peterson, R. Bazzi, and G. Neiger, *A gap theorem for consensus types*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 344–353.

[16] S. Plotkin, *Sticky bits and the universality of consensus*, in Proceedings of the 8th ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, 1989, pp. 159–175.

[17] O. Rachman, *Anomalies in the wait-free hierarchy*, in Proceedings of the 8th International Workshop on Distributed Algorithms, Terschelling, The Netherlands, Springer-Verlag, Berlin, 1994, pp. 156–163.

[18] E. Ruppert, *Determining Consensus Numbers*, Technical Report 303/96, Department of Computer Science, University of Toronto, Canada, 1996.

[19] E. Schenk, *Computability and Complexity Results for Agreement Problems in Shared Memory Distributed Systems*, Ph.D. thesis, University of Toronto, Canada, 1996.

[20] E. Schenk, *The consensus hierarchy is not robust (brief announcement)*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, p. 279.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 729–752

FROM GENE TREES TO SPECIES TREES*

BIN MA†, MING LI‡, AND LOUXIN ZHANG§

Abstract. This paper studies various algorithmic issues in reconstructing a species tree from gene trees under the duplication and the mutation cost model. This is a fundamental problem in computational molecular biology. Our main results are as follows.

1. A linear time algorithm is presented for computing all the losses in duplications associated with the least common ancestor mapping from a gene tree to a species tree. This answers a problem raised recently by Eulenstein, Mirkin, and Vingron [*J. Comput. Bio.*, 5 (1998), pp. 135–148].
2. The complexity of finding an optimal species tree from gene trees is studied. The problem is proved to be NP-hard for the duplication cost and for the mutation cost. Further, the concept of reconciled trees was introduced by Goodman et al. and formalized by Page for visualizing the relationship between gene and species trees. We show that constructing an optimal reconciled tree for gene trees is also NP-hard. Finally, we consider a general reconstruction problem and show it to be NP-hard even for the well-known nearest neighbor interchange distance.
3. A new and efficiently computable metric is defined based on the duplication cost. We show that the problem of finding an optimal species tree from gene trees is NP-hard under this new metric but it can be approximated within factor 2 in polynomial time. Using this approximation result, we propose a heuristic method for finding a species tree from gene trees with uniquely labeled leaves under the duplication cost. Our experimental tests demonstrate that when the number of species is larger than 15 and gene trees are close to each other, our heuristic method is significantly better than the existing program in Page's GeneTree 1.0 that starts the search from a random tree.

Key words. gene trees, species trees, NP-hardness, algorithms

AMS subject classifications. 68Q, 68W, 92B

PII. S0097539798343362

1. Introduction.

As DNA sequences become easier to obtain, in the field of evolutionary moleculor biology emphasis has been placed on constructing gene trees and, from the gene trees, reconstructing evolutionary trees for species (called *species trees*) [9, 11, 20]. The current strategy for reconstructing species trees is based on the separate consideration of distinct gene families represented by homologous sequences; these homologous sequences are assumed to evolve in the same way as species. However, because of the presence of paralogy, sorting of ancestral polymorphism, and horizontal transfer, gene trees and species trees are often inconsistent [21, 25, 31, 33] and a "correct" species tree may simply not exist. Hence, a fundamental problem that arises is how to reconcile different, sometimes contradictory, gene trees into a

*Received by the editors August 11, 1998; accepted for publication October 29, 1999; published electronically July 13, 2000. An extended abstract of this work was presented in the *Second Annual International Conference on Computational Molecular Biology*, New York, 1998, pp. 182–191.

http://www.siam.org/journals/sicomp/30-3/34336.html

†Department of Computer Science, City University of Hong Kong, 83 Tat Chee Avenue, Kowloon, Hong Kong (csbma@cityu.edu.hk). The research of this author was supported in part by HK RGC grants 9040297 and 9040352 and CityU Strategic grant 7000693.

‡Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1 (mli@math.uwaterloo.ca). The research of this author was supported in part by NSERC Operating grant OGP0046506, a CGAT grant, and a Steacie Fellowship. Part of this work was done at City University of Hong Kong.

§BioInformatics Center, Kent Ridge Digital Labs, 21 Heng Mui Keng Terrace, Singapore 119613 (lxzhang@krdl.org.sg). Current address: Department of Mathematics, National University of Singapore, Singapore 117543.

species tree [10]. This problem has been studied extensively for the last two decades. Several similarity/dissimilarity measures for gene trees and species trees have been proposed and efficient comparison methods have also been investigated. See, for example, [8, 14, 15, 18, 32].

This paper studies the problem of reconciling different gene trees into a species tree under two well-known duplication-based similarity measures. These measures were proposed by Goodman et al. [14], Page [23], and Guigó, Muchnik, and Smith [15]. Genes have *gene trees* because of gene replication. As a gene copy at a locus in the whole genome replicates and its copies are passed onto offsprings, branch points are generated. Since the gene copy has a single ancestral copy, the resulting history is a branching tree. Gene divergence causes all the inconsistencies among different gene trees. Such divergence can be the result of either speciation or duplication events [22]. If the common ancestry of two genes can be tracked back to a speciation event, then they are said to be related by *orthology*; if it is tracked back to a duplication event, then they are related by *paralogy* [10]. Taking orthology and paralogy into account, Goodman et al. proposed a similarity measure for annotating species trees with duplications, gene losses, and nucleotide replacements [14]. Later, Page developed a method based only on duplications for interpreting inconsistency between vertebrate globin gene trees and the species tree that is constructed from morphological data [23]; Guigó, Muchnik, and Smith elaborated the idea for locating the gene duplications in eukaryotic history [15].

A *species tree* can be defined as the pattern of branching of species lineages via the process of speciation. When species are split by speciation, the gene copies within species likewise are split into separate bundles of descent. Thus, gene trees are contained within species trees. However, a gene tree may disagree with the containing species tree because of the reasons mentioned above. The duplication and mutation costs were defined using a least common ancestor (LCA) mapping M from gene trees to a species tree. Assume that only genes from each contemporary species are presented in gene trees. In a gene tree, leaves denote the genes from contemporary species; internal nodes are considered as ancestral genes. We may think that an ancestral gene is uniquely determined by the subset of contemporary genes descending from it in the gene tree. Similarly, in a species tree, an internal node is considered as an ancient species (which might not exist today) and is determined by the contemporary species descending from it. We may denote a contemporary species and the genes from that species by the same label. The mapping M from a gene tree to a species tree just maps a contemporary gene to the corresponding species, and an ancestral one to the most recent species which contains that gene (as a subset). Hence, we call it the *LCA mapping* in this paper. When the gene and species trees are inconsistent, it maps an ancestral gene g, and its child gene $c(g)$ to the same ancient species. In this case, we say that a *duplication* happens at g. Furthermore, roughly speaking, the number of gene losses associated with g is defined as the total number of ancient species between $M(g)$ and $M(c(g))$ for all children $c(g)$. To measure the similarity between a gene and species trees, Page defined the duplication cost as the number of duplications, and Guigó, Muchnik, and Smith defined the mutation cost as the sum of the number of duplications and the number of gene losses [15]. The mutation cost is not only efficiently computable, as shown in [4] and [34] independently (see also [5]), but also biologically meaningful [18]. Reconstructing a global species tree is based on the parsimonious criterion of minimizing the concerned cost between the gene trees and the species tree. In their paper [15], Guigó, Muchnik, and Smith developed a heuristic method for the problem using a nearest neighbor interchange (NNI) search

algorithm and applied it to infer the most likely phylogenetic relationship among 16 major eukaryotic taxa from the sequences of 53 different genes. In spite of having several serious flaws, their work demonstrated the potential of these measures in studies of genome evolution.

The contributions of this paper are in three aspects. First, we study the properties of the LCA mapping as well as the duplication and mutation costs. In particular, we prove a less obvious fact that the duplication cost satisfies the triangle inequality (Lemma 5.1) and study the relation between the duplication cost and the best-known NNI distance. We also present a linear time algorithm for computing all the losses in all duplications (section 3). Secondly, the complexity of reconstructing a species tree from gene trees is investigated. We prove that the problem is NP-hard under the duplication cost and under the mutation cost. The concept of a reconciled tree was introduced by Goodman et al. [14] for studying hemoglobin gene phylogeny, where there were significant discrepancies between gene and organismal phylogenies; later it was formalized by Page [23] as a means of describing historical associations such as those between genes and species. We prove that finding the best reconciled tree from a gene tree is NP-hard. We also consider a general reconstruction problem and prove it to be NP-hard even for the NNI distance. These results justify the necessity of developing heuristic methods and experimental research for reconstructing species trees [15, 24]. Finally, we give a heuristic method for reconstructing species trees. To this end, we propose a new and efficiently computable metric, satisfying the metric axioms, based on the duplication cost. Under this new metric, we show that the problem of reconstructing a species tree from gene trees is NP-hard but can be approximated within factor 2 in polynomial time. Using this approximation result, we present a new heuristic method for reconstructing species trees from uniquely leaf-labeled gene trees under the duplication cost.

The rest of the paper is divided into six sections. In section 2, we define the concepts of gene duplications and losses, review the duplication cost and the mutation cost and their basic properties, and formalize three problems of reconstructing a species tree from gene trees. In section 3, we present a linear time algorithm for computing all the losses between a gene tree and a species tree. In section 4, we prove that the problems defined in section 2 are NP-hard. In section 5, a new metric is proposed based on the duplication cost. We prove that, under the new metric, reconstructing a species tree from gene trees is NP-hard but can be approximated within factor 2 in polynomial time. Then a new heuristic method for reconstructing species trees is proposed. Experimental results are given to demonstrate that our new heuristic works quite well. In section 6, we consider a general reconstruction problem and prove it to be NP-hard even for the popular NNI distance. In section 7, we discuss further research and open questions.

We refer the reader to [2, 13] for textbooks on NP-completeness and approximation algorithms.

2. Comparing gene and species trees: Duplications and losses. In this section we first define the gene trees and species trees. We then introduce the two duplication-based measures for comparing gene and species trees: the duplication and mutation costs. For their biological meaning, we refer the reader to [14, 15, 23].

2.1. Species trees and gene trees. For a set I of N biological taxa, the model for their evolutionary history is a rooted full binary tree T where there are N leaves each uniquely labeled by a taxon in I and $N-1$ unlabeled internal nodes. Here the term "full" means that each internal node has exactly two children. Such a tree is

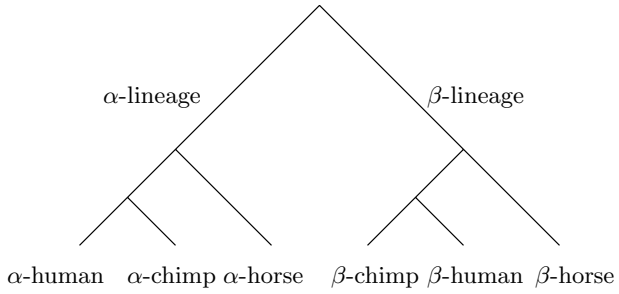


FIG. 1. *A gene tree based on α-hemoglobin and β-hemoglobin* [4].

called a *species tree*. In a species tree, we treat an internal node as a subset (called a *cluster*) which includes as its members its subordinate species represented by the leaves below it. Thus, the evolutionary relation "m is a descendant of n" is expressed using set-theoretic notation as "$m \subset n$."

The model for gene relationship is a rooted full binary tree with labeled leaves. Usually, a gene tree is constructed from a collection of genes each having several copies appearing in the studied species. For example, the gene family of hemoglobin genes in vertebrates contains α-hemoglobin and β-hemoglobin. A gene tree based on these two genes is illustrated in Figure 1 for human, chimpanzee, and horse. We use the species to label the genes appearing in it. Thus, the labels in a gene tree may not be unique. An internal node g corresponds to a multiset $\{x_1^{i_1}, x_2^{i_2}, \ldots, x_m^{i_m}\}$, where i_j is the number of its subordinate leaves labeled with x_j. The *cluster* of g is simply the set

$$S_g = \{x_1, x_2, \ldots, x_m\}.$$

Finally, we use $L(T)$ to denote the set of leaf labels in a species or gene tree T.

2.2. Gene duplications and the duplication cost. Given a gene tree G and a species tree S such that $L(G) \subseteq L(S)$. For any node $g \in G$, we define $M(g)$ to be the LCA of g in S, i.e., the smallest node $s \in S$ such that $S_g \subseteq s$. Here we used the term "smallest" to mean "farthest from the root." This correspondence M, first considered by Goodman et al. [14], is referred to as a mapping of G into S by Page [23]. We call M the *LCA mapping* from G to S. Obviously, if $g' \subset g$, then $M(g') \subseteq M(g)$, and any leaf is mapped onto a leaf with the same label. For an internal node g, we use $c(g)$ (sometimes $a(g)$ and $b(g)$) to denote a child of g and $G(g)$ the subtree rooted at g.

DEFINITION 2.1. *Let g be an internal node of G. If $M(c(g)) = M(g)$ for some child $c(g)$ of g, then we say $G(g)$ and $S(M(g))$ are root-inconsistent or a duplication happens at g.*

The total number $t_{dup}(G, S)$ of duplications happening in G under the LCA mapping M is proposed as a measure for the similarity between G and S [14, 23]. We call such a measure the *duplication cost*.

A subset A of (internal or leaf) nodes in a species tree S is *disjoint* if $x \cap y = \phi$ for any $x, y \in A$. For a disjoint subset A in S, the *restriction* of S on A is the smallest subtree of S containing A as its leaf set, denoted by $R_S(A)$. The *homomorphic subtree* $S|_A$ of S induced by A is a tree obtained from $R_S(A)$ by contracting all degree 2 nodes except the root. These concepts are illustrated in Figure 2. We state, without proofs, the following facts which will be used implicitly in the rest of this paper.

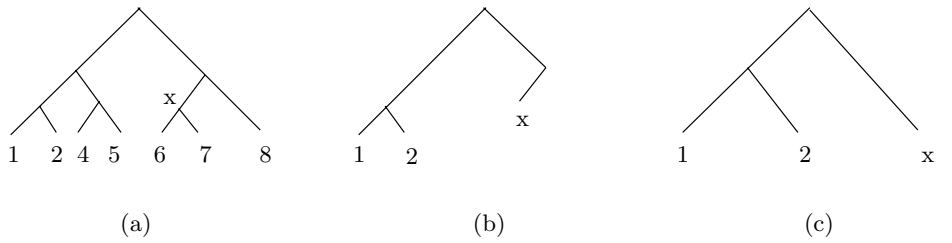


FIG. 2. (a) *A species tree S;* (b) *the restriction subtree $R_S(A)$ for $A = \{1, 2, x\}$;* (c) *the homomorphic subtree $S|_A$ induced by A.*

PROPOSITION 2.2. *Let G be a gene tree and S a species tree. Then $t_{dup}(G, S) = 0$ if and only if G is identical to $S|_{L(G)}$.*

PROPOSITION 2.3. *Let g be the root of G with children $a(g)$ and $b(g)$ and let s be the root of S with children $a(s)$ and $b(s)$. Then, if a duplication happens at g under the LCA mapping from G to S, then $t_{dup}(G, S) = 1 + t_{dup}(a(g), S) + t_{dup}(b(g), S)$.*

Furthermore, the duplication cost also satisfies the triangle inequality, which will be proved in Lemma 5.1 in section 5. Under the duplication cost, the problem of finding the "best" species tree from a set of known gene trees can be formulated as the following minimization problem.

Optimal Species Tree I (OST I).
INSTANCE: n gene trees $G_1, G_2, \ldots, G_n$.
QUESTION: Find a species tree S with the minimum duplication cost $\sum_{i=1}^{n} t_{dup}(G_i, S)$.

One can easily convert the above optimization problem into its *decision version* by having an extra integer c as input and requiring the minimum duplication cost to be less than c. This comment applies to all other optimization problems in this paper.

2.3. Gene losses and the mutation cost. After defining the duplication cost, we now introduce the mutation cost. We first define the *number of gene losses* associated with the LCA mapping M from G to S. Since $L(G) \subseteq L(S)$, $S|_{L(G)}$ is well defined and M induces an LCA mapping M' from G to $S|_{L(G)}$. Let g and g' be two nodes in $S|_{L(G)}$ such that $g \subseteq g'$. Define

$$d(g, g') = |\{h \in S|_{L(G)} \mid M'(g) \subset h \subset M'(g')\}|.$$

Let $a(g)$ and $b(g)$ denote the two children of g. The *number of losses* l_g associated to g is

$$l_g = \begin{cases} 0 & \text{if } M'(g) = M'(a(g)) = M'(b(g)); \\ d(a(g), g) + 1 & \text{if } M'(a(g)) \subset M'(g) \ \&\ M'(g) = M'(b(g)); \\ d(a(g), g) + d(b(g), g) & \text{if } M'(a(g)) \subset M'(g) \ \&\ M'(b(g)) \subset M'(g). \end{cases}$$

Note that our definition of $l(g)$ is a *generalization* of the one defined by Guigó, Muchnik, and Smith [15]. When $L(G) = L(S)$ and gene tree G is also uniquely labeled, our definition is identical to the one defined in [15]. The *mutation cost* is defined as the sum of t_{dup} and the total number of losses $l(G, S) = \sum_{g \in G} l_g$. This measure turns out to be identical to a biologically meaningful measure defined in Mirkin, Muchnik, and Smith [18] when G has the same number of uniquely labeled leaves as S. The problem of finding the "best" species tree from a set of known gene trees under this measure is formulated as the following.

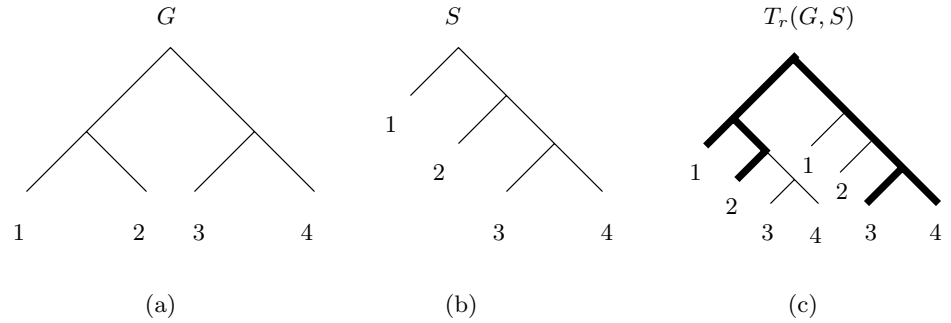


FIG. 3. (a) *A gene tree G;* (b) *a species tree S;* (c) *the reconciled tree* $T_r(G,S)$ *of G with respect to S.*

Optimal Species Tree II (OST II).
INSTANCE: n gene trees $G_1, G_2, \ldots, G_n$.
QUESTION: Find a species tree S with the minimum mutation cost $\sum_{i=1}^{n}(t_{dup}(G_i,S)+l(G_i,S))$.

2.4. Reconciled trees. For visualizing the relationship between gene and species trees, we use a third tree called the *reconciled tree* [14]. The reconciled tree has two important properties. The first property is that the observed gene tree is a subtree of the reconciled tree. The second property is that the clusters of the reconciled tree are all clusters of the species tree. Formally, the reconciled tree is defined as follows.

Let T' and T'' be two rooted trees; we use $T' \triangle T''$ to denote the rooted tree T obtained by adding a node r as the root and connecting r to $r(T')$ and $r(T'')$ so that T' and T'' are two subtrees rooted at the children of r. Further, let t be an internal node in T'; then $T'|_{\,t\to T''}$ denotes the tree formed by replacing the subtree rooted at t with T''. Similarly, $T'|_{\,t\to T_1,\, t'\to T_2}$ can be defined for disjoint nodes t and t'.

For a gene tree G rooted at g and a species tree S rooted at s such that $L(G) \subseteq L(S)$, let M be the LCA mapping from G to S and let $s' = M(a(g))$ and $s'' = M(b(g))$. The *reconciled tree* $R = R(G,S)$ of G with respect to S is defined as

$$(1)\qquad R = \begin{cases} R(G(a(g)),S) \triangle R(G(b(g)),S) & \text{if } s' = s'' = s, \\ S|_{\,s'\to R(G(a(g)),\, S(s'))} \triangle R(G(b(g)),S) & \text{if } s' \subseteq a(s),\ s'' = s, \\ S|_{\,s'\to R(G(a(g)),S(s')),\, s''\to R(G(b(g)),S(s''))} & \text{if } s' \subseteq a(s),\ s'' \subseteq b(s), \\ S|_{\,a(s)\to R(G,S(a(s)))} & \text{if } M(g) \subseteq a(s). \end{cases}$$

Such a concept is illustrated in Figure 3. An efficient algorithm was presented in [23] for computing a reconciled tree given a set of gene trees and a species trees. It is easy to see that the reconciled tree $R(G,S)$ satisfies the following three properties, of which the first two are mentioned above:

1. It contains G as a subtree, i.e., there is a subset L of leaves such that $R(G,S)|_L$ is isomorphic to G.
2. All clusters are in S, where a cluster is defined as a subset of species below an internal node in S (see subsection 2.1).
3. For any two children $a(g)$ and $b(g)$ of a node $g \in R(G,S)$, $a(g) \cap b(g) = \phi$, or $a(g) = b(g) = g$.

Actually, Page also defined the reconciled tree $R(G,S)$ as the smallest tree satisfying the above properties. However, these two definitions are not obviously equivalent. A

rigorous proof of this equivalence is needed and unknown. Reconstructing a species tree from a gene tree can be formulated as the following.

Optimal Species Tree III (OST III).
INSTANCE: A gene tree G.
QUESTION: Find a species tree S with the minimum duplication cost $t_{dup}(T_r(G,S),S)$.

3. Computing all loss events. When comparing a gene tree and a species tree, one may need to know both mutation cost and all "loss events" (to be defined). It is an open problem to compute all loss events efficiently [5]. In this section, we will develop a linear time algorithm to solve the problem. In the rest of this section, we assume that both gene tree G and species tree S are uniquely leaf labeled and $L(G) = L(S)$. We first introduce the concept of the gene loss events.

Let $u \in G$ and a duplication d_u occur at u. Recall that $S(M(u))$ denotes the subtree of S below $M(u)$. A node $v \in S(M(u))$ is *mixed* in the duplication d_u if $v \cap c(u) \neq \phi$ for any child $c(u)$ of u; it is *speciated* if $v \cap a(u) \neq \phi$ but $v \cap b(u) = \phi$ or vice versa; it is *gapped* if $v \cap c(u) = \phi$ for any $c(u)$. Finally, we say that a *loss event* occurs at a maximal speciated/gapped node in d_u. Note that a unique loss event occurs at some node on the path from $M(u)$ to any leaf in S. Figure 4 presents a mapping from a gene tree G (in (b)) to a species tree S (in (a)). Three duplications occur at nodes $r(g)$, $\{4,5,6\}$, and $\{7,8,9\}$ that are shown in (c), (d), and (e), respectively, where mixed nodes are labeled with "+−," speciated nodes with "+" or "−" depending on which intersection is empty, and gapped nodes are not labeled. All 14 loss events are marked by square boxes.

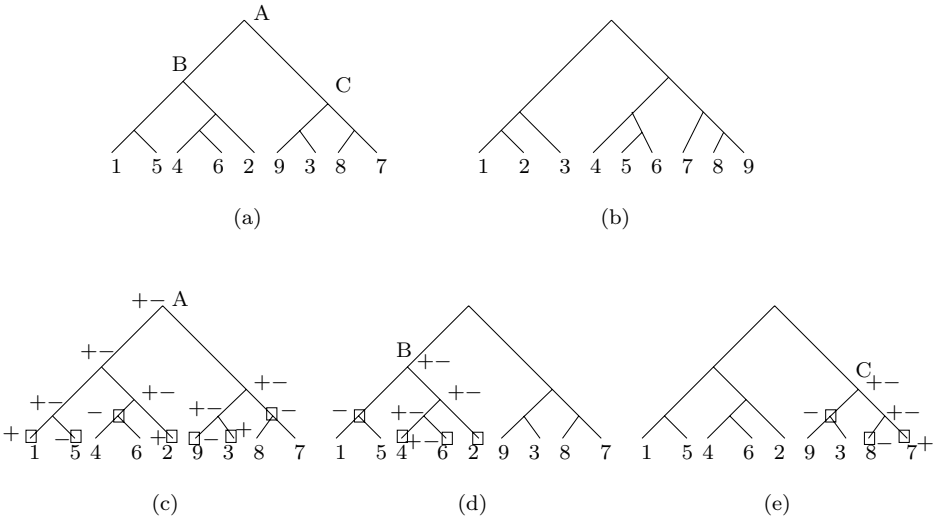


FIG. 4. *Duplications between a species tree* (a) *and a gene tree* (b).

Formally, the problem of computing all the loss events is formulated as follows. Given a gene tree G and a species tree S such that $L(G) = L(S)$, to find for each duplication d occurring at a node $g \in G$, the subtree $S(M^{-1}(g))$ of S with all the loss events as its leaves. For example, for the gene tree and species tree illustrated in Figure 4, the output is the three subtrees shown in Figure 5.

Note that the species tree S and gene tree G are rooted. We first impose an arbitrary ordering on the children of each node and produce an in-order traversal of G and S, respectively. Recall that in an in-order traversal, $i < j$ if and only if i is in

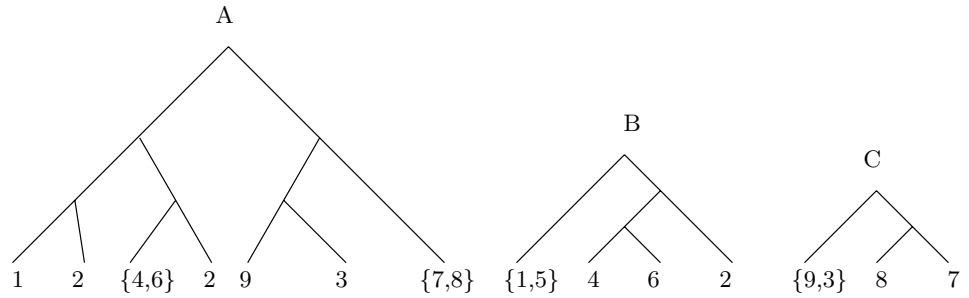


FIG. 5. *Output from computing all the loss events.*

the left subtree of j [2]. Without loss of generality, we may assume that each node of S is labeled by a number $k \leq 2n-1$, which is called the *in-order number* of the node. Preprocess the tree S in $O(n)$ steps so that an LCA query can be answered in constant time [16, 28]. Using this preprocessing, we can also compute the LCA mapping from G to S in linear time [34].

We store M in G as follows. To each node x in G, we associate a pair of $\langle i, j\rangle$, where i is its in-order number while j is the in-order number of $M(x)$ in S.

DEFINITION 3.1 (see [34]). *Let g be an internal node of G. It is said to be type-*1 *under the LCA mapping $M : G \to S$ if $M(a(g)) \subset M(g)$ and $M(b(g)) \subset M(g)$; it is type-*2 *if $M(a(g)) \subset M(g)$ and $M(b(g)) = M(g)$ or vice versa; it is type-*3 *if $M(a(g)) = M(b(g)) = M(g)$. Recall that $a(g)$ and $b(g)$ denote the children of g.*

To each node $y \in S$, we also assign an ordered pair $\langle i, n_{23}\rangle$, where i is its in-order number and n_{23} is the number of type-2 or type-3 nodes in G that is mapped to y. Observe that duplications occur at type-2 or type-3 nodes.

For a type-1 node g_1, $M(a(g_1))$ and $M(b(g_1))$ are distinct from $M(g_1)$. The unique path from $M(a(g_1))$ to $M(b(g_1))$ through $M(g_1)$ is called an *arc* in the mapping M from G to T. For our purpose, we say that such an arc *starts* at $M(g_1)$. We also say that such an arc *passes* through any intermediate nodes between $M(a(g_1))$ and $M(g_1)$ and between $M(b(g_1))$ and $M(g_1)$. For a type-2 node g_2, assume $M(a(g_2)) \subset M(g_2)$ and $M(b(g_2)) = M(g_2)$. The unique path from $M(g_2)$ to its descendant $M(a(g_2))$ is called an *arc* in the mapping M from G to T,[1] *starting* at $M(g_2)$. Such an arc *passes* through all intermediate nodes between $M(a(g_2))$ and $M(g_2)$. To each node y in S we associate a (linked) list $A(y)$ of all arcs passing or starting from y and two integers s_y and p_y, where s_y is the number of arcs starting at y, and p_y is the number of arcs passing y.

PROPOSITION 3.2. *The $A(y)$'s, s_y's, and p_y's can be computed in $O(l+2n)$ time, where l denotes the total number of loss events.*

Proof. We use breadth-first search starting from the root on the gene tree G. For each node $x \in G$, if $M(x) \neq M(a(x))$ and $M(x) \neq M(b(x))$, then $M(a(x)), M(b(x))$ is below $M(x)$ in S, we use the in-order numbers of $M(x)$, $M(a(x))$, and $M(b(x))$ to travel down from $M(x)$ to $M(a(x))$ and $M(b(x))$ in S, and add the arc $(a(x), b(x))$ to the list $A(y)$ and update s_y and p_y for each node y on the arc. If x is a type-2 node, let $M(x) \neq M(a(x))$ but let $M(x) = M(b(x))$. Then, we use the in-order numbers of $M(x)$, $M(a(x))$ to travel down from $M(x)$ to $M(a(x))$, during which we add the arc $(x, a(x))$ to the list $A(y)$ and update s_y and p_y for each node y on the arc.

[1] In [34], this is called a path.

Now we analyze the time complexity. For each node x, we take $O(d(M(a(x)), M(b(x))))$ in total to update the linked lists $A(y)$, the starting numbers s_y, and passing numbers p_y of nodes y on the arc. Thus, the algorithm takes

$$\begin{aligned} t &= \sum_{x \in G-L(G)} d(M(a(x)), M(b(x))) \\ &= \sum_{y \in S-L(S)} |A(y)| \\ &= n-1+\sum_{y \in Mixed(S)} |A(y)| \\ &\leq n+l+t_{dup} \\ &\leq 2n+l, \end{aligned}$$

where the third equality follows from the fact that the number of duplications in which a node y is mixed is equal to $s_y + p_y - 1$ [34], and the first inequality is based on the fact that for each duplication, the number of losses is equal to one plus the number of mixed nodes [34]. This concludes the proof. □

PROPOSITION 3.3. *Let x be an internal node in the species trees S; then a loss event occurs at x in some duplication if and only if $s_{p(x)} + p_{p(y)} - p_x - s_x + n_{23} > 0$, where n_{23} is the number of type-2 or type-3 nodes mapped to x under the LCA mapping M.*

Proof. There are exactly $s_{p(x)} + p_{p(y)} - 1$ duplications in which the parent $p(x)$ of x is a mixed node [34]. On the other hand, there are $s_x + p_x - 1$ duplications in which x is a mixed node. Further, n_{23} of these duplications occur at x. Thus, there are exactly $s_{p(x)} + p_{p(y)} - s_x - p_x + n_{23}$ duplications in which $p(x)$ is a mixed node but x is a speciation, i.e., a loss event occurs at x if $s_{p(x)} + p_{p(y)} - p_x - s_x + n_{23} > 0$. This concludes the proof. □

Thus, by Proposition 3.3, we can list all the nodes at which a loss event occurs in $O(n)$ steps by traveling down the species tree S. Moreover, we need to find out for each loss event which duplication causes it. Recall that $A(y)$ denotes the set of all arcs that pass or start at y for each node $y \in S$. Let

$$A(y) = \{(x_i, x_i') \mid i \leq m\}$$

and let $A^{-1}(y) = \{M^{-1}x_i, M^{-1}x_i' \mid i \leq m\}$. The following proposition is a combination of Claim 1 and Claim 2 in the proof of Proposition 3.4 in [34].

PROPOSITION 3.4. *The homomorphic subtree $G|_{A_y^{-1}}$ contains all the duplication nodes z in which y is a mixed node.*

By Proposition 3.4, we have the following.

PROPOSITION 3.5. *An in-order traversal of $G|_{A_y^{-1}}$ can be computed in $O(|A^{-1}(y)|)$ steps.*

Proof. Let $|A^{-1}(y)| = k$. Then $k = O(m)$. Radix sort $A^{-1}(y)$ in $O(m)$ steps. Let $z_1, z_2, \ldots, z_k$ be the in-order list of leaves of $G|_{A_y^{-1}}$. Let $z'_{2j-1} = z_j$ and $z'_{2j} = \mathrm{LCA}(z_j, z_{j+1})$. Then $z'_1, z'_2, \ldots, z'_{2m-1}$ is the in-order traversal of $G|_{A_y^{-1}}$. □

Let $\mathrm{Dup}(G, S)$ denote the set of all duplications occurring under the LCA mapping from G to S. For each duplication $d \in \mathrm{Dup}(G, S)$, let $\mathrm{Loss}(d)$ denote the set of nodes of S on which a loss event occurs in d. For each node $y \in S_{loss} = \cup \mathrm{Loss}(d)$ on which a loss event occurs, let

$$S_y = \{d \in \mathrm{Dup(G)} \mid y \in \mathrm{Loss}(d)\}. \tag{2}$$

Since for a duplication d, a loss event occurs at a node y if and only if the parent $p(y)$ of y is a mixed node in d, but y is a speciation node. Then, by Proposition 3.4, an in-order traversal of S_y can be obtained from difference between the in-order traversal of $G|_{A^{-1}(p(y))}$ and $G|_{A^{-1}(y)}$, which takes at most $O(s_{p(y)}+p_{p_y}+s_y+p_y)$ steps. Therefore, we have the following.

PROPOSITION 3.6. *The S_y's for all nodes y can be computed in $O(l+n)$ steps.*

Proof. Do a breadth-first search for loss nodes in S. For each loss node y, we find the in-order traversals of $G|_{A^{-1}(y)}$ and $G|_{A^{-1}(p(y))}$ and then find S_y from them as described above. The complexity is

$$t = \sum_{y \in S_{loss}} (O(s_{p(y)} + p_{p(y)} + s_y + p_y)) \leq O(l + 2n).$$

This concludes the proof. □

Recall that, for each duplication d, we use $\text{Loss}(d)$ to denote the set of nodes on which a loss event occurs in d and let

$$L_d = \{y \in S \mid y \in \text{Loss}(d)\}. \tag{3}$$

Then, from all S_y constructed above, we can derive all L_d as follows.

PROPOSITION 3.7. *All L_d can be computed in $O(l)$ steps. Thus, all loss subtrees can be constructed in $O(l)$ steps.*

Proof. Radix sort S_{loss} and let

$$y_1, y_2, \ldots, y_c$$

be the in-order list of nodes in S_{loss}. For a duplication, we will keep L_d in a linked list which is denoted by the same symbol. Let there be m duplications $d_1, d_2, \ldots, d_m$. Initially, L_{d_i} is empty for every i. Then, we examine $S_{y_1}, S_{y_2}, \ldots, S_{y_c}$ in order one by one. First, for each $d \in S_{y_1}$, we insert y_1 in L_d. In general, after $S_{y_1}, S_{y_2}, \ldots, S_{y_i}$ have been examined, we search $S_{y_{i+1}}$ in the same way: for each $d \in S_{y_i}$, we insert y_{i+1} in L_d. After all S_{y_i} have been examined, L_d stores all the nodes on which a loss event occurs in duplication d. Actually, following the construction carefully, one will see that L_d is an in-order traversal. Therefore, one can easily construct the loss subtree for duplication d from L_d. The time bound is obvious. □

We have proved the following theorem.

THEOREM 3.8. *Given a gene tree G and a species tree S, Algorithm* A *constructs the loss subtrees in $O(n+l)$ time.*

ALGORITHM A.
INPUT: A gene tree G and species tree S.
1. Impose an arbitrary ordering on the children of each node and produce an in-order traversal of G and S, respectively. Assume i_x denotes the in-order number of x for $x \in G, S$.
2. Compute the LCA mapping $M : G \to S$. To each $x \in G$ assign a pair $< i_x, i_{M(x)} >$; To each $y \in S$ assign a pair $< i_y, n_{23} >$, where n_{23} is the number of type-2 or type-3 nodes in G that are mapped to y.
3. Compute the set S_{loss} of all the nodes in which a loss event occurs using Proposition 3.3.
4. For each $y \in S_{loss}$, compute the set S_y that is defined in equation (2).
5. For each duplication d, use the sets S_y to compute L_d that is defined in equation (3).
6. Reconstruct all the loss subtrees from the L_d's.

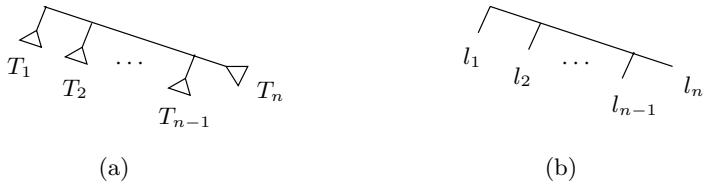


FIG. 6. (a) *The tree* $L[T_1, T_2, \ldots, T_n]$ *and* (b) *a line tree.*

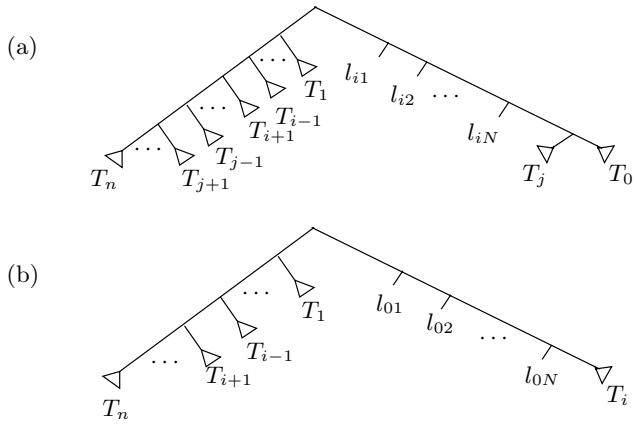


FIG. 7. (a) *The gene tree* G_{ij} *is constructed from the edge* (v_i, v_j). (b) *The gene tree* G_i *is constructed from the node* v_i.

4. The complexity of finding optimal species trees.

4.1. Optimal species tree I. Given n trees $T_1, T_2, \ldots, T_n$, we use $L[T_1, T_2, \ldots, T_n]$ to denote the tree T shown in Figure 6(a). When T_i is a single labeled node, the resulting tree is just a *line tree* as in Figure 6(b).

THEOREM 4.1. *The decision version of OST* I *is NP-complete.*

Proof. The problem is trivially in NP. To prove its NP-hardness, we reduce the independent set problem to OST I. Recall that the independent set problem is as follows: given a graph $G = (V, E)$ and an integer $d \leq |V|$, decide if G contains an independent set of size d, i.e., a subset of $V' \subseteq V$ such that $|V'| = d$ and no two nodes in V' are joined by an edge in E. Given an instance $G = (V, E)$ of the independent set problem, where $V = \{v_1, v_2, \ldots, v_n\}$, we construct a corresponding instance of OST I as follows.

Let $N = 5n^3$. For each v_i, we introduce N labels l_{ip}, $1 \leq p \leq N$, and a line tree $T_i = L[l_{i1}, l_{i2}, \ldots, l_{iN}]$. We also introduce extra N labels l_{0p}, $1 \leq p \leq N$, and a line tree $T_0 = L[l_{01}, l_{02}, \ldots, l_{0N}]$. For each pair (i, j) $(1 \leq i \neq j \leq n)$ such that $(v_i, v_j) \in E$, we define a tree G_{ij} with leaves labeled by $A = \{l_{ip} \mid 0 \leq i \leq n, 1 \leq p \leq N\}$ as shown in Figure 7(a). In G_{ij}, the left subtree is formed by connecting all T_p's $(p > 1)$ except for T_i and T_j by a line tree. Note that G_{ij} and G_{ji} have different right subtrees. Hence, we use two trees G_{ij} and G_{ji} to encode an edge (i, j). Finally, for each $v_i \in V$, we define a tree G_i with leaves labeled by A as shown in Figure 7(b). The left subtree of G_i is formed by connecting all T_p's except for T_i by a line tree, and the right subtree is a line tree with leaves l_{0k} $(1 \leq k \leq N)$ and l_{ik} $(1 \leq k \leq N)$ from left to right. Overall, we encode an edge (v_i, v_j) by two trees G_{ij} and G_{ji} and a node v_i by one

tree G_i. Obviously, such a construction can be carried out in polynomial time. The NP-hardness of OST I derives from the following lemma. □

LEMMA 4.2. *The graph G contains an independent set of size d if and only if there is a species tree S for all the gene trees G_{ij} and G_i constructed above with the duplication cost $c < (|E| + n - d + \frac{1}{2})N$.*

Proof. ($\Rightarrow$) Assume that G contains an independent set K of size d. Without loss of generality, we assume $V(K) = \{v_1, v_2, \ldots, v_d\}$. Then, we define a species tree S as

$$S = L[l_{n1}, \ldots, l_{nN}, \ldots, l_{(d+1)1}, \ldots, l_{(d+1)N}, l_{01}, \ldots, l_{0N}, l_{d1}, \ldots, l_{dN}, \ldots, l_{11}, \ldots, l_{1N}].$$

For each $i \le d$, $t_{dup}(G_i, S) = n - 1$. For each $i > d$, $t_{dup}(G_i, S) = N + n - 1$. Further, for any $(v_i, v_j) \in E$, either $i > d$ or $j > d$, and so

$$N \le t_{dup}(G_{ij}, S) + t_{dup}(G_{ji}, S) \le N + 2n. \tag{4}$$

Thus, the duplication cost c of S is

$$\begin{aligned}
&\sum_{(v_i,v_j)\in E} (t_{dup}(G_{ij}, S) + t_{dup}(G_{ji}, S)) + \sum_{1\le i\le n} t_{dup}(G_i, S) \\
&\quad \le |E|(N + 2n) + (n - d)(N + n - 1) + d(n - 1) \\
&\quad \le (|E| + n - d)N + 2n^3 \\
&\quad < \left(|E| + n - d + \frac{1}{2}\right) N.
\end{aligned}$$

($\Leftarrow$) We prove the converse by contradiction. Suppose that the optimal duplication cost is c for gene trees G_{ij} and G_i. Denote $A_i = \{l_{ip} \mid 1 \le p \le N\}$. Let S be an optimal species tree. Then one can define a total order $\prec$ on $\{A_i | 1 \le i \le n\}$ such that $LCA(A_i) \subset LCA(A_j)$ implies $A_i \prec A_j$. Suppose $A_{i_n} \prec A_{i_{n-1}} \prec \cdots \prec A_{i_0}$ is such a total order; then we define a line tree S' as

$$S' = L[l_{i_0 1}, \ldots, l_{i_0 N}, l_{i_1 1}, \ldots, l_{i_1 N}, \ldots, l_{i_n 1}, \ldots l_{i_n N}].$$

Let S' have duplication cost c'. Then we have the following two facts.

Fact 1. $c' \le 2\,n^3 + c$.

Proof. Since $S'|_{A_i} = T_i$, no duplication happens at all subtrees T_i ($0 \le i \le n$) in each gene tree $G_{i'j'}$ and $G_{i'}$. On the other hand, let u be any internal node on a right subtree of G_{ij}. If u is a parent of some l_{ip} ($0 \le p \le N - 1$) and u is not a duplication node in the mapping from G_{ij} to S, then it is easy to see $LCA(A_j) \subset LCA(A_i)$ and $LCA(A_0) \subset LCA(A_i)$. Thus $A_j \prec A_i$ and $A_0 \prec A_i$. Therefore, u is not a duplication node in the mapping from G_{ij} to S'. Note that two exceptions are the parent and the brother of l_{iN}. Similarly, for each internal node $x \in G_i$ that is a parent of l_{0p} ($1 \le p \le N - 1$), it will not be a duplication node for S' if it is not for S. Thus, the duplication cost for S' on all the right subtrees of gene trees is at most the cost for S plus $2n^2$.

Since there are at most $n' = n^2 + n(n - 1)(n - 2)$ extra internal nodes on the left subtrees of gene trees that have not been considered above, we have that $c' \le 2n^2 + n' + c \le 2n^3 + c$. This finishes the proof of Fact 1. □

Fact 2. $c' \ge (|E| + n - d + 1)N$.

Proof. Let $E_< = \{(v_i, v_j) \in E \mid \text{parent}_{S'}(l_{i1}), \text{parent}_{S'}(l_{j1}) \subset \text{parent}_{S'}(l_{01})\}$ and $V_< = \{v_i \in V | \text{parent}_{S'}(l_{i1}) \subset \text{parent}_{S'}(l_{01})\}$. If $G = (V, E)$ does not contain

(a) (b)

FIG. 8. *Rooted line trees.*

an independent set of size d, then $|E_<| - |V_<| \geq 1 - d$. In fact, this is trivial if $|V_<| < d$. Otherwise, let the largest independent set of restriction subgraph $G|_{V_<}$ be K'. Then $|K'| \leq d-1$. Since K' is largest, for any node $v \in V_< - K'$, $(v, v') \in E$ for some $v' \in K'$. This implies that $|E_<| \geq |V_<| - |K'| \geq |V_<| - d + 1$ or, equivalently, $|E_<| - |V_<| \geq 1 - d$ when $|V_<| \geq d$.

It is easy to verify that, for any i, j, if $(v_i, v_j) \in E_<$, then

$$t_{dup}(G_{ij}, S') + t_{dup}(G_{ji}, S') \geq 2N, \tag{5}$$

and if $v_i \notin V_<$, then

$$t_{dup}(G_i, S') \geq N. \tag{6}$$

By formulae (4), (5), and (6), we have

$$\begin{aligned} c' &\geq \sum_{v_i \in V - V_<} t_{dup}(G_i, S') + \sum_{(v_i,v_j) \in E - E_<} (t_{dup}(G_{ij}, S') + t_{dup}(G_{ji}, S')) \\ &\quad + \sum_{(v_i,v_j) \in E_<} (t_{dup}(G_{ij}, S') + t_{dup}(G_{ji}, S')) \\ &\geq N\,(n - |V_<|) + N\,(|E| - |E_<|) + 2\,N\,|E_<| \\ &\geq (|E| + n + |E_<| - |V_<|)N \\ &\geq (|E| + n - d + 1)N. \end{aligned}$$

Thus, Fact 2 is proved. □

Combining Fact 1 and Fact 2, we have that $c > (|E| + n - d + \frac{1}{2})N$, a contradiction. Thus, we finish the proof of the lemma. □

Remark 1. We have actually proved that OST I is NP-hard even for all gene trees with the same uniquely labeled leaves. Such a stronger conclusion will be used to prove that OST III is NP-hard in section 4.3.

Remark 2. Based on the above remark, we can also prove that the decision version of OST I remains NP-complete even for one gene tree that are not uniquely leaf-labeled. The proof of this result can be found in the proof of Theorem 4.7.

4.2. Optimal species tree II. Let C be a set of full binary trees G with leaves uniquely labeled by $L(G)$, and let T be a full binary tree with leaves uniquely labeled by $\sum_{G \in C} L(G)$. We say that C is *compatible* with T if for every $G \in C$, the homomorphic subtree $T|_{L(G)}$ of T induced by $L(G)$ is G, and it is *compatible* if it is compatible with some tree. Finally, recall that $L[z, w, v, u, x]$ denotes a rooted line tree with 5 leaves z, w, v, u, x as shown in Figure 8(a).

LEMMA 4.3. *If a collection C of 5-leave rooted line trees $L[y, w_i, v_i, u_i, x]$, $1 \leq i \leq k$, is compatible, then it is compatible with a rooted line tree $L[y, x_n, x_{n-1}, \ldots, x_1, x]$, where $\{x_1, x_2, \ldots, x_n\} = \cup_{i=1}^{k} \{u_i, v_i, w_i\}$.*

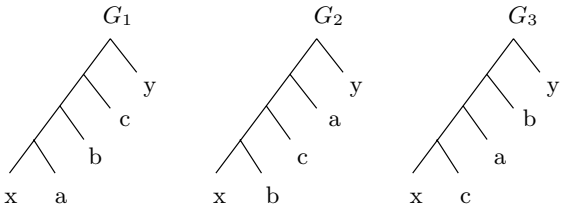


FIG. 9. *Three trees correspond to an ordered triple* (a, b, c).

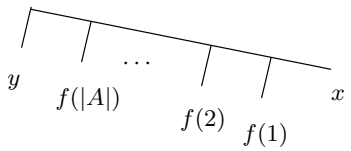


FIG. 10. *The species tree constructed from a cyclic ordering* f.

Proof. Choose a label z not in $\{x, y\}$ and $\cup_{i=1}^{k}\{u_i, v_i, w_i\}$. For each $t = L[y, w_i, v_i, u_i, x]$, we add an edge between z and the root so that the resulting tree t^z is an unrooted full binary tree in which each internal node has degree 3. It is not difficult to see that t^z is defined by the following set of quartets [29]:

$$Q(t^z) = \{xu_i|v_iz, xv_i|w_iz, xu_i|yz, xv_i|yz, xw_i|yz\}.$$

Suppose that C is compatible with a rooted full binary tree T; then $C^z = \{t^z \mid t \in C\}$ is compatible with T^z, and thus quartet set $\cup_{t\in C}Q(t^z)$ is compatible with T^z. By a lemma in [29], $\cup_{t\in C}Q(t^z)$ is compatible with a line tree $L[x, u_1, u_2, \ldots, u_{|A|}, y, z]$. This implies that C is compatible with the binary tree rooted at the internal node that is jointed with z (after the removal of z), which has the form shown in Figure 8(b). □

THEOREM 4.4. *The decision version of OST* II *is NP-complete.*

Proof. The problem is obviously in NP. To prove its NP-hardness, we now describe a transformation from the cyclic ordering problem [13] to OST II. The cyclic ordering problem is defined as follows.

> INSTANCE: A finite set A, and a collection C of ordered triples (a, b, c) of distinct elements from A.
> QUESTION: Is there a one-to-one function $f : A \to \{1, 2, \ldots, |A|\}$ such that, for each $(a, b, c) \in C$, we have either $f(a) < f(b) < f(c)$ or $f(b) < f(c) < f(a)$ or $f(c) < f(a) < f(b)$?

The problem is proved to be NP-complete in [12].

Suppose an instance (A, C) of the cyclic ordering problem is given. We construct for each ordered triple $\pi = (a, b, c) \in C$ three gene trees $G_1^\pi = L[y, c, b, a, x]$, $G_2^\pi = L[y, a, c, b, x]$, and $G_3^\pi = L[y, b, a, c, x]$ as shown in Figure 9, where x and y are two new labels fixed for all triples in C. Now, we consider a collection $G(C) = \{G_i^\pi \mid 1 \le i \le 3, \pi \in C\}$ of $3|C|$ gene trees. Obviously, such a construction can be carried out in polynomial time.

We claim that there is a species tree, with leaves $A \cup \{x, y\}$, which has mutation cost at most $14|C|$ if and only if A has a cyclic ordering.

Suppose a cyclic ordering f exists. Let $f(i)$ denote the ith smallest element in A and let $S = L[y, f(|A|), \ldots, f(2), f(1), x]$ as in Figure 10.

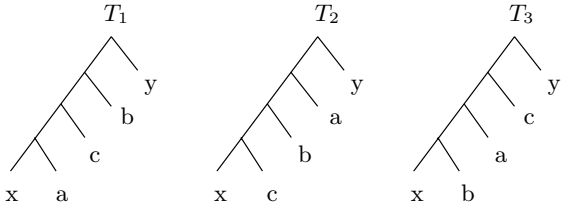


FIG. 11. *Three trees in the first column in Table* 1.

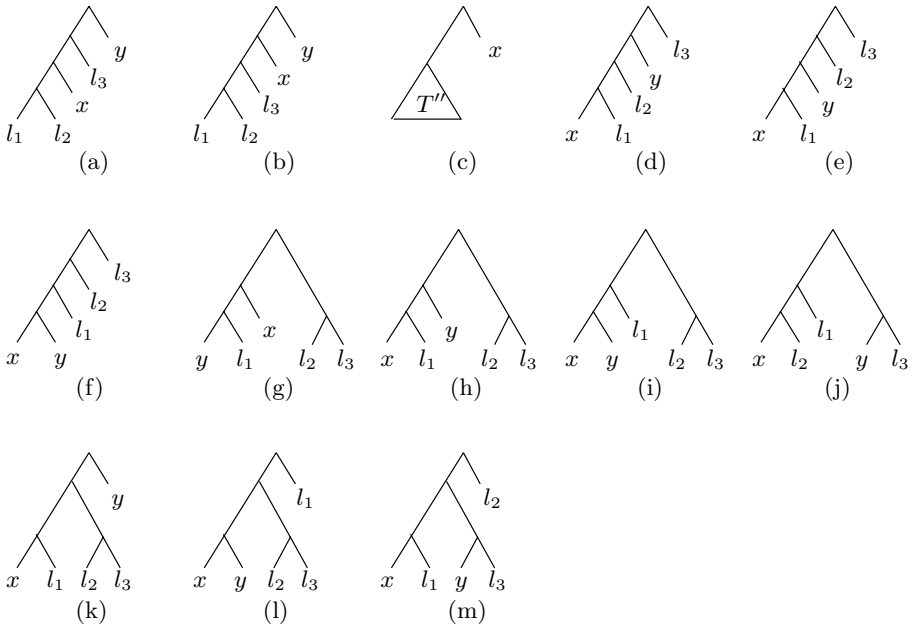


FIG. 12. *Cases* (a)–(m) *in the proof of Claim* 1.

For a triple $\pi = (a, b, c) \in C$, without loss of generality, we may assume that $f(a) < f(b) < f(c)$. Then G_1^π is the homomorphic subtree of S on $\{x, a, b, c, y\}$. Thus, $c(G_1, S) = 0$, $c(G_2, S) = 5$, and $c(G_3, S) = 9$. Hence, the total mutation cost over all $3|C|$ gene trees is $14|C|$.

Conversely, suppose that T is a species tree with leaves $A \cup \{x, y\}$ and with mutation cost at most $14|C|$. Then we have the following fact.

Fact. For any $\pi = (a, b, c) \in C$, the homomorphic subtree of T on $\{x, a, b, c, y\}$ is G_1, G_2, or G_3 as shown in Figure 9.

Proof. The homomorphic subtree T' of T on $\{x, a, b, c, y\}$ is a full binary tree with five labeled leaves. Assume that it is not one of G_1^π, G_2^π, or G_3^π. All possible homomorphic subtrees are illustrated in Figure 11 and Figure 12 and a case-by-case analysis of the mutation cost of G_1, G_2, and G_3 with T is shown in Table 1.

Hence, T has mutation cost at least $14|C| + 1$. This is a contradiction. Thus we conclude the fact. ∎

By Lemma 4.3, there exists a line tree such that for each triple $\pi = (a, b, c)$, the homomorphic subtree on $\{x, y, a, b, c\}$ is one of the gene trees $G_1^\pi, G_2^\pi, G_3^\pi$. It is not difficult to see that such a line tree induces a cyclic ordering. This concludes the proof of Theorem 4.4. ∎

TABLE 1
Case-by-case analysis of duplications.

Case	T_i	(a)	(b)	(c)	(d)
Cost	17	18	27	42, 45	29,32
Case	(e)	(f)	(g)	(h)	(i)
Cost	31,34	32,35	35	34	35
Case	(j)	(k)	(l)	(m)	
Cost	26,29	20	33	28, 29,32	

4.3. Optimal species tree III. To prove the hardness result, we need to establish Lemma 4.5 and Lemma 4.6, which are derived from the definition of reconciled trees. Recall that for a node g in a gene tree G, $G(g)$ denotes the subtree of G rooted at g.

LEMMA 4.5. *Given a gene tree G and a species tree S, let T_r be the reconciled tree of G with respect to S, and let g be an internal node in G. If g is mapped to $t \in T_r$ when G is considered as a subtree of T_r, then $T_r(t)$ is the reconciled tree of $G(g)$ with respect to $S(t)$.*

Proof. The lemma follows from the definition of reconciled trees. □

LEMMA 4.6. *Let T_r be the reconciled tree of G with respect to S. Then $t_{dup}(T_r, S) = t_{dup}(G, S)$.*

Proof. We prove this lemma by induction on the number of leaves in G. It is obviously true for a gene tree G that has only three leaves. Now assume that G has at least four leaves. Let t be the root of T_r with children $a(t)$ and $b(t)$, let g be the root of G with children $a(g)$ and $b(g)$, and let s be the root of S with children $a(s)$ and $b(s)$. We consider the following cases.

Case 1. $a(t) \cap b(t) = \phi$.

Note that $t = s$ and $a(t)$ and $b(t)$ are two clusters in S. Further, by the definition of reconciled trees, $a(t) \neq t$, and $b(t) \neq t$. Thus, t is not a duplication node under the LCA mapping from T_r to S. On the other hand, since G is identical to $T_r|_{L(G)}$, we have that $a(g) \subset a(t), b(g) \subset b(t)$ or $a(g) subset b(t)$, $b(g) \subset a(t)$. Let $a(g)$ and $b(g)$ be mapped to t_1 and t_2, respectively, when G is considered as a subtree of T_r. By Lemma 4.5, $T_r(t_1) = T_r(G(a(g)), S(t_1))$ and $T_r(t_2) = T_r(G(b(g)), S(t_2))$. By induction, $t_{dup}(T_r(t_1), S(t_1)) = t_{dup}(G(a(g)), S(t_1))$ and $t_{dup}(T_r(t_2), S(t_2)) = t_{dup}(G(b(g)), S(t_2))$. Since $a(g) \subseteq t_1$ and $b(g) \subseteq t_2$, g is not a duplication node under the LCA mapping from G to S. Thus,

$$\begin{aligned} t_{dup}(G, S) &= t_{dup}(G(a(g)), S(a(s))) + t_{dup}(G(b(g)), S(b(s))) \\ &= t_{dup}(T_r(a(t)), S(a(s))) + t_{dup}(T_r(b(t)), S(b(s))) \\ &= t_{dup}(T_r, S). \end{aligned}$$

Case 2. $a(t) = b(t)$.

Then $a(t) = b(t) = t = s$. Thus a duplication happens at t under the LCA mapping from T_r to S. Since $a(t) = b(t)$, then either $a(g)$ is mapped to $a(t)$ or $b(g)$ is mapped to $b(t)$. Otherwise, $a(t)$ or $b(t)$ contains G as a subtree, which contradicts the fact that T_r is the reconciled tree of G with respect to S. Without loss of generality, we may assume that the former is true. Let $b(g)$ be mapped to t'. Note that $t' \subseteq b(t) = s$. Under the LCA mapping from G to S, $a(g)$ is mapped to s. Thus, by induction,

$$\begin{aligned} t_{dup}(T_r, S) &= 1 + t_{dup}(T_r(a(t)), S) + t_{dup}(T_r(b(t)), S) \\ &= 1 + t_{dup}(a(g), S) + t_{dup}(G(b(g)), S(t')) \\ &= t_{dup}(G, S). \end{aligned}$$

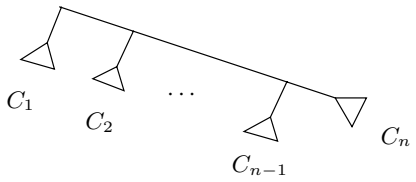


FIG. 13. *Connection of m gene trees in a right line tree.*

This proves Lemma 4.6. □

By Lemma 4.6, the problem OST III is a special case of the problem OST I in which each instance has only one gene tree. Unfortunately, such a problem is still NP-hard when a given gene tree is not a uniquely leaf-labeled tree.

THEOREM 4.7. *The decision version of OST* III *is NP-complete.*

Proof. Again, the problem is obviously in NP. To prove its NP-hardness, by Lemma 4.6, we need only to prove the following problem to be NP-hard:

Given a gene tree, find a species tree S with the minimum duplication cost $t_{dup}(G, S)$.

Given a class C of m gene trees with the same n uniquely labeled leaves, we construct a gene G by connecting all gene trees in C through a right line tree as shown in Figure 13. Since all gene trees in C have the same labeled leaves, we have that for any species tree S,

$$t_{dup}(G, S) = m - 1 + \sum_{1 \le i \le m} t_{dup}(G_i, S).$$

This finishes the reduction from an NP-hard problem OST I to the problem given above (see Remark 1 after Theorem 4.1). □

5. A heuristic method. We have proved that the problem of reconstructing an optimal species tree from gene trees is NP-hard. Therefore, there is unlikely an efficient algorithm for the problem. In this section, we will develop a heuristic method for it in a special case when all gene trees are uniquely leaf-labeled. Throughout this section, we will assume that trees are uniquely leaf-labeled without explicitly mentioning it.

5.1. A new metric. In this section, we introduce a new metric for measuring the similarity of two rooted full binary trees with uniquely labeled leaves based on the concept of duplications. Given two rooted full binary trees T_1 and T_2, in which each internal node has at least two children, we define the LCA mapping M from T_1 to T_2 as in section 2. We say a *duplication* happens at $x \in T_1$ under M if and only if for some child $c(x)$ of x, $M(c(x)) = M(x)$. We also use $t_{dup}(T_1, T_2)$ to denote the number of duplications occurring under the mapping M.

Let T be a rooted full binary tree. For any internal edge $e = (u, v)$, the *contraction tree* of T at e is the resulting tree after the removal of e and combining u and v into a new node p such that p is adjacent to all the neighbors of both u and v.

LEMMA 5.1. *The duplication cost satisfies the triangle inequality, i.e.,* $t_{dup}(T_1, T_3) \le t_{dup}(T_1, T_2) + t_{dup}(T_2, T_3)$ *for any three rooted full binary trees* T_1, T_2, *and* T_3 *with same uniquely labeled leaves.*

Proof. Let M_{ij} denote the LCA mapping from T_i to T_j. Now let T_1' be the resulting tree from T_1 by contracting all edges (u, v) such that $M_{12}(u) = M_{12}(v)$. Furthermore, let M_{12}' be the mapping from T_1' to T_2. We prove the following facts.

Fact 1. For any $m \in T_1'$, $M_{12}'(m) = m$. Thus, $t_{dup}(T_1', T_2) = 0$.

Proof. It follows from the definition of T_1'. □

Fact 2. $t_{dup}(T_1, T_3) \leq t_{dup}(T_1, T_2) + t_{dup}(T_2, T_3)$ if $t_{dup}(T_1', T_3) \leq t_{dup}(T_2, T_3)$.

Proof. Under the mapping M_{13}, a duplication happens at a node $n \in T_1$ if and only if $M_{13}(n) = M_{13}(c(n))$ for some child $c(n)$ of n. Let D denote the set of such duplication nodes in T_1 under M_{13}. We divide D into two disjoint subsets:

$$D_1 = \{n \in D \mid M_{12}(n) = M_{12}(c(n))\}$$

and

$$D_2 = \{n \in D \mid M_{12}(n) \neq M_{12}(c(n))\}.$$

Obviously, $|D_1| \leq t_{dup}(T_1, T_2)$ since any node in D_1 is also a duplication node under M_{12}. Furthermore, by the definition of T_1', $|D_2| \leq t_{dup}(T_1', T_3)$ since any node in D_2 is a duplication under the LCA mapping from T_1' to T_3. Hence, $t_{dup}(T_1, T_3) = |D_1| + |D_2| \leq t_{dup}(T_1, T_2) + t_{dup}(T_1', T_3) \leq t_{dup}(T_1, T_2) + t_{dup}(T_2, T_3)$ if $t_{dup}(T_1', T_3) \leq t_{dup}(T_2, T_3)$. This concludes the proof of Fact 2. □

Let $M_{12}'(n) = p$ and $M_{12}'(c(n)) = q$. Then, by Fact 1, $n = p$ and $c(n) = q$. If $M_{13}(n) = M_{13}(c(n))$, then all nodes in the path from $M_{23}(p)$ and $M_{23}(q)$ are mapped to the same node in T_3. This implies that $t_{dup}(T_1', T_3) \leq t_{dup}(T_2, T_3)$ and so, by Fact 2, $t_{dup}(T_1, T_3) \leq t_{dup}(T_1, T_2) + t_{dup}(T_2, T_3)$. This finishes the proof of Lemma 5.1. □

Now we define a new similarity measure between two rooted full binary trees as

$$d(T_1, T_2) = \frac{t_{dup}(T_1, T_2) + t_{dup}(T_2, T_1)}{2}.$$

Since the duplication cost is computable in linear time [34], the measure $d(.,.)$ is also efficiently computable. Further, it satisfies the three metric axioms.

PROPOSITION 5.2. *For any three full binary trees T_1, T_2, and T_3 with the same uniquely labeled leaves, $d(.,.)$ satisfies the following properties:*

(1) $d(T_1, T_2) = 0$ *if and only if* $T_1 = T_2$;

(2) $d(T_1, T_3) \leq d(T_1, T_2) + d(T_2, T_3)$ *for any* T_2;

(3) $d(T_1, T_2) = d(T_2, T_1)$.

In what follows, we call $d(.,.)$ the *symmetric duplication cost*. Interestingly, the symmetric duplication cost is closely related to the NNI distance for full binary trees, which was introduced independently in [19] and [27]. An NNI operation swaps two subtrees that are separated by an internal edge (u, v) as illustrated in Figure 14. The *NNI distance*, $D_{NNI}(T_1, T_2)$, between two full binary trees T_1 and T_2 is defined as the minimum number of NNI operations required to transform one tree into the other.

PROPOSITION 5.3. *For any species trees T_1 and T_2, $d(T_1, T_2) \leq D_{NNI}(T_1, T_2)$.*

Proof. Suppose T_1 is converted into T_2 by one NNI operation. Then, we can easily verify that $d(T_1, T_2) = 1$. Thus, $d(T_1, T_2) \leq D_{NNI}(T_1, T_2)$. Since $d(.,.)$ satisfies the triangle inequality, the result holds in general also. □

We now prove the following NP-completeness result.

THEOREM 5.4. *The decision version of finding an optimal species tree from a set of gene trees is NP-complete under the symmetric duplication cost.*

Proof. Obviously, it is in NP. In section 4.1, we have shown that OST I is NP-complete even for all gene trees with the same uniquely labeled leaves. Moreover, we may even assume that the duplication cost between any two gene trees is at least 2.

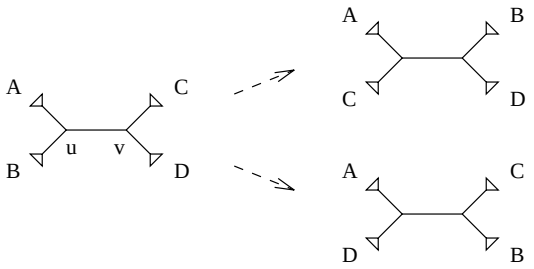


FIG. 14. *The two possible NNI operations on an internal edge* (u, v).

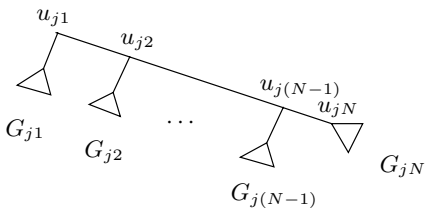


FIG. 15. *The tree* G'_j *of Theorem* 5.4.

We reduce this special case of OST I to the problem of finding an optimal species tree from gene trees under the symmetric duplication cost. Given an instance of OST I $I_1 = \{G_1, G_2, \ldots, G_n\}$ where each G_i is leaf uniquely labeled, and for any $i \neq j$, $L(G_i) = L(G_j)$ and $t_{dup}(G_i, G_j) \geq 2$. Assume that the leaf label set is $L = \{l_1, l_2, \ldots, l_m\}$ for each gene tree. For any species tree S with leaf label set L, if $S \neq G_i$ for any i, $1 \leq i \leq n$, then $t_{dup}(S, G_i) \geq 1$ for any i. If $S = G_i$ for some i in the range from 1 to n, then $t_{dup}(S, G_j) \geq 2$ for any $j \neq i$. Thus, for any species tree S with leaf label set L,

$$\sum_{j=1}^{n} t_{dup}(S, G_j) \geq n. \tag{7}$$

Let $N = 3n^2$. We introduce mN new labels l_{ik}, $1 \leq i \leq m$ and $1 \leq k \leq N$. For any j $(1 \leq j \leq n)$ and k $(1 \leq k \leq N)$, we construct G_{jk} from G_j by replacing the leaf l_i by l_{ik} for every i, $1 \leq i \leq m$. Let G'_j be the tree $L(G_{j1}, G_{j2}, \ldots, G_{jN})$ defined in Figure 15. Note that G'_j is a tree with mN labeled leaves. Finally, let $I_2 = \{G'_1, G'_2, \ldots, G'_n\}$. In order to finish the reduction, we now prove that I_1 has a solution with cost at most d if and only if I_2 has a solution with cost less than $(\frac{d+n}{2} + \frac{1}{4})N$.

Suppose S is a solution for I_1 with cost d. Let S' be the tree obtained from S by replacing each leaf l_i by a line tree $L(l_{i1}, l_{i2}, \ldots, l_{iN})$. Then it is easy to see that $t_{dup}(G_{jk}, S') = t_{dup}(G_j, S)$. Thus, $t_{dup}(G'_j, S') \leq t_{dup}(G_j, S)N + N$. Furthermore, since $t_{dup}(L(l_{i1}, l_{i2}, \ldots, l_{iN}), G'_k) = 0$, and S' has $n-1$ internal nodes that are not in $L(l_{i1}, l_{i2}, \ldots, l_{iN})$ for any i, $t_{dup}(S', G'_j) \leq n-1$. Therefore, the symmetric duplication cost of the solution S' for I_2 is

$$\sum_{i=1}^{n} \frac{t_{dup}(S', G'_j) + t_{dup}(G'_j, S')}{2} \leq \frac{1}{2}(dN + nN + n(n-1)) < \left(\frac{d+n}{2} + \frac{1}{4}\right) N.$$

Conversely, assume that the optimal solution for I_1 has duplication cost at least

$d+1$. Suppose S is a solution of I_2. For any $1 \le k \le N$, let $A_k = \{l_{ik} | 1 \le i \le m\}$, $S_k = S|_{A_k}$, and let u_k be the LCA of u_{jk} in S, where u_{jk}'s are the nodes in G'_j as shown in Figure 15. Note that u_k does not depend on the choice of j. Obviously, $u_N \subseteq \cdots \subseteq u_2 \subseteq u_1$. Assume that there are h indices k's ($1 \le k \le N-1$) satisfying $u_{k+1} \subset u_k$. Let these indices be $k_1, k_2, \ldots, k_h$. Then for any $k \ne k_t$ ($t = 1, 2, \ldots, h$), $1 \le k \le N-1$, u_{jk} is a duplication node in the mapping from G'_j to S. Hence, we have that

$$t_{dup}(G'_j, S) \ge \sum_{k=1}^{N} t_{dup}(G_{jk}, S) + N - 1 - h. \tag{8}$$

We use h_j to denote the number of duplications that occur on one of the nodes u_{k_1}, u_{k_2}, ..., u_{k_h} under the LCA mapping from S to G'_j. Let j' be the index that minimizes h_j over all j from 1 to n and let $h' = h - h_{j'}$. Assume that $u_{r_1}, u_{r_2}, \ldots, u_{r_{h'}}$ are the h' nonduplication nodes in the mapping from S to $G'_{j'}$. We have that $\{r_1, r_2, \ldots, r_{h'}\} \subseteq \{k_1, k_2, \ldots, k_h\}$ and $r_1 < r_2 < \cdots < r_{h'}$. Let $A = \bigcup_{t=1}^{h'} A_{r_t}$; then it is easy to verify that in the tree $S|_A$, for any $1 \le s \le h'$, $A_{r_s} \cap LCA(\bigcup_{t=s+1}^{h'} A_{r_t}) = \emptyset$. Thus,

$$t_{dup}(S, G'_j) \ge h_j + t_{dup}(S|_A, G'_j) \ge h - h' + \sum_{t=1}^{h'} t_{dup}(S_{r_t}, G_{jr_t}). \tag{9}$$

Combining formulae (7), (8), and (9), we have

$$\begin{aligned}
&\sum_{j=1}^{n} \left[t_{dup}(G'_j, S) + t_{dup}(S, G'_j) \right] \\
&\quad \ge \sum_{k=1}^{N} \sum_{j=1}^{n} t_{dup}(G_{jk}, S) + n(N - 1 - h') + \sum_{t=1}^{h'} \sum_{j=1}^{n} t_{dup}(S_{r_t}, G_{jr_t}) \\
&\quad \ge (d + n + 1)N - n.
\end{aligned}$$

Thus we know that for any solution of I_2, the cost is at least $\frac{(d+n+1)N-n}{2} \ge (\frac{d+n}{2} + \frac{1}{4})N$. □

5.2. A heuristic method for finding species trees. Although finding an optimal species tree from gene trees is NP-hard for the symmetric duplication cost $d(.,.)$, we have the following approximation result.

THEOREM 5.5. *There is a polynomial-time approximation of ratio* 2 *to the problem of finding an optimal species tree from gene trees with the symmetric duplication cost* $d(.,.)$.

Proof. Given an input of n gene trees $G_1, G_2, \ldots, G_n$, we compute $\sum_{i \ne j}^{n} d(G_i, G_j)$ for each $j \le n$ and output G_j with the minimum cost $\sum_{i \ne j}^{n} d(G_i, G_j)$ as the species tree. We now prove that the output species tree has at most twice the optimal symmetric duplication cost. Assume that G_1 is the output and S is an optimal species tree. Then

$$\begin{aligned}
\sum_{i \le n} d(G_i, G_1) &\le \frac{\sum_{i \le n} \sum_{j \le n} d(G_i, G_j)}{n} \\
&\le \frac{\sum_{i \le n} \sum_{j \le n} (d(G_i, S) + d(G_j, S))}{n} \\
&\le 2 \sum_{i \le n} d(G_i, S).
\end{aligned}$$

This proves Theorem 5.5. □

In general, the optimal species tree for a set of gene trees under the symmetric duplication cost is different from ones under the duplication and mutation costs. However, these trees should be quite similar to each other intuitively. Hence, based on Theorem 5.5, we propose the following heuristic method for the problem.

Search Paradigm

Input: Gene tree $G_1, G_2, \ldots, G_n$.

1. Find a gene tree $T' = T_k$ with the minimum symmetric duplication cost $\sum_{i \le n} d(T_i, T_k)$.
2. Search for the optimal species tree starting from T' using NNI, CP, or alternate NNI and CP.

Here cut and paste (CP) is also known as subtree pruning and regrafting [30]. According to the experimental research conducted by Page and Charleston [24], the best choice seems to be alternating between the NNI and CP method in step 2 of our heuristic method.

We have extensively tested our heuristic method and compared it with the algorithm that starts the search from a random tree. The latter was implemented in Page's package GeneTree Version 1.0. When running Page's algorithm, we start from a random tree and search near-optimal species trees using the method of alternating NNI and CP. We also use the method of alternating NNI and CP to do the search in our algorithm. When there are less than 10 species, and gene trees in each data set are chosen randomly, both algorithms perform well. They produce quickly species trees with optimal duplication costs. However, when there are over 15 species, and gene trees in a data set are closely related, which is usually true for practical molecular data, our algorithm performs much better. We have conducted 22 tests. We generated a set of gene trees as follows: (a) Generate a random tree R using an algorithm of Rémy [1, 26]; (b) repeatedly generate a tree by randomly choosing up to 10 NNI operations and applying these operations on R. The results are listed in Table 2 except for three unfinished tests in which the algorithm of searching from a random tree took over one hour and was stopped before finishing, but our algorithm finished within half a minute. Our algorithm found species trees with better duplication costs in all the cases and took much fewer CP and NNI operations (and hence much less time) to get the solution. We used a Pentium MMX-233 personal computer. In each of the 22 tests, our algorithm finished in less than half a minute, while the search-from-a-random-tree algorithm took more than one hour for 6 tests.

6. A general reconstructing problem. There is a large family of genes each having several distinct copies in the studied species. In order to derive a species tree that truly reflects the evolution of species, one needs full knowledge about which copies of the gene are comparable. This is usually impossible until a careful study of the species has been done. However, one may have different confidences in different genes. Hence, it is natural to propose the following general problem. We use I^+ to denote the set of positive integers and let m be any similarity measure between gene and species trees.

General Optimal Species Tree (GOST).

INSTANCE: A set of n gene trees $G_1, G_2, \ldots, G_n$, to each tree a confidence value $c_i \in I^+$ is associated.

QUESTION: Find a species tree S with the minimum cost $\sum_{i=1}^{n} c_i m(G_i, S)$.

TABLE 2

R = the algorithm that starts the search with a random tree, SP = our search paradigm. The last column contains the numbers of NNI/CP operations used by the two algorithms.

Data sets	Species	Gene trees	Alg.	Optimal trees	Dup. cost	NNI/CP operations
1	15	5	SP R	14	18	10628 37872
2	15	5	SP R	4	21	2848 53394
3	15	10	SP R	1	34	716 147540
4	15	10	SP R	1	32	1466 17228
5	17	5	SP R	3	16	2780 102758
6	17	5	SP R	2 999	17 21	994 872660
7	17	5	SP R	9 6	17	9221 42356
8	17	5	SP R	7	18	7310 105370
9	17	5	SP R	14 999	18 20	12638 878552
10	18	5	SP R	6 999	19 23	7490 1059318
11	18	5	SP R	6	19	7289 113472
12	18	5	SP R	2	20	1684 16462
13	18	5	SP R	6	16	6430 33968
14	18	5	SP R	1	17	1068 98694
15	19	7	SP R	1	22 23	1216 84170
16	19	7	SP R	2	26	2374 15880
17	20	5	SP R	7 999	13 15	8898 1326562
18	20	7	SP R	1 999	31 36	1430 1384018
19	20	8	SP R	4 999	38 43	6656 1323186

Clearly, GOST is NP-hard under the duplication cost and the mutation cost. For the NNI distance, the same conclusion also holds.

THEOREM 6.1. *The decision version of GOST is NP-complete for the NNI distance.*

Proof. We reduce the problem of computing NNI distance between two trees (see [3]) to GOST. Given are two binary trees T_1 and T_2 with n leaves. By applying an NNI operation to T_1, there are as many as $2n-2$ different resulting trees. Let T_3 be such a tree, i.e., $d_{NNI}(T_3, T_1) = 1$. We consider the following instance I of GOST:

$$I = \{T_1, T_2, T_3, c_1 = 2, c_2 = 2, c_3 = 1\}.$$

Let S be an optimal species tree for I. Then one can easily verify that $S = T_3$ if and only if $d_{NNI}(T_1, T_2) = d_{NNI}(T_1, T_3) + d_{NNI}(T_3, T_2)$. Note that the NNI distance

$d_{NNI}(T_1, T_2)$ is at most $n \log n + 2n$ [17]. If GOST is solved in polynomial time, we can compute $d_{NNI}(T_1, T_2)$ using an efficient search as follows. For each T_3 such that $d_{NNI}(T_1, T_3) = 1$, compute the optimal species tree S for the instance I defined above. If $S = T_3$, then we use T_3 to replace T_1, compute $d_{NNI}(T_3, T_2)$ inductively, and output $1 + d_{NNI}(T_3, T_2)$. This finishes the reduction and hence the proof. □

Note that the approximation algorithm in Theorem 5.5 cannot be generalized to GOST. Therefore, it is challenging to develop polynomial-time algorithms with good approximation factors for GOST under the various similarity measures.

7. Further research. Further studies on our topics in relation with parametric complexity classes have been carried out recently by Fellows et al. We refer the reader to [6, 7]. We end the paper with a list of open questions.

1. Is the definition of reconciled tree in section 2.4 identical to the one defined by Page (the smallest tree satisfying the three properties listed in section 2.4)?
2. Study the complexity of approximating the problems OST I, OST II, and OST III. Is it possible to develop efficient polynomial-time approximation algorithms for these problems?
3. Develop efficient polynomial-time approximation algorithms for GOST defined in section 6 under the various measures studied here.

REFERENCES

[1] L. Alonso and R. Schott, *Random Generation of Trees*, Kluwer Academic Publishers, Boston, 1995.

[2] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[3] B. DasGupta, X. He, T. Jiang, M. Li, J. Tromp, and L. Zhang, *On distance between phylogenetic trees*, in Proceedings of the 8th ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, SIAM, Philadelphia, PA, 1997, pp. 427–436.

[4] O. Eulenstein and M. Vingron, *On the Equivalence of Two Tree Mapping Measures*, Arbeitspapiere der GMD, 936, Bonn, Germany, 1996.

[5] O. Eulenstein, B. Mirkin, and M. Vingron, *Duplication-based measures of difference between gene and species trees*, J. Comput. Bio., 5 (1998), pp. 135–148.

[6] M. Fellows, M. Hallett, C. Korostensky, and U. Stege, *Analogs and duals of the MAST problem for sequences and trees*, in Proceedings of European Symposium on Algorithms (ESA'98), Venice, Italy, 1998, Lecture Notes in Comput. Sci. 1461, Springer-Verlag, Berlin, 1998, pp. 103–114.

[7] M. Fellows, M. Hallett, and U. Stege, *On the multiple gene duplication problem*, in Proceedings of the 9th International Symposium on Algorithms and Computation (ISSAC'98), Taejon, Korea, 1998, Lecture Notes in Comput. Sci. 1533, Springer-Verlag, Berlin, 1998, pp. 347–356.

[8] M. Farach and M. Thorup, *Fast comparison of evolutionary trees*, in Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms, 1994, Arlington, VA, SIAM, Philadelphia, PA, pp. 481–488.

[9] J. Felsenstein, *Phylogenies from molecular sequences: Inference and reliability*, Ann. Review Genet., 22 (1988), pp. 521–561.

[10] W. Fitch, *Distinguishing homologous and analogous proteins*, Syst. Zool., 19 (1970), pp. 99–113.

[11] W. Fitch and E. Margoliash, *Construction of phylogenetic trees*, Science, 155 (1967), pp. 279–284.

[12] Z. Galil and N. Megiddo, *Cyclic ordering is NP-complete*, Theoret. Comput. Sci., 5 (1977), pp. 179–182.

[13] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, New York, 1979.

[14] M. Goodman, J. Czelusniak, G. W. Moore, A. E. Romero-Herrera, and G. Matsuda, *Fitting the gene lineage into its species lineage, a parsimony strategy illustrated by cladograms constructed from globin sequences*, Syst. Zool., 28 (1979), pp. 132–163.

[15] R. Guigó, I. Muchnik, and T. Smith, *Reconstruction of ancient molecular phylogeny*, Mol. Phy. and Evol., 6 (1996), pp. 189–213.
[16] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.
[17] M. Li, J. Tromp, and L. Zhang, *Some notes on the nearest neighbor interchange distance*, J. Theoret. Bio., 182 (1996), pp. 463–467.
[18] B. Mirkin, I. Muchnik, and T. Smith, *A biologically meaningful model for comparing molecular phylogenies*, J. Comput. Bio., 2 (1995), pp. 493–507.
[19] G.W. Moore, M. Goodman, and J. Barnabas, *An iterative approach from the standpoint of the additive hypothesis to the dendrogram problem posed by molecular data sets*, J. Theoret. Bio., 38 (1973), pp. 423–457.
[20] M. Nei, *Molecular Evolutionary Genetics*, Columbia University Press, New York, 1987.
[21] J. E. Neigel and J. C. Avise, *Phylogenetic relationship of mitochondrial DNA under various demographic models of speciation*, in Evolutionary Processes and Theory, Academic Press, Orlando, FL, 1986, pp. 515–534.
[22] S. Ohno, *Evolution by Gene Duplication*, Springer-Verlag, Berlin, 1970.
[23] R. Page, *Maps between trees and cladistic analysis of historical associations among genes, organisms, and areas*, Syst. Bio., 43 (1994), pp. 58–77.
[24] R. Page and M. Charleston, *From gene to organismal phylogeny: Reconciled trees and the gene tree/species tree problem*, Mol. Phy. and Evol., 7 (1997), pp. 231–240.
[25] P. Pamilo and M. Nei, *Relationship between gene trees and species trees*, Mol. Bio. Evol., 5 (1988), pp. 568–583.
[26] J. L. Rémy, *Un procédé itératif de dénombrement d'arbres binaires et son application á leur génération aléatoire*, R.A.I.R.O. Informatique Théorique, 19 (1985), pp. 179–195.
[27] D. F. Robinson, *Comparison of labeled trees with valency trees*, J. Combin. Theory Ser. B, 11 (1971), pp. 105–119.
[28] B. Schieber and U. Vishkin, *On finding lowest common ancestors: Simplification and parallelization*, SIAM J. Comput., 17 (1988), pp. 1253–1262.
[29] M. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.
[30] D. Swofford and G. Olsen, *Phylogeny reconstruction*, in Molecular Systematics, D. M. Hillis et al., eds., Sinauer Associates, Sunderland, MA, 1990, pp. 411–501.
[31] N. Takahata, *Gene genealogy in three related populations: Consistency probability between gene and population trees*, Genetics, 122 (1989), pp. 957–966.
[32] M. Waterman and T. Smith, *On the similarity of dendrograms*, J. Theoret. Bio., 73 (1978), pp. 789–800.
[33] C.-I. Wu, *Inference of species phylogeny in relation to segregation of ancient polymorphisms*, Genetics, 127 (1991), pp. 429–435.
[34] L. Zhang, *On a Mirkin-Muchnik-Smith conjecture for comparing molecular phylogenies*, J. Comput. Bio., 4 (1997), pp. 177–188.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 753–808

THE GENERAL STRUCTURE OF EDGE-CONNECTIVITY OF A VERTEX SUBSET IN A GRAPH AND ITS INCREMENTAL MAINTENANCE. ODD CASE*

YEFIM DINITZ† AND ALEK VAINSHTEIN‡

Abstract. Let $G = (V, E)$ be an undirected graph, S be a subset of its vertices, $\mathfrak{C}_S$ be the set of minimum edge-cuts partitioning S, and λ_S be the cardinality of such a cut. We suggest a graph structure, called the *connectivity carcass* of S, that represents both cuts in $\mathfrak{C}_S$ and the partition of V by all these cuts; its size is $O(\min\{|E|, \lambda_S|V|\})$. In this paper we present general constructions and study in detail the case λ_S odd; the specifics of the case λ_S even are considered elsewhere. For an adequate description of the connectivity carcass we introduce a new type of graph: locally orientable graphs, which generalize digraphs. The connectivity carcass consists of a locally orientable quotient graph of G, a cactus tree (in case λ_S odd, just a tree) representing all distinct partitions of S by cuts in $\mathfrak{C}_S$, and a mapping connecting them. One can build it in $O(|S|)$ max-flow computations in G. For an arbitrary sequence of u edge insertions not changing λ_S, the connectivity carcass can be maintained in time $O(|V| \min\{|E|, \lambda_S|V|\} + u)$. For two vertices of G, queries asking whether they are separated by a cut in $\mathfrak{C}_S$ are answered in $O(1)$ worst-case time per query. Another possibility is to maintain the carcass in $O(|S| \min\{|E|, \lambda_S|V|\} + u)$ time, but to answer the queries in $O(1)$ time only if at least one of the vertices belongs to S.

Key words. edge-connectivity, minimum cuts, graph structures, incremental maintenance, dynamic algorithms

AMS subject classifications. 68R10, 68P05, 68Q25, 05C40

PII. S0097539797330045

1. Introduction.

Connectivity problems play an important role in applications of graph theory in computer science and have been extensively studied. In particular, much attention has been given to edge-connectivity problems. A pair of vertices s and t of an undirected graph $G = (V, E)$ is said to be k(-*edge*)-*connected* if there exist k edge-disjoint paths between them (equivalently, there is no k'-cut, $k' \leqslant k - 1$, separating s and t). The k-connectivity is an equivalence relation, and its equivalence classes are called k-*connectivity classes*. It is known that the system of globally minimum cuts and connectivity classes formed by them in a graph is represented by a *cactus tree*, i.e., a graph whose blocks are edges and cycles [DKL] (for a brief description in English see [NV, NGM]); it is just a tree if the size of globally minimum cuts is odd. On the other hand, the lattice of all (s, t)-minimum cuts for fixed vertices s and t is represented by all closed sets of some directed acyclic graph (dag) with one source and one sink [PQ].[1] Both these data structures can be maintained efficiently

*Received by the editors November 17, 1997; accepted for publication (in revised form) January 4, 2000; published electronically July 13, 2000. Parts of this work were published in a preliminary form in *Proceedings of the 26th ACM Symposium on Theory of Computing (STOC '94)* and in *Proceedings of the 6th SIAM-ACM Symposium on Discrete Algorithms (SODA '95)*.

http://www.siam.org/journals/sicomp/30-3/33004.html

†Department of Computer Science, Technion, Haifa, Israel 32000 (up to 1990: E. A. Dinic, Moscow). Present address: Department of Computer Science, Ben-Gurion University of the Negev, P.O. Box 653, Beer-Sheva, Israel 84105 (dinitz@cs.bgu.ac.il). The research of this author was supported in part by the Rashi Foundation and the Fund for the Promotion of Research at the Technion, Israel.

‡Department of Mathematics and Department of Computer Science, University of Haifa, Mount Carmel, Haifa, Israel 31905 (alek@mathcs2.haifa.ac.il). The research of this author was supported by the Rashi Foundation.

[1]This fact was discovered independently by A. Karzanov (personal communication, 1977).

under edge insertions (see [DW] for the first one and, e.g., [I] for the second). In [DV1] we have suggested a new structure, which is a natural generalization of both the aforementioned representations. This data structure represents all minimum cuts of G partitioning a fixed vertex subset and the corresponding partition of this subset into its connectivity classes; besides, it represents the partition of V by all these cuts. Our structure admits efficient incremental maintenance as well.

In this paper we describe general constructions and provide a detailed analysis of the odd case, that is, of the case when the cardinality of the cuts in question is odd. This allows us to introduce all the main ideas while avoiding certain technical difficulties. For a brief analysis of the general case, see [DV1, DV2]; a detailed analysis will be given elsewhere.

A well-known model for pairwise minimum cuts in a graph is the Gomory–Hu tree [GH]. It represents one minimum (x, y)-cut for any pair of vertices x, y, which is far from being sufficient for the purpose of incremental maintenance. Indeed, if the newly inserted edge increases the value of the single minimum (x, y)-cut represented by the Gomory–Hu tree, one cannot tell whether the size of minimum (x, y)-cuts remains the same or increases.

One of the goals of connectivity studies is to provide tools for fast answering connectivity queries in dynamic graphs, such as "Are vertices u and v k-connected?" or "Show k-cuts separating u and v." An important problem of this type is maintaining the hierarchy of the k-connectivity classes, $k \leqslant l$; the corresponding incremental algorithms (that allow insertions of edges to G) are presented in [WT] for $l = 2$, [GI] for $l = 3$, and [DW] for $l = 4$. We hope that our data structure can play the crucial role in solving this problem for arbitrary l. Indeed, take for a vertex subset a $(k-1)$-connectivity class, then its connectivity subclasses are just k-connectivity classes of G. The totality of these subclasses gives the partition of V into k-classes. Thus, it suffices to maintain our data structures for all $(k-1)$-classes of the graph, $k \leqslant l$. Observe that when an edge insertion causes merging of some $(k-1)$-classes, their new connectivity structures must be merged as well. The analysis of this merge is beyond the scope of the present paper; the case $l = 4$ is handled in [DW]. So, the results cover the "local" part of the work on the maintenance of the hierarchy of connectivity classes for a general l.

Another application of connectivity structures is augmentation problems of the type: given a graph, add the minimum number of edges so as to increase the connectivity of the whole graph to a prescribed value. The cactus structure of minimum cuts was employed essentially in [NGM, Be] to develop clear and fast schemes and algorithms for graph augmentation. One can suppose that the suggested generalization of this structure to locally minimum cuts will help to improve the results in this area.

Finally, the analysis of the connectivity structure of a vertex subset in a graph can arise itself in an application. For example, let us consider an interconnection network $\mathcal{N}$ whose nodes are terminals and auxiliary intermediate stations. Then it is natural to define the connectivity structure of $\mathcal{N}$ as the connectivity structure of the set of terminals in $\mathcal{N}$. As an illustration, consider the network with the terminals A, B, and C shown on Figure 1.1; the 3-cuts shown by dashed lines are the minimum cuts partitioning $\{A, B, C\}$, but there exist also two irrelevant 2-cuts and one irrelevant 3-cut.

Let us consider a vertex subset S and the set of all minimum S-cuts, that is, cuts of G that partition S and have the minimum cardinality, λ_S, among such cuts; λ_S is said to be the *connectivity* of S. Two such cuts are called equivalent if they partition

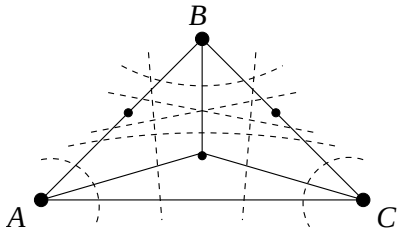


FIG. 1.1. *Minimum cuts in an interconnection network.*

S in the same way. We prove that the system of all such partitions of S, and thus also all the $(\lambda_S + 1)$-connectivity classes of S, is represented by a cactus tree $\mathcal{H}_S$ (called the skeleton), similarly to the representation of [DKL]. These partitions are defined by cuts of $\mathcal{H}_S$ almost bijectively. In the odd case the skeleton is just a tree, and the representation of the partitions by its cuts is a bijection.

Each family of equivalent minimum S-cuts is represented, similarly to [PQ], by cuts of a two-terminal dag of a special type, called a strip, which is defined up to the reversal of all its edges. Such a strip is obtained from G by certain contractions; in what follows we identify each vertex of the strip with the subset of V contracted into this vertex. We show that these strips agree on intersections in the following sense. First, if two nonterminal vertices of two strips intersect (as subsets of V), then these vertices coincide. Thus, the total partition of V by all minimum S-cuts subdivides only the two terminal vertices of each strip. We denote by $\mathcal{F}_S$ the graph obtained from G by contraction of each part of this total partition into a single vertex. Second, let two vertices of two strips coincide; then these vertices have, in a sense, the same set of incident edges. It turns out that the two 2-partitions of this set into incoming and outcoming edges in both strips coincide as unordered 2-partitions. As a result, at each vertex of $\mathcal{F}_S$ there arises a unique distinguished unordered 2-partition of the set of incident edges (called the inherent partition). The graph $\mathcal{F}_S$ with the inherent partitions at its vertices we call the flesh; it represents the entire family of minimum S-cuts. In this paper we extend to $\mathcal{F}_S$ methods used for dags in the case $|S| = 2$.

In order to build appropriate tools for analyzing $\mathcal{F}_S$, we introduce and study a special type of graphs: undirected graphs with a 2-partition of the set of incident edges at each vertex; we call them locally orientable graphs. In general such a graph cannot be oriented globally, but still preserves certain properties of digraphs. An analogue of an oriented path in a locally orientable graph is a path that agrees with inherent partitions (called a coherent path); we consider reachability along coherent paths. For graphs satisfying a certain natural restriction on cyclic coherent paths one can define the notions of reachability cones and strongly connected components, and adjust depth-first search (DFS) to scanning such graphs and finding these components. These techniques apply to the flesh since it satisfies even a stronger restriction: all its coherent paths are simple (thus, $\mathcal{F}_S$ is, in a sense, acyclic).

Finally, to each vertex of $\mathcal{F}_S$ we assign the set of families of equivalent minimum S-cuts in whose strips this vertex is nonterminal. The corresponding cuts of $\mathcal{H}_S$ define a subset of edges in $\mathcal{H}_S$ (called the projection π_S of the vertex); it turns out to be a path. The structure consisting of the skeleton, the flesh, and the projection mapping is called the *connectivity carcass of* S.

The connectivity carcass allows to answer readily several types of queries. The query asking whether two given vertices in S are $(k + 1)$-connected is answered just

by checking whether they are mapped into the same node of $\mathcal{H}_S$ or by checking whether they are contained in the same vertex of $\mathcal{F}_S$. In the latter way one can check also whether two arbitrary vertices of G are separated by a minimum S-cut. A minimum S-cut separating two vertices of which at least one belongs to S is defined immediately in terms of their projections. The same query in the general case may require, in addition, to combine such a cut with a reachability cone of one of the given vertices. For any two subsets S_1, S_2 of S, the corresponding strip of minimum S-cuts separating S_1 from S_2 is obtained by a certain contraction of the flesh defined by the projection mapping. The orientations of the edges in this strip are induced by the inherent partitions at the vertices of the flesh.

The connectivity carcass can be constructed by a polynomial algorithm. This is possible since it is "glued" naturally from (s,t)-strips of type [PQ], $s,t \in S$, s fixed (because of their "coincidences on intersections").

When considering incremental dynamics of the connectivity carcass, we assume that the cardinality of minimum S-cuts does not change under edge insertions; if such a change occurs, we say that the connectivity carcass vanishes. As a matter of fact, there arises a new system of larger minimum S-cuts, but this transition is beyond the scope of dynamic considerations in this paper.

Upon inserting a new edge into G, all the three components of the carcass change accordingly, and all the changes are of a contractive nature (provided λ_S does not change). The change of the graph of the flesh consists in contracting several vertices into a single new vertex. In particular, the endpoints U_1 and U_2 of the image of the new edge in $\mathcal{F}_S$, and all the vertices that can be visited by a coherent path between U_1 and U_2, "fall" into the new vertex. The other flesh vertices that are contracted are defined in terms of projections and reachability. The inherent partitions remain the same, except for the new flesh vertex, where the inherent partition is glued naturally from those of constituting vertices, or becomes trivial in case it was trivial for at least one of the constituting vertices. The change of the skeleton consists in contracting edges and "squeezing" cycles in the path-of-edges-and-cycles connecting the projections of U_1 and U_2 in the cactus tree $\mathcal{H}_S$; in the odd case it is just an ordinary path, and all of its edges are contracted. The projection of the new flesh vertex is $\pi_S(U_1) \cap \pi_S(U_2)$. The set of the other flesh vertices for which the projection changes is defined by their reachability from U_1 and U_2. From the projection of such a vertex, certain parts of the sets $\pi_S(U_1) \setminus \pi_S(U_2)$ or $\pi_S(U_2) \setminus \pi_S(U_1)$ are deleted.

The incremental algorithm [DV1] for maintaining the connectivity carcass is built according to the above scheme. In particular, it maintains the reachability cones of units, and thus involves, as a subproblem, incremental maintenance of transitive closures. The complexity of this algorithm is $O(|V| \min\{|E|, \lambda_S|V|\} + u)$, where u is the number of edge insertions (it is assumed that all of them preserve the connectivity of S); therefore, we do not exceed the complexity of best known algorithms for maintenance of the transitive closure (see [I, LL]). Each query asking whether two given vertices are separated by a minimum S-cut is answered in $O(1)$ worst-case time; such a cut itself can be shown in $O(|V|)$ amortized time. The dag representation of all minimum S-cuts separating any two subsets of S can be obtained in $O(|E|)$ amortized time. (For a comparison, in [GN] such a representation for an arbitrary pair of one-element subsets is constructed in $O(|V| \cdot |E|)$ time.)

The complexity of maintenance can be reduced at the expense of an increase in the reaction time for certain queries: we guarantee the same $O(1)$ worst-case time only in the cases when at least one of the vertices in question belongs to S. To

avoid the time-consuming maintenance of reachability cones, we suggest (see [DV2]) maintaining, instead of the actual flesh, a weaker contraction of the initial flesh, called the preflesh; the actual flesh can be obtained from the preflesh by contraction of its strongly connected components. The skeleton, the preflesh, and the projection mapping are maintained incrementally in time $O(|S|\min\{|E|,\lambda_S|V|\}+|V|\log|V|+u)$ for an arbitrary sequence of u edge insertions preserving the connectivity of S. The only increase of the reaction time, to $O(|E|)$ worst case, arises for the separation query in the case when the vertices in question both do not belong to S and have the same projection; the cut query in this situation is answered within the same $O(|E|)$ time.

As a corollary, we can simultaneously maintain the connectivity carcasses for all parts $S_1,\dots,S_r$ of an arbitrary partition of V in $O(|V|\min\{|E|,\lambda_S|V|\}+r|V|\log|V|+ru)$ time. Certain more delicate considerations extend this complexity bound (with the same overhead and even cheaper updates) to the "local" part of the work on the maintenance of the hierarchy of k-connectivity classes, $k\leqslant l$, mentioned above. The crucial observation is that when an edge (v,w) is added to G, the totality of connectivity carcasses for all k-connectivity classes, $k\leqslant l-1$, changes if and only if v and w belong to different l-classes. It turns out that the complexity of this local work is $O(|V|\min\{|E|,\lambda_S|V|\}+r|V|\log|V|+u)$, where r is the number of $(l-1)$-connectivity classes.

The first part of the paper treats the general case (λ_S arbitrary). In section 2 we give basic definitions and concepts. In section 3 we introduce locally orientable graphs and study their properties. In section 4 we consider a number of simple cases to gain intuition and to prove several basic lemmas. In section 5 we define the flesh (for the general case) and the skeleton (for the odd case only) and state their properties. In the rest of the paper λ_S is assumed to be odd. In section 6 we define the projection and present further properties of the connectivity carcass. In section 7 we describe the transformations of the connectivity carcass under the insertion of an edge to G. In section 8 we describe algorithms for construction and incremental maintenance of the connectivity carcass and for answering the queries.

The results of this paper were published in a preliminary form in [DV1, DV2].

2. Basic definitions and properties. In this paper we consider connected undirected graphs $G=(V,E)$, $|V|\geqslant 2$, without loops and possibly with parallel edges. We assume that these properties are preserved under vertex contractions; that is, we delete the obtained loops, but keep all the parallel edges. As usually, we denote $|V|$ by n and $|E|$ by m. For each ordered 2-partition $\mathcal{P}=(V_\mathcal{P},\bar{V}_\mathcal{P})$, $\bar{V}_\mathcal{P}=V\setminus V_\mathcal{P}$, of the vertex set V we define its edge-set $E_\mathcal{P}$ as the set of edges whose ends lie in distinct parts of $\mathcal{P}$. An ordered 2-partition $\mathcal{C}$ is said to be a *cut* of G if no proper subset of $E_\mathcal{C}$ coincides with $E_\mathcal{P}$ for some other 2-partition $\mathcal{P}$. The sets $V_\mathcal{C}$ and $\bar{V}_\mathcal{C}$ are called the *sides* of $\mathcal{C}$; we say that a vertex $v\in V$ lies *inside* $\mathcal{C}$ if $v\in V_\mathcal{C}$, and *outside* $\mathcal{C}$ if $v\in\bar{V}_\mathcal{C}$. For any cut $\mathcal{C}$, the opposite cut $\bar{\mathcal{C}}$ is defined by $V_{\bar{\mathcal{C}}}=\bar{V}_\mathcal{C}$, $\bar{V}_{\bar{\mathcal{C}}}=V_\mathcal{C}$; we say that $\bar{\mathcal{C}}$ is obtained from $\mathcal{C}$ by *flipping*. Let $\mathcal{C}$ and $\mathcal{C}'$ be two cuts such that $V_\mathcal{C}\subseteq V_{\mathcal{C}'}$; we say that $\mathcal{C}'$ *dominates* $\mathcal{C}$ and write $\mathcal{C}\preceq\mathcal{C}'$. If the inclusion $V_\mathcal{C}\subset V_{\mathcal{C}'}$ is strict, we write $\mathcal{C}\prec\mathcal{C}'$.

For any two cuts $\mathcal{C}$ and $\mathcal{C}'$, denote by $\mathcal{C}\cap\mathcal{C}'$ the 2-partition $(V_\mathcal{C}\cap V_{\mathcal{C}'},\bar{V}_\mathcal{C}\cup\bar{V}_{\mathcal{C}'})$, and by $\mathcal{C}\cup\mathcal{C}'$ the 2-partition $(V_\mathcal{C}\cup V_{\mathcal{C}'},\bar{V}_\mathcal{C}\cap\bar{V}_{\mathcal{C}'})$. Observe that the 2-partitions $\mathcal{C}\cap\mathcal{C}'$ and $\mathcal{C}\cup\mathcal{C}'$ are not necessarily cuts; however, if they are, then $\mathcal{C}\cap\mathcal{C}'\preceq\mathcal{C}\preceq\mathcal{C}\cup\mathcal{C}'$ and $\mathcal{C}\cap\mathcal{C}'\preceq\mathcal{C}'\preceq\mathcal{C}\cup\mathcal{C}'$. As usually, the *cardinality* $c(\mathcal{C})=c(V_\mathcal{C},\bar{V}_\mathcal{C})$ of a cut $\mathcal{C}$ is defined to be the size of its edge-set $E_\mathcal{C}$; clearly, $c(\bar{\mathcal{C}})=c(\mathcal{C})$. The notion of cardinality is extended naturally to arbitrary 2-partitions of V and, moreover, to arbitrary pairs of

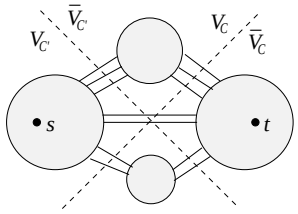


FIG. 2.1. *Two intersecting S-mincuts.*

disjoint subsets of V: if $X_1, X_2 \subset V$, $X_1 \cap X_2 = \varnothing$, then $c(X_1, X_2)$ stands for the number of edges with one end in X_1 and the other in X_2.

Let S be a subset of V; we denote by σ the cardinality of S. Any cut $\mathcal{C}$ defines a 2-partition $(S_\mathcal{C}, \bar{S}_\mathcal{C})$ of S, with $S_\mathcal{C} = V_\mathcal{C} \cap S$, $\bar{S}_\mathcal{C} = \bar{V}_\mathcal{C} \cap S$. A cut $\mathcal{C}$ is said to be an *S-cut* if both $S_\mathcal{C}$ and $\bar{S}_\mathcal{C}$ are nonvoid. For $S = \{a, b\}$, we speak of (a, b)-cuts instead of $\{a, b\}$-cuts to keep the usual notation. It is easy to see that the minimum cardinality among 2-partitions dividing S is equal to that among S-cuts; we denote it by λ_S. Moreover, each 2-partition of minimum cardinality λ_S dividing S is an S-cut. In this paper we are interested only in S-cuts of cardinality λ_S; we call them minimum S-cuts, or *S-mincuts*. The following basic property of S-mincuts (see Figure 2.1) is used extensively throughout the paper.

LEMMA 2.1. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two arbitrary S-mincuts such that both $V_\mathcal{C} \cap V_{\mathcal{C}'} \cap S$ and $\bar{V}_\mathcal{C} \cap \bar{V}_{\mathcal{C}'} \cap S$ are nonempty. Then*

(i) $c(V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'}, \bar{V}_\mathcal{C} \cap V_{\mathcal{C}'}) = 0$;

(ii) $c(V_\mathcal{C} \cap V_{\mathcal{C}'}, V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'}) = c(V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'}, \bar{V}_\mathcal{C} \cap \bar{V}_{\mathcal{C}'})$, $c(V_\mathcal{C} \cap V_{\mathcal{C}'}, \bar{V}_\mathcal{C} \cap V_{\mathcal{C}'}) = c(\bar{V}_\mathcal{C} \cap V_{\mathcal{C}'}, \bar{V}_\mathcal{C} \cap \bar{V}_{\mathcal{C}'})$.

Proof. (i) We assume that $\mathcal{C} \neq \mathcal{C}'$, since otherwise the assertion is trivial. Let s and t be two arbitrary vertices in $V_\mathcal{C} \cap V_{\mathcal{C}'} \cap S$ and $\bar{V}_\mathcal{C} \cap \bar{V}_{\mathcal{C}'} \cap S$, respectively. Then both $\mathcal{C}$ and $\mathcal{C}'$ are (s, t)-mincuts, and hence, by [FF, Chap. 1, Corollary 5.3], every edge e between $V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'}$ and $V_{\mathcal{C}'} \cap \bar{V}_\mathcal{C}$ must be both free of any maximal (s, t)-flow and saturated by it.

(ii) By (i), for any maximal (s, t)-flow $c(V_\mathcal{C} \cap V_{\mathcal{C}'}, V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'})$ is the flow entering $V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'}$, and $c(V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'}, \bar{V}_\mathcal{C} \cap \bar{V}_{\mathcal{C}'})$ is the flow leaving $V_\mathcal{C} \cap \bar{V}_{\mathcal{C}'} X \cap \bar{Y}$. These two quantities are equal by the flow conservation law. The second equality is proved in the same way. □

The following observation, though trivial, is important: for any two S-mincuts $\mathcal{C}_1$ and $\mathcal{C}_2$, at least one of the pairs $\mathcal{C}_1, \mathcal{C}_2$ and $\mathcal{C}_1, \bar{\mathcal{C}}_2$ satisfies the conditions of Lemma 2.1.

Two S-cuts are said to be S-equivalent if they define the same 2-partition of S. Equivalence classes of S-mincuts are called *bunches*. Evidently, if one replaces each cut of a bunch by the opposite cut, then one again gets a bunch, which is said to be opposite to the initial one. The following statement is an immediate corollary of Lemma 2.1.

FACT 2.2. *Let $\mathcal{C}$ and $\mathcal{C}'$ be two S-mincuts satisfying the conditions of Lemma* 2.1. *Then*

(i) *both $\mathcal{C} \cap \mathcal{C}'$ and $\mathcal{C} \cup \mathcal{C}'$ are S-mincuts;*

(ii) *let $\mathcal{C}''$ be an S-mincut S-equivalent to $\mathcal{C}'$; then $\mathcal{C} \cap \mathcal{C}''$ and $\mathcal{C} \cup \mathcal{C}''$ are S-mincuts S-equivalent to $\mathcal{C} \cap \mathcal{C}'$ and $\mathcal{C} \cup \mathcal{C}'$, respectively.*

In particular, the intersection and the union of two S-equivalent S-mincuts are cuts from the same bunch as well.

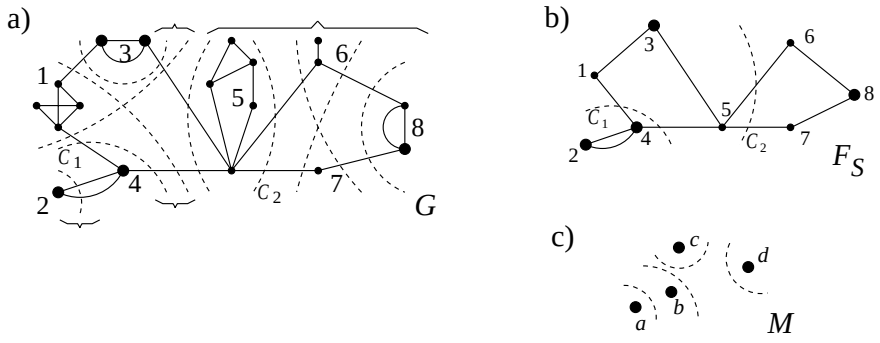


FIG. 2.2. *S-mincuts, the graph $\mathcal{F}_S$, and inducing of cuts and bunches.*

Let S_1 and S_2 be two disjoint subsets of S. An S-mincut is said to be an (S_1, S_2)-*mincut* if it keeps S_1 and S_2 on different sides. It follows immediately from Fact 2.2(i) that the intersection of all (S_1, S_2)-mincuts having S_1 inside is again an (S_1, S_2)-mincut. We call it the S_1-*tight* (S_1, S_2)-mincut.

Two vertices of G are said to be S-equivalent if they are not separated by any S-mincut. Equivalence classes of vertices are called S-units, or simply *units*. Evidently, the unit containing a given vertex is just the inner side of the intersection of all S-mincuts having this vertex inside. We denote by $\mathcal{F}_S$ the quotient graph obtained from G by contracting all the vertices of the same unit. The vertices of $\mathcal{F}_S$ are naturally called units. Each S-mincut induces naturally a cut of $\mathcal{F}_S$ separating the same units; in what follows we do not distinguish between these two cuts.

Observe that, in general, S is partitioned by the set of all S-mincuts into subsets $S_1, S_2, \ldots, S_r$, and r can be less than σ. This situation is handled in the following lemma.

LEMMA 2.3. (i) *Contract each S_i, $1 \leqslant i \leqslant r$, to a supervertex and denote the set of supervertices by $\bar{S}$. Then $\bar{S}$-mincuts and S-mincuts coincide up to the above contraction*; *in particular*, $\mathcal{F}_S = \mathcal{F}_{S'}$.

(ii) *Fix an arbitrary vertex v_i in each S_i, $1 \leqslant i \leqslant r$, and denote $S' = \{v_1, \ldots, v_r\}$. Then the sets of S'-mincuts and S-mincuts coincide*; *in particular*, $\mathcal{F}_S = \mathcal{F}_{S'}$.

Proof. (i) Follows immediately from definitions.

(ii) Indeed, since $S' \subseteq S$, each S'-cut is an S-cut, and hence either $\lambda_{S'} > \lambda_S$, or $\lambda_{S'} = \lambda_S$ and each S'-mincut is an S-mincut. On the other hand, any S-mincut separates some pair (S_i, S_j) and thus separates v_i from v_j. Therefore it is also an S'-cut, and moreover an S'-mincut, as required. □

Let $f : V \to M$ be a mapping to an arbitrary set M. For any $W \subseteq V$, we denote by $f(W)$ the set $\{f(w) : w \in W\}$. We say that a 2-partition $\mathcal{P}$ of M *f-induces a cut* $\mathcal{C}$ if $V_{\mathcal{C}} = f^{-1}(M_{\mathcal{P}})$. Similarly, given a mapping $g : S \to M$, we say that $\mathcal{P}$ *g-induces a bunch* of S-cuts if $S_{\mathcal{C}} = g^{-1}(M_{\mathcal{P}})$ for any cut $\mathcal{C}$ in this bunch.

To illustrate the above notions, let us consider the graph G presented on Figure 2.2(a). Large black circles denote vertices in S; thus $\lambda_S = 2$. All the ten S-mincuts are shown by dashed lines. They fall into four bunches consisting of 1, 2, 2, and 5 cuts, respectively, marked by braces. The vertices of G fall into eight units marked by numbers. The quotient graph $\mathcal{F}_S$ is presented on Figure 2.2(b). The quotient mapping f takes all the vertices of a unit i to the vertex i of $\mathcal{F}_S$. Each S-mincut is f-induced by a unique cut of $\mathcal{F}_S$ (e.g., see cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ in Figures 2.2(a) and (b)).

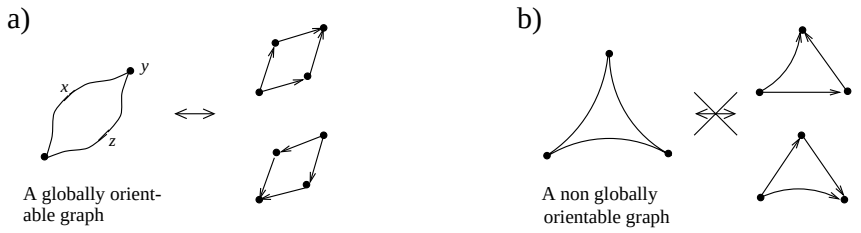


FIG. 3.1. *Local and global orientability.*

Let us consider also a four-element set $M = \{a, b, c, d\}$ and the mapping $g : S \to M$ that takes the vertices of S belonging to the units 2, 3, 4, 8 to a, c, b, d, respectively. The four 2-partitions of M shown in Figure 2.2(c) g-induce the four bunches of S-mincuts in G. Observe that all the above constructions are valid also for the set S' obtained by deleting from S any of the two vertices in unit 3.

3. Locally orientable graphs.

3.1. General locally orientable graphs. Recall that in the case $|S| = 2$ the quotient graph $\mathcal{F}_S$ possesses the structure of a dag, up to the reversal of all its edges. In general, $\mathcal{F}_S$ has a more sophisticated structure; however, as is shown in this paper, one can think of $\mathcal{F}_S$ as being glued from simpler dag-like objects of the above type. The basic local property of $\mathcal{F}_S$ is that the edges incident to any vertex are subdivided in a unique way into two groups. This subdivision is inherited from the subdivision into the edges leaving and entering the same vertex in the corresponding dags.

In this section we introduce and study general graphs equipped with this additional local structure, which are in a sense intermediate between undirected and directed graphs. We further impose on these graphs additional restrictions and arrive at certain important properties similar to those known for directed graphs. These properties support the detailed analysis of $\mathcal{F}_S$ carried on in subsequent sections.

Let v be an arbitrary vertex of an undirected graph. The *star* of v is the set E_v of edges incident to v. Assume that a certain unordered 2-partition of E_v is fixed; the parts of this 2-partition are called the two *sides* of the star. If both sides of E_v are nontrivial (nonempty), then v is called a *stretched* vertex; otherwise (when one of the sides coincides with E_v and the other is empty) v is called a *terminal*. A graph is said to be *locally orientable* if a 2-partition of the star into two sides is fixed at each of its vertices. In other words, a locally orientable graph is a pair $(G; \mathcal{E})$, where $G = (V, E)$ is an undirected graph and $\mathcal{E}$ is a mapping that assigns to each vertex $v \in V$ an unordered 2-partition of E_v. We say that $(G; \mathcal{E})$ *overlies* the initial graph G, and that G *underlies* the locally orientable graph $(G; \mathcal{E})$.

In particular, for any vertex v of an arbitrary directed graph, the partition of arcs incident to v into incoming and outcoming defines a 2-partition of E_v in the underlying undirected graph. Thus, for any digraph D, there exists a canonically defined locally orientable graph, which overlies the underlying undirected graph of D. A locally orientable graph $(G; \mathcal{E})$ is called *globally orientable* if one can direct the edges of G in such a way that for any vertex v all the edges entering v form one side of E_v defined by $\mathcal{E}$, while all the edges leaving v form its other side. Figure 3.1 provides two examples of locally orientable graphs of which one is globally orientable, while the other is not. Here and in what follows stretched vertices are denoted by rectangles to visualize the corresponding 2-partitions; terminals are denoted by circles. Globally

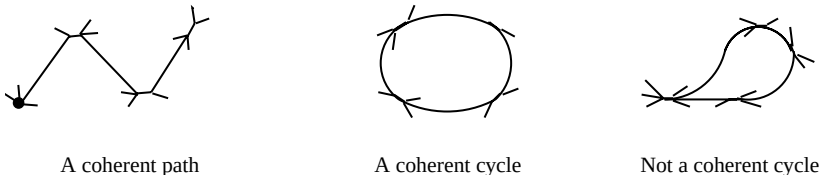


FIG. 3.2. *Coherence of paths and cycles.*

orientable graphs are essentially equivalent to directed graphs in the following sense.

LEMMA 3.1. *Given a connected globally orientable graph, the overlying directed graph is restored uniquely up to the global reversal of arcs.*

Proof. Let $\vec{G}_1$ and $\vec{G}_2$ be two digraphs overlying a given locally orientable graph G. Let V^+ be the subset of vertices at which the ordered 2-partitions of the star into incoming and outcoming arcs in $\vec{G}_1$ and $\vec{G}_2$ coincide, and let V^- be the subset of vertices at which they are opposite. Evidently, (V^+, V^-) is a partition of V. Assume that both its parts are nonempty. Since G is connected, it contains an edge between vertices $v^+ \in V^+$ and $v^- \in V^-$. Assume, without loss of generality (w.l.o.g.), that the corresponding arc in $\vec{G}_1$ leaves v^+ and enters v^-. Then the corresponding arc in $\vec{G}_2$ must leave both of them, a contradiction. Thus either $V^- = \varnothing$ and $\vec{G}_1 \cong \vec{G}_2$, or $V^+ = \varnothing$ and $\vec{G}_2$ is opposite to $\vec{G}_1$. □

The following construction provides a criterion for the global orientability of a locally orientable graph G. (We do not use this criterion in what follows.) Let us split each vertex v of G and its star into the two vertices v_1 and v_2 whose stars are exactly the sides at v and add the auxiliary edge (v_1, v_2); the obtained graph we denote G'.

PROPOSITION 3.2. *A locally orientable graph G is globally orientable if and only if G' is bipartite.*

Proof. Let G be globally orientable and $\vec{G}$ be its overlying digraph. Let us transform $\vec{G}$ into $\vec{G}'$ similarly to the above transformation; evidently, $\vec{G}'$ overlies G'. The partition $(\{v_1 : v \in V\}, \{v_2 : v \in V\})$ shows that G' is bipartite.

Now let G' be bipartite with the partition (V^*, V^{**}). Let us direct all edges from V^* to V^{**}. The contraction of all auxiliary edges results in a digraph overlying G. It is consistent with the 2-partitions at vertices since for each $v \in V$ the corresponding vertices v_1 and v_2 belong to different parts, i.e., arcs only leave v_1 and only enter v_2, or vice versa. □

According to Lemma 3.1, locally orientable graphs are a generalization of directed graphs. Some concepts of digraphs have their natural analogues for this generalization. A path $(v_0, e_1, v_1, e_2, \ldots, e_r, v_r)$ in a locally orientable graph is said to be *coherent* if for any of its inner vertices v_i, $1 \leqslant i \leqslant r-1$, the edges e_i and e_{i+1} belong to the opposite sides of the star of v_i (see Figure 3.2). Clearly, the inverses of any coherent path, any single edge path, and any single vertex path are coherent paths. Observe that each coherent path can be extended beyond its endpoint, provided it is not a terminal. Coherent paths are natural analogues of directed paths in digraphs. In particular, a coherent path in a globally orientable graph corresponds to a directed path in the overlying digraph or to the inverse of such a path.

For an arbitrary locally orientable graph, all vertices and edges of coherent paths beginning at a vertex v are called *reachable* from v. All these vertices and edges form

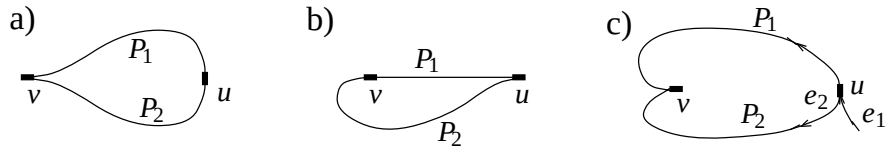


FIG. 3.3. *Cases in the proof of Lemma* 3.4.

the *reachability subgraph* of v. Observe that the reachability relation on vertices is symmetric (though not transitive: see vertices x, y, z in Figure 3.1(a)). Therefore, the notion of mutual reachability of two vertices is well defined.

We call a 2-partition $\mathcal{P}$ of the vertex set of a locally orientable graph (in particular, its cut) *transversal* if any coherent path intersects the corresponding edge-set $E_{\mathcal{P}}$ at most by one edge. Thus, if both endpoints of a coherent path lie on the same side of a transversal cut, then all of the path lies on the same side of this cut. Observe that the edge-set of a transversal 2-partition intersects at most one side of the star of any vertex. Moreover, the following statement can be proved easily.

FACT 3.3. *Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two transversal 2-partitions of the vertex set of a locally orientable graph such that $V_{\mathcal{P}_1} \cap V_{\mathcal{P}_2}$ is a single stretched vertex v. Then the intersections of the star of v with the edge-sets of $\mathcal{P}_1$ and $\mathcal{P}_2$ are exactly the two sides of this star.*

3.2. Coherent locally orientable graphs. A cyclic coherent path (for which $v_0 = v_r$ and $r > 1$) is called a *coherent cycle* if the edges e_r and e_1 belong to the opposite sides at v_0 (see Figure 3.2). A locally orientable graph is called *coherent* if each its cyclic coherent path is a coherent cycle. Coherent locally orientable graphs possess the following remarkable property.

LEMMA 3.4. *The reachability subgraph of any vertex of a coherent locally orientable graph is globally orientable.*

Proof. To construct a global orientation of the reachability subgraph of v we proceed as follows. Let us choose a side of the star of v. Moving along any coherent path starting at v and leaving v from this side, we direct all its edges from v. Similarly, moving along any coherent path starting at v and leaving v from the opposite side, we direct all its edges to v.

Let us prove the following statement: if two paths P_1 and P_2 from v to an arbitrary vertex u leave v from the same side (from the opposite sides), then they enter u at the same side (at the opposite sides). Assume to the contrary that P_1 and P_2 leave v from the same side and enter u at the opposite sides (see Figure 3.3(a)). Then the cyclic coherent path composed of P_1 and the inverse of P_2 is not a coherent cycle. Similarly, if P_1 and P_2 leave v from the opposite sides and enter u at the same side, then the cyclic coherent path composed of the inverse of P_2 and P_1 is not a coherent cycle (see Figure 3.3(b)).

Assume now that in our construction some edge e acquires two opposite orientations via paths P_1 and P_2. Evidently, the behavior of these paths at an arbitrary endpoint of e distinct from v contradicts to the above statement. Thus, the orientation of edges is well defined. Finally, to prove its consistency with the initial orientability structure we have to ensure that if two edges e_1, e_2 incident to a vertex u are oriented to and from u, respectively, then they are at the opposite sides of u. For $u = v$ it is evident. For $u \neq v$, we have to consider four cases depending on the types of the paths P_1 and P_2 that defined the orientations of the edges e_1, e_2, respectively. For

example, let both P_1 and P_2 leave v from the side opposite to the one chosen in the construction and let e_1 and e_2 be oriented as in Figure 3.3(c). Then P_2 enters u via e_2 while P_1 enters u via an edge preceding e_1 in P_1. Thus P_2 and P_1 enter u at the opposite sides, a contradiction with the above statement. The three other cases are handled in the same way. □

According to Lemmas 3.1 and 3.4, one can construct in an obvious way a version of the DFS that starts from an arbitrary vertex v and scans all the vertices and edges reachable from v in the same direction (i.e., by paths leaving v from the same side).

For a coherent locally orientable graph, the relation "two vertices belong to the same coherent cycle" is symmetric and transitive; the subgraphs induced by its equivalence classes are naturally called strongly connected components. It is easy to see that strongly connected components intersecting a reachability subgraph lie entirely in this subgraph and correspond bijectively to the strongly connected components of its overlying digraph. Hence, we can execute any DFS-based linear algorithm for finding strongly connected components on any reachability subgraph.

An analogue of a dag is an *acyclic* locally orientable graph, the one without cyclic coherent paths; thus, it is coherent. In particular, the underlying graph of a dag is acyclic in the above sense. As in the proof of Lemma 3.4, it follows from the acyclicity that for any two coherent paths joining v_1 and v_2, their initial edges leave v_1 from the same side. Thus, the set of vertices reachable from v_1 splits into two *reachability cones* corresponding to the sides of its star. (If v_1 is a terminal, one of the cones is trivial.) It is easy to see that if v_2 belongs to a reachability cone $\mathcal{R}$ of v_1, then one of the reachability cones of v_2 contains v_1 while the other lies strictly inside $\mathcal{R}$. (This property may be regarded as a weak analogue of transitivity.) Acyclicity implies that each coherent path can be extended beyond each of its endpoints up to a terminal. As a corollary, each nontrivial reachability cone of a vertex contains at least one terminal distinct from this vertex.

3.3. Strips. A locally orientable graph is called *balanced* if for any stretched vertex the two sides of its star have equal cardinalities. In this subsection we study balanced acyclic two-terminal locally orientable graphs. It is shown that such graphs are essentially identical to dags used in the analysis of (s,t)-mincuts. In fact, the quotient graph $\mathcal{F}_S$ in the general case has all the above properties except for one: it has more than two terminals (see section 5). To study this sophisticated object we use the results presented in this subsection.

LEMMA 3.5. *Any acyclic two-terminal locally orientable graph is globally orientable.*

Proof. By Lemma 3.4, it is enough to prove that the reachability subgraph of a terminal in such a graph coincides with the entire graph. Each vertex or edge is a coherent path; therefore, it can be extended to a coherent path between two terminals. By the acyclicity, these terminals are distinct, and the result follows. □

According to this Lemma, properties of an acyclic two-terminal locally orientable graph G are similar to those of its overlying dag $\vec{G}$. In particular, the following statements concerning transversal cuts are true.

FACT 3.6. (i) *The transversal cuts of G underlie the cuts of $\vec{G}$ in which all arcs are directed from the part containing the source to the part containing the sink.*

(ii) *Any transversal 2-partition of G is a (transversal) cut separating the terminals.*

(iii) *A reachability cone of any vertex in G defines a transversal cut; it is the intersection of all transversal cuts such that both this vertex and the terminal that*

belongs to this cone lie inside them.

An acyclic two-terminal globally orientable graph is called a *strip* (of width w) if all its transversal cuts are of the same cardinality w. Observe that the degree of a terminal in a strip equals its width (since the 2-partition isolating this terminal is a transversal cut).

Let s and t be the terminals and v be a vertex of a strip. We label the sides of the 2-partition and the cones at v by s and t; the cones are denoted $\mathcal{R}_s(v)$ and $\mathcal{R}_t(v)$. Thus, any two coherent paths $(u_1, \dots, u_2) \in \mathcal{R}_x(u_1)$ and $(u_2, \dots, u_3) \in \mathcal{R}_x(u_2)$, $x = s$ or t, can be concatenated into a coherent path $(u_1, \dots, u_2, \dots, u_3) \in \mathcal{R}_x(u_1)$.

Strip Lemma. (i) *An acyclic two-terminal locally orientable graph is a strip if and only if it is balanced.*

(ii) *There exists at most one strip for a fixed underlying graph and a fixed pair of terminals.*

Proof. (i) Let G be a balanced acyclic locally orientable graph with two terminals s and t. By Lemma 3.5, G is globally orientable. Let $\vec{G}$ be the overlying digraph of G in which s is the source and let $\mathcal{C}$ be a transversal cut of G such that $s \in V_{\mathcal{C}}$. We consider the difference Δ between the sum of outdegrees (in $\vec{G}$) for all vertices in $V_{\mathcal{C}}$ and the sum of their indegrees. On one hand, since G is balanced and $t \notin V_{\mathcal{C}}$, Δ equals the outdegree of s. On the other hand, since each internal arc of $V_{\mathcal{C}}$ contributes 1 to both sums and all external arcs are directed outside (by Fact 3.6(i)), Δ equals the cardinality of $E_{\mathcal{C}}$. Thus $c(\mathcal{C})$ equals the outdegree of s for any transversal cut $\mathcal{C}$.

Let now G be a strip and $\vec{G}$ be its overlying dag with the source s and the sink t. Let v be an arbitrary nonterminal vertex and let $\mathcal{R}$ be the reachability cone of v in $\vec{G}$. Both 2-partitions $(\mathcal{R}, V \setminus \mathcal{R})$ and $(\mathcal{R} \setminus v, (V \setminus \mathcal{R}) \cup v)$ are transversal since any directed path can only enter $\mathcal{R}$ (or $\mathcal{R} \setminus v$) but not leave it. By Fact 3.6(ii), they are cuts. Their cardinalities differ by the difference between the indegree and the outdegree of v. Since both cardinalities are equal, the indegree and outdegree at v are equal as well.

(ii) The statement is evident for graphs with two vertices; hence, we assume that there exists a vertex $v \neq s, t$. Let us consider a strip G with a source s and the sink t. It is easy to see that the deletion of s from G yields new terminals. Indeed, the one-vertex coherent path v can be extended up to terminals in both directions; these terminals are distinct since the graph is acyclic. As it was shown in the proof of Lemma 3.4, all the deleted edges incident to an arbitrary vertex belong to the same side of the vertex. Hence, for any new terminal exactly half of incident edges are deleted, and for any other vertex strictly less than half (a characterization in terms of the underlying graph). Let us prove that the contraction of the new terminals implies a strip. The obtained graph is evidently balanced and two-terminal. Assume that there exists a cyclic coherent path in it. Evidently, it starts and ends at the new terminal and thus can be extended to a cyclic coherent path starting and ending at s in the initial strip, a contradiction. Finally, the initial strip can be restored uniquely from the strip obtained.

Assume to the contrary that assertion (ii) is not valid. Then there exists a minimal counterexample: a graph with the minimal number of vertices giving rise to two distinct strips. Let us execute with these strips the above procedure. According to the characterization mentioned, the resulting undirected graphs coincide. Thus, by the minimality of the counterexample, the resulting strips coincide as well. By the remark at the end of the previous paragraph, the same holds for the initial strips, a contradiction. □

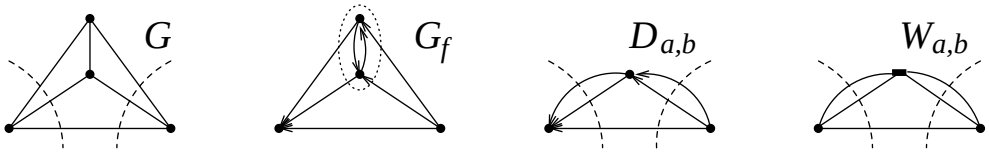


FIG. 3.4. *Construction of the* (a, b)*-strip.*

The concept of a strip allows us to reformulate and refine the main result of [PQ] on the representation of all minimum cuts between two vertices of a graph. Let us consider an undirected graph G and an arbitrary pair of its vertices a, b. Assume f is a maximal flow from a to b in G, and G_f is the corresponding residual graph. The result of the contraction of each strongly connected component of G_f to a new supervertex is a dag $D_{a,b}$ not depending on f. The corresponding quotient mapping is denoted by $\delta_{a,b}$, and the underlying locally orientable graph of this dag by $\mathcal{W}_{a,b}$ (see Figure 3.4).

THEOREM 3.7. *Let G be an undirected graph and a, b be a pair of its vertices. Then the graph $\mathcal{W}_{a,b}$ overlies $\mathcal{F}_{\{a,b\}}$, is a strip, and its transversal cuts correspond bijectively to the minimum (a, b)-cuts of G via the $\delta_{a,b}$-inducing.*

Proof. Recall that a vertex subset X in an arbitrary dag is said to be closed if no other vertices can be reached from this subset, or, equivalently, if no arc goes from X to $\bar{X}$. By [PQ] the $\delta_{a,b}$-inducing provides a bijection between the minimum (a, b)-cuts of G and the 2-partitions of type {a closed set of $D_{a,b}$, its complement}. According to Fact 3.6(i), the 2-partitions of the above type overlie exactly the transversal cuts of $\mathcal{W}_{a,b}$. Since all minimum (a, b)-cuts of G have the same cardinality, so do all the transversal cuts of $\mathcal{W}_{a,b}$, and hence $\mathcal{W}_{a,b}$ is a strip. It follows from the above discussion that the partition of V into the vertices of $\mathcal{W}_{a,b}$ is a refinement of the partition into the vertices of $\mathcal{F}_{\{a,b\}}$. To prove that they, in fact, coincide, it is sufficient to show that each vertex U of $\mathcal{W}_{a,b}$ is separated from all other vertices by transversal cuts. Indeed, by Fact 3.6(iii), these cuts are just the two cuts defined by reachability cones of U. □

Remark. Observe that natural analogues of the above construction and of Theorem 3.7 are valid for directed graphs.

The strip $\mathcal{W}_{a,b}$ will be referred to as the (a, b)-strip, and its vertices as (a, b)-units, or units of $\mathcal{W}_{a,b}$. Obviously, $\mathcal{W}_{b,a}$ coincides with $\mathcal{W}_{a,b}$ and $\delta_{b,a} = \delta_{a,b}$. The above constructions and results can be extended easily to the case of minimum cuts separating two disjoint vertex subsets A and B: it is enough to contract A and B into new supervertices. The corresponding (A, B)-strip is denoted by $\mathcal{W}_{A,B}$ and the quotient mapping by $\delta_{A,B}$.

LEMMA 3.8. *Let u be a vertex of G and $U \neq \delta_{a,b}(b)$ be the corresponding (a, b)-unit. The contraction of the reachability cone of U in $\mathcal{W}_{a,b}$ containing the terminal $\delta_{a,b}(a)$ gives the $(\{a, u\}, b)$-strip; its width is equal to the width of the initial strip $\mathcal{W}_{a,b}$.*

Proof (see Figure 3.5). It is easy to see that the contraction of a side of a transversal cut in an acyclic locally orientable graph preserves acyclicity. Therefore, by Fact 3.6(iii) and the Strip Lemma, the above contraction yields a strip $\mathcal{W}$. Since the terminal $\delta_{a,b}(b)$ remains untouched, the width of $\mathcal{W}$ equals that of $\mathcal{W}_{a,b}$.

By part (ii) of the Strip Lemma, it is enough to prove now that the underlying graph of $\mathcal{W}$ coincides with that of $\mathcal{W}_{\{a,u\},b}$. Since all $(\{a, u\}, b)$-cuts are (a, b)-cuts,

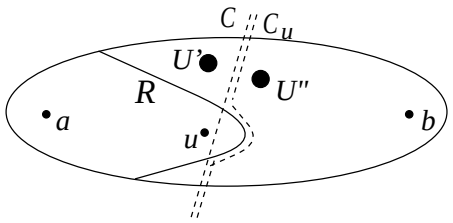


FIG. 3.5. *To the proof of Lemma* 3.8.

we can study them (more exactly, their images) in $\mathcal{W}_{a,b}$. First, by Fact 3.6(iii), the reachability cone $\mathcal{R}$ in question coincides with the $(\{a,u\},b)$-unit containing $\{a,b\}$. Second, let U' and U'' be two (a,b)-units not belonging to $\mathcal{R}$ and let $\mathcal{C}$ be an (a,b)-cut separating them. Evidently, the union of $\mathcal{C}$ and the cut defined by $\mathcal{R}$ is an $(\{a,u\},b)$-cut $\mathcal{C}_u$ separating U' and U''. □

Observe that for an arbitrary vertex subset S, the set of all minimum S-cuts is the union of the sets of minimum (a,b)-cuts for certain pairs of vertices $a,b \in S$. It is natural to ask whether it is possible to glue the corresponding (a,b)-strips together to form a single object. The main finding of this paper is that such an object exists: it is $\mathcal{F}_S$ with a canonical structure of local orientability defined by these (a,b)-strips.

4. Simple cases.

4.1. Generalities. To make our ideas more intuitive, we consider in this section several simple cases. The word "case" refers to the structure of partitions of S by S-mincuts; an example of such a structure is shown in Figure 2.2(c). From this point of view, the simplest possible cases arise when S is divided (by all S-mincuts together) into two or three parts.

Assume that S is divided into two parts S_1 and S_2. By Lemma 2.3(i), we can consider these parts as supervertices a and b and thus reduce this case to the situation when $S=\{a,b\}$. It is convenient to represent this situation by the "skeleton" graph shown in Figure 4.1(a). In a similar way, the case when S is divided into three parts is reduced to the situation when $S=\{a,b,c\}$ and any one of these vertices is separated from any other one by an S-mincut. Clearly, from the three possible types of S-mincuts, $(a,\{b,c\})$, $(b,\{a,c\})$, and $(c,\{a,b\})$, at least two must exist. So we have two subcases: the asymmetric, when exactly two types of S-mincuts are present, and the symmetric, when there are present all three of them. As before, we represent these subcases by certain "skeleton" structures. The asymmetric subcase is represented by the path (a,b,c) (provided we assume that there are no $(b,\{a,c\})$-cuts), and the symmetric subcase by the 3-star with terminals a, b, c (see Figures 4.1(b) and (c)). Observe that the 1-cuts of all these skeleton structures (they are shown by dashed lines in Figure 4.1) correspond bijectively to distinct partitions of S by minimum S-cuts in the original graph.

In addition, in the last subsection we consider the case when S is divided into an arbitrary number of parts, but the corresponding skeleton structure is just a path.

The definitions and results of this section can be regarded as prototypes of definitions and results for the general case presented in sections 5–7. Sometimes the proofs are omitted; this means that a more general result is proved in a subsequent section.

4.2. A two-element subset *S*. From the static point of view, the case $\sigma=2$ is covered completely by Theorem 3.7; recall that for $S=\{a,b\}$ one has $\mathcal{F}_S=\mathcal{W}_{a,b}$.

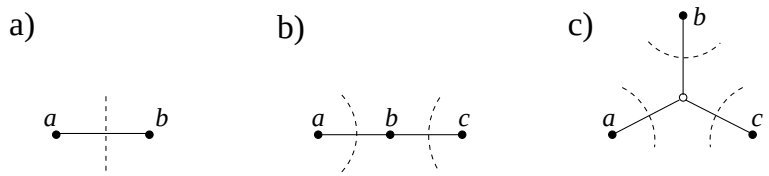


FIG. 4.1. *Skeleton structures.*

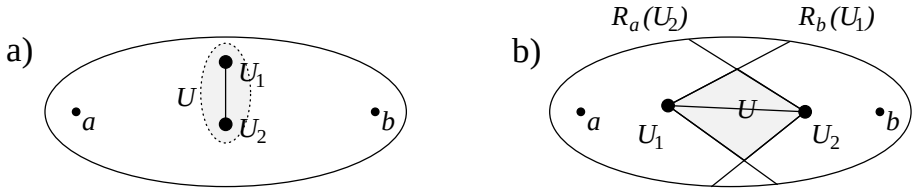


FIG. 4.2. *Dynamics of* $\mathcal{F}_{\{a,b\}}$.

Let us turn to dynamics. In what follows, we put a hat over any notation (e.g., $\widehat{\mathcal{R}}_c$, or $\widehat{\mathcal{F}}_S$) to denote the corresponding object after edge insertion. Recall that the units of $\mathcal{F}_S$ are defined as the strongly connected components of the graph G_f, where f is a maximum flow from a to b in G. Since we restrict ourselves to the case of a fixed value of (a,b)-mincuts, we may use the *same* flow f for the entire dynamic process. Hence, we can maintain the graph G_f fixing local changes. Namely, the insertion of a new edge (u_1,u_2) implies the addition of two arcs (u_1,u_2) and (u_2,u_1) to G_f. Moreover, one can maintain the dag $D_{a,b}$ of the strongly connected components of G_f as follows. Let U_1,U_2 be the components containing u_1,u_2, respectively. The new dag $\widehat{D}_{a,b}$ is obtained from the current one by adding arcs (U_1,U_2) and (U_2,U_1) and shrinking the new strongly connected component containing both U_1 and U_2. This transformation can be easily extended to the underlying strip $\mathcal{W}_{a,b}$ as follows.

If $U_1=U_2$, then there are no changes in $\mathcal{W}_{a,b}$. Otherwise, U_1 and U_2 are contracted to a new unit $U^{\rm new}$.

If U_1 and U_2 are not mutually reachable in $\mathcal{W}_{a,b}$, then there are no other changes in $\mathcal{W}_{a,b}$ (see Figure 4.2(a)). Otherwise, assume w.l.o.g. that $U_2\in\mathcal{R}_b(U_1)$ and $U_1\in\mathcal{R}_a(U_2)$; then $\mathcal{R}_b(U_1)\cap\mathcal{R}_a(U_2)$ (that is, the set of all the units and edges of all the paths between U_1 and U_2) is contracted into $U^{\rm new}$ as well (see Figure 4.2(b)).

If both U_1 and U_2 are terminals, then the system of S-mincuts vanishes. If exactly one of them is a terminal, then $U^{\rm new}$ is the corresponding terminal in the new strip. If both U_1 and U_2 are stretched, then $U^{\rm new}$ is stretched as well, and the side of the 2-partition at $U^{\rm new}$ labeled by a (resp., b) is glued from the sides of the 2-partitions at all the contracted units labeled by the same letter (except for the internal edges of the contracted subgraph).

Observe that the dag of the strongly connected components is exactly the object maintained by algorithms [I, LL], with the time complexity $O(mn)$. Evidently, any such algorithm can be easily modified to maintain a strip, with the same complexity.

As was mentioned in the previous subsection, the above description covers also a more general situation, when σ is arbitrary, but all the S-mincuts divide S in the same way into two subsets S_1 and S_2. Recall that by Lemma 2.3 in this case $\mathcal{F}_S=\mathcal{W}_{S_1,S_2}=\mathcal{W}_{s_1,s_2}=\mathcal{F}_{\{s_1,s_2\}}$ for any $s_1\in S_1$, $s_2\in S_2$.

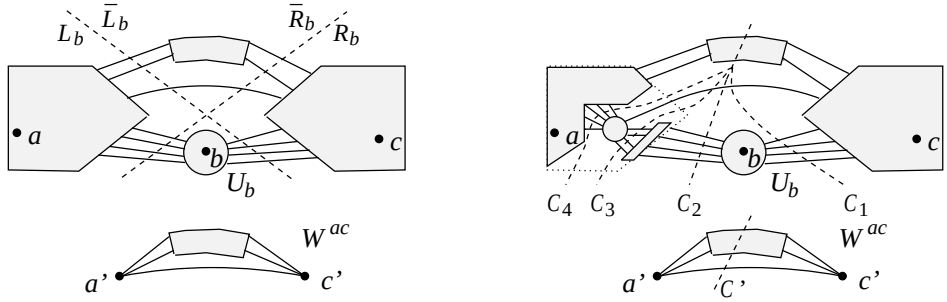


FIG. 4.3. *Structure of $\mathcal{F}_{\{a,b,c\}}$ in the asymmetric case.*

4.3. A three-element subset S: The asymmetric subcase. The smallest nontrivial case, from the static point of view, is $S = \{a, b, c\}$ and any two of these vertices are separated by an S-mincut. In this subsection we consider the asymmetric subcase; we assume w. l. o. g. that there are no $(b, \{a, c\})$-cuts. Thus, every S-mincut is an (a, c)-cut, and vice versa. Therefore, the entire family of S-mincuts is represented by the strip $\mathcal{W}_{a,c}$; in particular, (a, c)-units coincide with S-units. We define the structure of local orientability on the graph $\mathcal{F}_{\{a,b,c\}}$ by keeping that of $\mathcal{W}_{a,c}$ at all the units, except for the unit U_b containing b, which is now regarded as a terminal. Since $\mathcal{W}_{a,c}$ is acyclic and balanced, and turning a vertex to a terminal does not extend the set of coherent paths, we see that $\mathcal{F}_{\{a,b,c\}}$ is acyclic and balanced as well. Besides, we see that all S-mincuts of G are transversal cuts in $\mathcal{F}_{\{a,b,c\}}$; observe that the converse is not true; see, e.g., the cut that separates U_b from all the other units.

We denote by $(L_b, \bar{L}_b)$ the $\{a, b\}$-tight $(\{a, b\}, c)$-mincut (for the definition see section 2 immediately after Fact 2.2). Similarly, $(R_b, \bar{R}_b)$ is the $\{b, c\}$-tight $(\{b, c\}, a)$-mincut (see Figure 4.3(a)). Observe that according to Lemma 2.1(i), there are no edges between $L_b \cap R_b$ and $\bar{L}_b \cap \bar{R}_b$.

In our next statement we summarize other properties of the cuts $(L_b, \bar{L}_b)$ and $(R_b, \bar{R}_b)$.

LEMMA 4.1. (i) *L_b and R_b are the two reachability cones of the unit U_b in $\mathcal{W}_{a,c}$, and hence $L_b \cap R_b$ coincides with this unit.*

(ii) *The sides of the star of U_b in $\mathcal{W}_{a,c}$ are induced in $\mathcal{F}_{\{a,b,c\}}$ by edge-sets of $(L_b, \bar{L}_b)$ and $(R_b, \bar{R}_b)$.*

(iii) *The contraction of L_b in $\mathcal{F}_{\{a,b,c\}}$ gives $\mathcal{W}_{b,c}$, the contraction of R_b in $\mathcal{F}_{\{a,b,c\}}$ gives $\mathcal{W}_{a,b}$.*

Proof. (i) The proof follows from Fact 3.6(iii).

(ii) The proof follows from (i) and Fact 3.3.

(iii) The proof follows from Lemma 3.8. □

Evidently, either U_b is a separating vertex of $\mathcal{F}_{\{a,b,c\}}$, or $\mathcal{F}_{\{a,b,c\}} \setminus U_b$ is connected. In the latter case we build from $\mathcal{F}_{\{a,b,c\}}$ a locally orientable graph $\mathcal{W}^{ac}$ by deleting all the vertices of U_b and contracting $L_b \cap \bar{R}_b$ to the terminal a' and $\bar{L}_b \cap R_b$ to the terminal c' (see Figure 4.3(a)). (Informally, $\mathcal{W}^{ac}$ is the common part of $\mathcal{W}_{a,b}$ and $\mathcal{W}_{b,c}$ in $\mathcal{F}_{\{a,b,c\}}$.) An S-mincut $\mathcal{C}$ of G that separates $L_b \cap \bar{R}_b$ from $\bar{L}_b \cap R_b$ and thus generates an (a', c')-mincut $\mathcal{C}'$ in $\mathcal{W}^{ac}$ we call an *extension* of $\mathcal{C}'$.

LEMMA 4.2. (i) *The graph $\mathcal{W}^{ac}$ is a strip of width $\lambda' = \lambda_S - \frac{1}{2} \deg U_b < \frac{1}{2}\lambda_S$.*

(ii) *All of its transversal cuts, and only they, are extendible to S-mincuts.*

(iii) *If a 2-partition of $L_b \cup R_b$ enters an extension of some transversal cut of*

FIG. 4.4. *Dynamics of $\mathcal{F}_{\{a,b,c\}}$ in the asymmetric case.*

$\mathcal{W}^{ac}$, *then it enters some extension of any such cut.* (*Informally, all transversal cuts of $\mathcal{W}^{ac}$ are interchangeable in S-mincuts.*)

Proof. (i) Observe that deletion of vertices does not give rise to new coherent paths; thus, it preserves the acyclicity of a locally orientable graph. For the same reason, any transversal 2-partition induces a transversal 2-partition in the new graph. Since the contraction of a side of a transversal 2-partition preserves acyclicity, $\mathcal{W}^{ac}$ is an acyclic two-terminal balanced locally orientable graph. Hence, it is a strip by part (i) of the Strip Lemma. It is easy to see, applying Lemma 2.1, that $\lambda_S = c(L_b, \bar{L}_b) = \deg a' + \frac{1}{2}\deg U_b$. Since $\lambda' = \deg a'$, we get $\lambda' = \lambda_S - \frac{1}{2}\deg U_b$. Assume that $\lambda' \geqslant \frac{1}{2}\lambda_S$. Then $\deg U_b \leqslant \lambda_S$, which means that $(L_b \cap R_b, \bar{L}_b \cup \bar{R}_b)$ is a 2-partition of cardinality at most λ_S separating b from $\{a, c\}$, a contradiction.

(ii) Let us denote by $\mathcal{W}'$ the graph obtained from $\mathcal{F}_{\{a,b,c\}}$ by deleting U_b. By (i), any transversal cut $(Z, \bar{Z})$ of $\mathcal{W}^{ac}$ has cardinality λ' and generates a 2-partition $(X, \bar{X})$ of the same cardinality in $\mathcal{W}'$. By Lemma 2.1, the edge-set of the 2-partition $(X \cup U_b, \bar{X})$ in $\mathcal{F}_{\{a,b,c\}}$ has exactly by $\frac{1}{2}\deg U_b$ more edges, i.e., $\lambda' + \frac{1}{2}\deg U_b = \lambda_S$ edges. Thus, it is a cut and is an extension of $(Z, \bar{Z})$.

Conversely, let $(X, \bar{X})$ be an S-mincut separating $L_b \cap \bar{R}_b$ from $\bar{L}_b \cap R_b$. It generates a 2-partition of cardinality λ' in $\mathcal{W}'$ and a 2-partition of the same cardinality separating a' and c' in $\mathcal{W}^{ac}$. Thus, it is a λ'-cut separating a' and c', i.e., a transversal cut.

(iii) Since there are exactly two such partitions, namely, $((L_b \cap \bar{R}_b) \cup U_b, \bar{L}_b \cap R_b)$ and $(L_b \cap \bar{R}_b, (\bar{L}_b \cap R_b) \cup U_b)$, one can proceed exactly as in the proof of (ii). □

Remark. Observe that Lemma 4.2 remains true for a more general definition of an extension, in which $\mathcal{C}$ is required only to separate the ends of all edges between $L_b \cap \bar{R}_b$ and $\bar{L}_b \cap \bar{R}_b$ that lie in $L_b \cap \bar{R}_b$ and the ends of all edges between $\bar{L}_b \cap \bar{R}_b$ and $\bar{L}_b \cap R_b$ that lie in $\bar{L}_b \cap R_b$. Such a definition would allow us to consider, for example, the cuts $\mathcal{C}_3$ and $\mathcal{C}_4$ on Figure 4.3(b) as extensions of $\mathcal{C}'$, while according to the current definition, the only extensions of $\mathcal{C}'$ are $\mathcal{C}_1$ and $\mathcal{C}_2$. However, in what follows we do not consider such a generalization of the notion of extension.

Dynamics of $\mathcal{F}_{\{a,b,c\}}$ in the asymmetric subcase are induced by those of $\mathcal{F}_{\{a,c\}} = \mathcal{W}_{a,c}$ (see section 4.2) straightforwardly, except for the case when U_b, and possibly some other units, is contracted to U^{new}. Let us consider this case; it occurs when $U_b \in \mathcal{R}_c(U_1) \cap \mathcal{R}_a(U_2)$, up to flipping of U_1 and U_2 (see Figure 4.4). The only specifics in this case is that U^{new} becomes a new terminal (in fact, $\widehat{U}_b$).

Remark. To translate the definition of the contracted set into the terms of $\mathcal{F}_{\{a,b,c\}}$, one has to take into account that a coherent path in $\mathcal{W}_{a,c}$ is either a coherent path in $\mathcal{F}_{\{a,b,c\}}$, or a concatenation of two coherent paths: a path to U_b in L_b and a path from U_b in R_b. Under this translation, the above condition on U_b is equivalent to $U_1 \in \mathcal{R}_a(U_b) = L_b$ and $U_2 \in \mathcal{R}_c(U_b) = R_b$.

There is a special situation when not only U_b but U_a as well is contracted to U^{new}; this occurs when $U_1 = U_a$ (see Figure 4.4(b)). In this situation the asymmetric

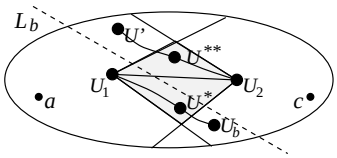


FIG. 4.5. *To the proof of Lemma* 4.3.

three-element case degenerates, and we arrive at the case considered at the end of section 4.2; that is, $\mathcal{F}_{\{a,b,c\}}$ becomes equal to $\mathcal{W}_{\{a,b\},c} = \mathcal{F}_{b,c} = \mathcal{F}_{a,c}$. Evidently, U_a above can be replaced by U_c.

Finally, if $\{U_1, U_2\} = \{U_a, U_b\}$, then the system of S-mincuts vanishes.

As we see, the reduction of the three-element case to the two-element case is defined via subsets L_b and R_b. Let us consider the dynamics of L_b (those of R_b are similar) in the case when $\mathcal{F}_{\{a,b,c\}}$ does not degenerate.

LEMMA 4.3. *The set L_b is extended if and only if $U_1 \in L_b$ and $U_2 \notin L_b$ (up to flipping of U_1 and U_2). The new part of $\widehat{L}_b$ is the union of units U' such that $U' \in \mathcal{R}_a(U_2)$ and $U' \notin L_b$.*

Proof. The "if" part of the first statement is trivial, since U_2 is added to L_b.

Let U' be a new unit in $\widehat{L}_b$ and let $(U', \dots, U^{\text{new}}, \dots, U_b)$ be a coherent path in $\widehat{\mathcal{W}}_{a,c}$; clearly, it lies entirely in $\widehat{\mathcal{R}}_c(U')$. Its edges form two coherent paths in $\mathcal{F}_{\{a,b,c\}}$: $(U', \dots, U^*) \in \mathcal{R}_c(U')$ and $(U^{**}, \dots, U_b) \in \mathcal{R}_c(U^{**})$ (see Figure 4.5). Since U' is new in $\widehat{L}_b$, we have $U^* \neq U^{**}$, and thus the statement holds for the case when U^{new} contains only U_1 and U_2. Otherwise we may assume w.l.o.g. that $U^*, U^{**} \subset U^{\text{new}} = \mathcal{R}_c(U_1) \cap \mathcal{R}_a(U_2)$. Hence in $\mathcal{W}_{a,c}$ there exist coherent paths $(U^*, \dots, U_2) \in \mathcal{R}_c(U^*)$ and $(U_1, \dots, U^{**}) \in \mathcal{R}_c(U_1)$. The first of them does not contain U_b by assumption, and thus $(U', \dots, U^*, \dots, U_2)$ is a coherent path in $\mathcal{F}_{\{a,b,c\}}$. Therefore, $U_2 \in L_b$ would imply $U' \in L_b$ in a contradiction to the assumption that U' is new in $\widehat{L}_b$. The path $(U_1, \dots, U^{**}, \dots, U_b)$ does not contain U_b twice (as a coherent path in $\mathcal{W}_{a,c}$), and thus is a coherent path in $\mathcal{F}_{\{a,b,c\}}$; hence, $U_1 \in L_b$. The assertion follows. ∎

4.4. A three-element subset S: The symmetric subcase. Assume now that all types of $\{a,b,c\}$-cuts exist in G. Let $\mathcal{C}_x = (V_x, \bar{V}_x)$ be the $\{y,z\}$-tight $(\{y,z\}, x)$-mincut (Figure 4.6(a)); here and in what follows $x, y, z \in \{a,b,c\}$, $x \neq y \neq z$. As usual, these three 2-partitions define the following eight subsets of V: $V^{\varnothing} = V_a \cap V_b \cap V_c$; $V^a = \bar{V}_a \cap V_b \cap V_c$; $V^b = V_a \cap \bar{V}_b \cap V_c$; $V^c = V_a \cap V_b \cap \bar{V}_c$; $V^{ab} = \bar{V}_a \cap \bar{V}_b \cap V_c$; $V^{bc} = V_a \cap \bar{V}_b \cap \bar{V}_c$; $V^{ac} = \bar{V}_a \cap V_b \cap \bar{V}_c$; $V^{abc} = \bar{V}_a \cap \bar{V}_b \cap \bar{V}_c$. These subsets are called *cells*.

The following statement is crucial for the rest of our results (see Figure 4.6(b)).

3-STAR LEMMA. (i) $V^{abc} = \varnothing$.

(ii) *Vertices of V^{xy} can be adjacent only to vertices of V^{xy}, V^x, V^y; besides, $c(V^{xy}, V^x) = c(V^{xy}, V^y)$.*

Proof. Let us consider the cut $\mathcal{C}^{x,xy} = \mathcal{C}_z \cap \bar{\mathcal{C}}_x = (V^x \cup V^{xy}, V \setminus (V^x \cup V^{xy}))$. Since both $\mathcal{C}_z$, $\mathcal{C}_x$ are minimum (x,z)-cuts, the cut $\mathcal{C}^{x,xy}$ is a minimum (x,z)-cut as well, and hence $c(\mathcal{C}^{x,xy}) = \lambda_S$. Similarly, $\mathcal{C}^{y,xy} = \mathcal{C}_z \cap \bar{\mathcal{C}}_y = (V^y \cup V^{xy}, V \setminus (V^y \cup V^{xy}))$ is a minimum (y,z)-cut with $c(\mathcal{C}^{y,xy}) = \lambda_S$. Therefore, both $\mathcal{C}^{x,xy}$ and $\mathcal{C}^{y,xy}$ are minimum (x,y)-cuts. According to Lemma 2.1(i), there are no edges between V^{xy} and $V \setminus (V^x \cup V^{xy} \cup V^y)$, and, moreover, $c(V^{xy}, V^x) = c(V^{xy}, V^y)$, so assertion (ii) is proved.

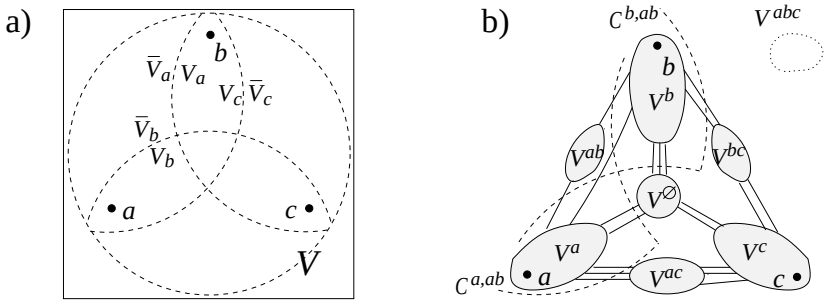


FIG. 4.6. *Structure of $\mathcal{F}_{\{a,b,c\}}$ in the symmetric case.*

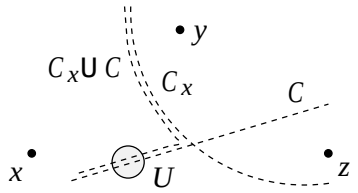


FIG. 4.7. *To the proof of Lemma* 4.4(i).

We know already that there are no edges between V^{abc} and V^{xy}. By Lemma 2.1(i) applied to the minimum (x,y)-cuts $\mathcal{C}_x$ and $\mathcal{C}_y$, we get that there are no edges between $V^{xy} \cup V^{abc}$ and $V^z \cup V^\varnothing$. So, there are no external edges incident to V^{abc} at all. Since the initial graph is connected, we thus get $V^{abc} = \varnothing$. □

Properties of the cuts $\mathcal{C}_x$ are similar to those of the cuts $(L_b, \bar{L}_b)$ and $(R_b, \bar{R}_b)$ (see section 4.2). In particular, the first statement below can be regarded as an analogue of Lemma 4.1(iii).

LEMMA 4.4. (i) *Contraction of the inner side V_x of $\mathcal{C}_x$ in $\mathcal{F}_{\{a,b,c\}}$ gives the underlying undirected graph of the strip $\mathcal{W}_{\{y,z\},x}$.*

(ii) *Every unit, except for $V^\varnothing$, corresponds in this way to a vertex in either one or two strips.*

(iii) *For every unit $U \neq V^\varnothing$, the 2-partitions of its star induced by the 2-partitions at the corresponding vertices in these strips coincide.*

Proof. (i) Since all $(\{y,z\},x)$-mincuts are S-mincuts and $\mathcal{C}_x$ is the $\{y,z\}$-tight $(\{y,z\},x)$-mincut, the result of the contraction still represents all $(\{y,z\},x)$-mincuts. Therefore, the only thing we have to prove is that no $(\{y,z\},x)$-unit distinct from the contracted one can be divided by an S-mincut. Indeed, let U, to the contrary, be such a unit and $\mathcal{C}$ be an S-mincut dividing U. W.l.o.g. we can assume that $\mathcal{C}$ is an $(z,\{x,y\})$-cut containing z inside; recall that z lies also inside $\mathcal{C}_x$ (see Figure 4.7). Then $\mathcal{C}_x \cup \mathcal{C}$ is a $(\{y,z\},x)$-mincut dividing U, a contradiction.

(ii) The proof follows from (i) immediately.

(iii) Assume w.l.o.g. that U lies in V^{xy}. According to (i) and (ii), U is a unit in the strips $\mathcal{W}_{\{y,z\},x}$ and $\mathcal{W}_{\{x,z\},y}$. By Lemma 3.8, both these strips are contractions of the same strip $\mathcal{W}_{x,y}$. Moreover, in both cases U is not a contracted unit, so local orientations at U in both cases are inherited from $\mathcal{W}_{x,y}$, and thus coincide. □

Let us assign to every unit, except for $V^\varnothing$, the 2-partition of its star defined in

Lemma 4.4(iii), and to $V^{\varnothing}$ the trivial 2-partition. Then $\mathcal{F}_{\{a,b,c\}}$ becomes a locally orientable graph. Evidently, the contraction of $\mathcal{F}_{\{a,b,c\}}$ described in Lemma 4.4(i) yields now the strip $\mathcal{W}_{\{y,z\},x}$ itself. One can say, thus, that all the strips $\mathcal{W}_{\{y,z\},x}$ "are glued" into the aggregate structure $\mathcal{F}_{\{a,b,c\}}$ because they "coincide on intersections."

LEMMA 4.5. (i) *The locally orientable graph $\mathcal{F}_{\{a,b,c\}}$ is balanced and acyclic.*

(ii) *Each S-mincut is transversal in $\mathcal{F}_{\{a,b,c\}}$.*

Proof. (i) Evidently, $\mathcal{F}_{\{a,b,c\}}$ is balanced, since the 2-partition at each nonterminal unit is inherited from a strip. Let us prove its acyclicity. Indeed, let there exist a cyclic coherent path P. By the definition, any two units lying on P are separated by an S-mincut. However, this cut fails to be transversal, since its edge-set has at least two edges in common with P. Thus, (i) follows from (ii).

(ii) First, let us observe that a coherent path in $\mathcal{F}_{\{a,b,c\}}$ that enters V^x cannot leave it. Indeed, the 2-partition $(V^x, \bar{V}^x)$ can be represented as $\mathcal{C}^{x,xy} \cap \mathcal{C}^{x,xz}$ (see the proof of the 3-Star Lemma) and is thus a minimum $(x, \{y,z\})$-cut. Therefore, a coherent path in $\mathcal{F}_{\{a,b,c\}}$ that enters and leaves V^x yields in $\mathcal{W}_{x,\{y,z\}}$ a coherent path that intersects at least twice the edge-set of the transversal cut $(V^x, \bar{V}^x)$, a contradiction.

Now let $\mathcal{C}$ be an S-mincut contradicting the claim, and let P be a coherent path intersecting $E_{\mathcal{C}}$ at least twice. Assume w.l.o.g. that $\mathcal{C}$ is an $(x, \{y,z\})$-cut; then it is represented as a transversal cut in $\mathcal{W}_{x,\{y,z\}}$. If the path P lies entirely in $V^x \cup V^{xy} \cup V^{xz}$, then it is represented as a coherent path in $\mathcal{W}_{x,\{y,z\}}$, a contradiction. Thus, by part (ii) of the 3-Star Lemma, P must enter either $V^{\varnothing}$, or V^y, or V^z. However, $V^{\varnothing}$ is a terminal, and a path entering V^y or V^z cannot leave them. Hence, in this case the contraction takes P to a coherent path in $\mathcal{W}_{x,\{y,z\}}$, and we are done. □

Therefore, an arbitrary undirected graph with three distinguished vertices in the symmetric case, as well as in the asymmetric one, defines canonically an acyclic balanced locally orientable graph $\mathcal{F}_{\{a,b,c\}}$.

Remark. Recall that a strip can be considered as a general graph model of a two-ended object of a constant width. In a sense, the locally orientable graph $\mathcal{F}_{\{a,b,c\}}$ with the structure given by Figure 4.6(b) represents a general *three-ended object of a constant width.*

The following statement stems from the proof of Lemma 4.5.

FACT 4.6. *Any intercell coherent path in $\mathcal{F}_{\{a,b,c\}}$ belongs to one of the following four types*: V^x-V^{xy}; V^x-V^{xy}-V^y; V^x-V^y; V^x-$V^{\varnothing}$.

According to Lemma 4.1(i) and (ii), in the asymmetric subcase one can reconstruct the strip $\mathcal{W}_{a,c}$ starting from $\mathcal{F}_{\{a,b,c\}}$ and using the cuts $(L_b, \bar{L}_b)$ and $(R_b, \bar{R}_b)$. In a similar way, in the symmetric subcase one can obtain each of the three strips $\mathcal{W}_{x,y}$ by a contraction of $\mathcal{F}_{\{a,b,c\}}$ defined in terms of the cuts $\mathcal{C}_x$.

LEMMA 4.7. *Let us contract $V_y \cap V_z$ in $\mathcal{F}_{\{a,b,c\}}$ and assign to the contracted unit the 2-partition of its star into the edges crossing the cuts $\mathcal{C}_y$ and $\mathcal{C}_z$, respectively. The result is the strip $\mathcal{W}_{y,z}$.*

Proof (to visualize the statement and the proof observe that $V_y \cap V_z = V^{\varnothing} \cup V^x$, see Figure 4.6). Let us denote by $\mathcal{W}$ the locally orientable graph defined in the statement of the lemma. (It is well defined due to part (ii) of the 3-Star Lemma.) We consider the two following ways of constructing the strip $\mathcal{W}_{\{x,z\},y}$. First, we take the strip $\mathcal{W}_{y,z}$ and, according to Lemma 3.8, contract the cone of x (more exactly, of the unit U_x containing x) that contains z. Second, we take $\mathcal{F}_{\{a,b,c\}}$ and contract the inner part V_y of the cut $\mathcal{C}_y$ (see Lemma 4.4(i)). Since both ways yield the same result, we

conclude that all units $U \in V^y \cup V^{yx} \cup V^{yz}$ in $\mathcal{W}_{y,z}$ and $\mathcal{F}_{\{a,b,c\}}$ coincide, as well as the 2-partitions at these units. Proceeding in the same way with the strip $\mathcal{W}_{\{x,y\},z}$, we get the coincidence for all units, except for U_x in $\mathcal{W}_{y,z}$ and $V_y \cap V_z$, which is a single unit in $\mathcal{W}$; thus, $U_x = V_y \cap V_z$. Observe that, on one hand, the two sides of U_x in $\mathcal{W}$ are just the intersections of its star with the edge-sets of the cuts $\mathcal{C}_y$ and $\mathcal{C}_z$. On the other hand, by Fact 3.6(iii), these two cuts correspond to the cones of U_x in $\mathcal{W}_{y,z}$ via our contraction mapping. Thus, the 2-partitions at U_x in the two graphs coincide as well, and the proof is completed. □

Similarly to the asymmetric case, one can build a locally orientable graph $\mathcal{W}^{xy}$, $x, y \in \{a, b, c\}$, $x \neq y$, by deleting from $\mathcal{F}_{\{a,b,c\}}$ all the vertices of $V^{\varnothing} \cup V^z \cup V^{xz} \cup V^{yz}$, $z \neq x, y$, and contracting V^x and V^y to terminals. (Informally, $\mathcal{W}^{ab}$, $\mathcal{W}^{bc}$, and $\mathcal{W}^{ac}$ are the common parts of all three strips $\mathcal{W}_{a,b}$, $\mathcal{W}_{b,c}$, and $\mathcal{W}_{a,c}$.) An S-mincut $\mathcal{C}$ of G that separates V^x from V^y, and thus generates a cut $\mathcal{C}'$ in $\mathcal{W}^{xy}$ separating its terminals, we call an *extension* of $\mathcal{C}'$ (compare with the definition preceding Lemma 4.2). The following statement is an analogue of Lemma 4.2.

LEMMA 4.8. (i) *The graphs $\mathcal{W}^{xy}$ are strips of width $\lambda^{xy} \leqslant \frac{1}{2}\lambda_S$.*

(ii) *All their transversal cuts, and only they, are extendible to S-mincuts.*

(iii) *Each pair of transversal cuts of $\mathcal{W}^{xy}$ and $\mathcal{W}^{xz}$, $z \neq x, y$, has a mutual extension to a minimum $(x, \{y, z\})$-cut.*

(iv) *If a 2-partition of $V \setminus V^{xy}$ enters an extension of some transversal cut of $\mathcal{W}^{xy}$, then it enters some extension of any such cut.* (*Informally, all transversal cuts of this strip are interchangeable in S-mincuts.*)

Proof. (i) Observe that, according to Lemma 4.7, one can construct $\mathcal{W}^{xy}$ in a similar way starting from $\mathcal{W}_{x,y}$. Therefore, the proof of Lemma 4.2(i) applies with minor changes: the assumption $\lambda^{xy} > \frac{1}{2}\lambda_S$ implies the existence of a 2-partition of cardinality less than λ_S separating z from $\{x, y\}$.

(ii) The "all" part of the assertion is proved exactly as in Lemma 4.2. To prove the "only" part we consider an arbitrary cut $\mathcal{C}$ separating V^x from V^y. Assume, w.l.o.g., that $\mathcal{C}$ is an $(x, \{y, z\})$-cut containing x inside. Replacing $\mathcal{C}$ by $\mathcal{C} \cup \mathcal{C}_y$ we arrive to the case considered in the proof of Lemma 4.2(ii).

(iii) It follows easily from the proof of Lemma 4.2(ii) that any transversal cut $\mathcal{C}'_1$ of $\mathcal{W}^{xy}$ can be extended to an $(x, \{y, z\})$-cut $\mathcal{C}_1$ of $\mathcal{F}_{\{a,b,c\}}$. Then $\tilde{\mathcal{C}}_1 = \mathcal{C}_1 \cap \mathcal{C}_z$ is a minimum (x, z)-cut, as the intersection of minimum (x, z)-cuts. Observe that $\tilde{\mathcal{C}}_1$ is an $(x, \{y, z\})$-cut. Similarly, starting from a transversal cut $\mathcal{C}'_2$ of $\mathcal{W}^{xz}$ we find an $(x, \{y, z\})$-cut $\tilde{\mathcal{C}}_2$. Their union is evidently a mutual extension of $\mathcal{C}'_1$ and $\mathcal{C}'_2$. Observe that $\tilde{\mathcal{C}}_1 \cup \tilde{\mathcal{C}}_2$ is the unique mutual extension of $\mathcal{C}'_1$ and $\mathcal{C}'_2$, since by definition, each such extension contains V^x inside and $V^{\varnothing} \cup V^y \cup V^z \cup V^{yz}$ outside.

(iv) Let $\mathcal{C}$ be an extension of some transversal cut of $\mathcal{W}^{xy}$ and let $\mathcal{C}'_1$ be an arbitrary transversal cut of $\mathcal{W}^{xy}$. Assume w.l.o.g. that both x and $V^{\varnothing}$ lie inside $\mathcal{C}$. Recall that $\mathcal{C}^{y,xy}$ defined in the proof of the 3-Star Lemma is a minimum (x, y)-cut. Therefore, $\tilde{\mathcal{C}} = \mathcal{C} \cap \bar{\mathcal{C}}^{y,xy}$ is also a minimum (x, y)-cut. On the other hand, similarly to the proof of Lemma 4.2(ii), there exists an extension $\mathcal{C}_1$ of $\mathcal{C}'_1$ such that $V^x \cup V^{xz}$ lies inside $\mathcal{C}_1$ and $V^y \cup V^{yz} \cup V^{\varnothing} \cup V^z$ outside it. The cut $\tilde{\mathcal{C}} \cup \mathcal{C}_1$ is thus an extension of $\mathcal{C}'_1$ (since $\bar{\mathcal{C}}$ is an extension of the cut separating the unit U_x in $\mathcal{W}^{xy}$), and it partitions $V \setminus V^{xy}$ in the same way as $\mathcal{C}$. □

Let us analyze the incremental dynamics of $\mathcal{F}_S$ in the symmetric subcase. Since $\mathcal{F}_S$ is glued from the strips $\mathcal{W}_{x,\{y,z\}}$ (or their extensions $\mathcal{W}_{x,y}$), it suffices to study the dynamics of these strips. According to section 4.2, insertion of an edge (u_1, u_2) modifies each of the strips in such a way that all the units, as well as the canonical

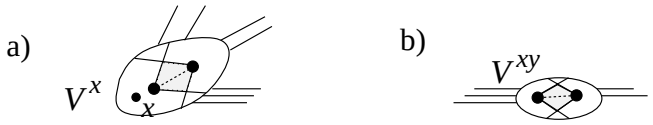


FIG. 4.8. *Dynamics of $\mathcal{F}_{\{a,b,c\}}$ in the symmetric case,* I.

2-partitions of their stars, remain the same except for the units contracted into the new one (containing both u_1 and u_2). Hence, by Lemma 4.4, the same holds for the locally orientable graph $\mathcal{F}_S$. According to the same lemma, if for some strip $\widehat{\mathcal{W}}_{x,\{y,z\}}$ the single new unit is not the terminal containing y and z, then it coincides with the new unit U^{new} of $\mathcal{F}_S$. The same is true if for some strip $\widehat{\mathcal{W}}_{x,y}$ the new unit is not the unit containing z.

In the description of incremental dynamics of $\mathcal{F}_S$ we make use of the partition into the cells V^x, V^{xy}, $V^{\varnothing}$. Therefore, to be self-contained, we have to describe the changes in the cells; note that it suffices for this to clear up the dynamics of the sets V_x.

Below we consider all the nine distinct possibilities to insert a new edge (u_1, u_2), $u_1 \in U_1$, $u_2 \in U_2$, with respect to the cell partition.

Case 1. If $U_1 = U_2$ (in particular, $U_1 = U_2 = V^{\varnothing}$), then $\mathcal{F}_S$ evidently remains the same.

Case 2. If U_1 and U_2 belong to the same set V^{xy} (see Figure 4.8(a)), we can trace the dynamics in any of $\mathcal{W}_{x,\{y,z\}}$, $\mathcal{W}_{\{x,z\},y}$, and $\mathcal{W}_{x,y}$. Clearly, the definition of the contracted unit U^{new} (see section 4.2) can be translated straightforwardly into terms of $\mathcal{F}_S$: U^{new} results from the contraction of U_1, U_2, and the units and edges on the coherent paths between U_1 and U_2 (all of them belong to V^{xy}). Since the contracted unit in the modified strip is nonterminal, this contraction is the only change in $\mathcal{F}_S$. Note that the changes are localized in V^{xy} and can be traced also in $\mathcal{W}^{xy}$.

Case 3. Similarly, if $U_1, U_2 \in V^x$ (see Figure 4.8(b)), we can trace the changes in any of the strips $\mathcal{W}_{x,\{y,z\}}$, $\mathcal{W}_{x,y}$, or $\mathcal{W}_{x,z}$ to define U^{new} in the same way. It is easy to show that the changes are localized in V^x.

In all the above cases all the cuts $\mathcal{C}_a$, $\mathcal{C}_b$, $\mathcal{C}_c$ remain the same, and thus the partition into cells is stable.

Case 4. Let now $U_1 \in V^x$, $U_2 \in V^{xy}$ (see Figure 4.9(a)). As in the previous two cases, the new unit U^{new} can be defined via $\mathcal{W}_{x,\{y,z\}}$, $\mathcal{W}_{x,y}$, or $\mathcal{W}_{x,z}$. The strip $\mathcal{W}_{x,y}$ can be used also for establishing the dynamics of the set V_y via Lemma 4.3; in terms of this lemma, it is the set L_z. By the same lemma, the new part of $\widehat{V}_y$ is $V' = \mathcal{R}_x(U_2) \setminus V_y$. Another way is to use $\mathcal{W}_{\{x,z\},y}$, where the set V_y forms the terminal U_{xz} containing x and z; then the new part of $\widehat{V}_y = \widehat{U}_{yz}$ is $\mathcal{R}_x(U_2) \setminus U_{yz}$. It follows from Fact 4.6 that V' is contained in V^{xy} and coincides with the cone of U_2 in $\mathcal{W}^{xy}$ in the direction to the terminal containing x, minus this terminal. From the dynamics of $\mathcal{W}_{x,\{y,z\}}$ and $\mathcal{W}_{z,\{x,y\}}$, respectively, it is clear that the sets V_x and V_z do not change. Therefore, all the cells are stable except for $\widehat{V}^x = V^x \cup V'$ and $\widehat{V}^{xy} = V^{xy} \setminus V'$.

Case 5. If $U_1 \in V^{xz}$, $U_2 \in V^{xy}$ (see Figure 4.9(b)), we can use strips $\mathcal{W}_{x,\{y,z\}}$, $\mathcal{W}_{x,y}$, and $\mathcal{W}_{x,z}$ for determining U^{new}. In this case, there are no coherent paths between U_1 and U_2; hence, $U^{\text{new}} = \{U_1, U_2\}$. Using strips $\mathcal{W}_{x,y}$ and $\mathcal{W}_{x,z}$, respectively,

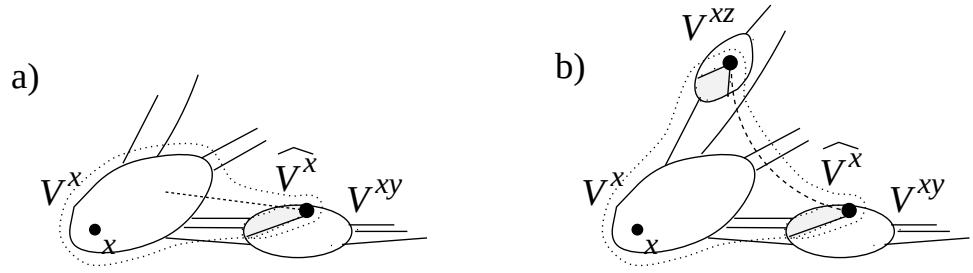


FIG. 4.9. *Dynamics of* $\mathcal{F}_{\{a,b,c\}}$ *in the symmetric case,* II.

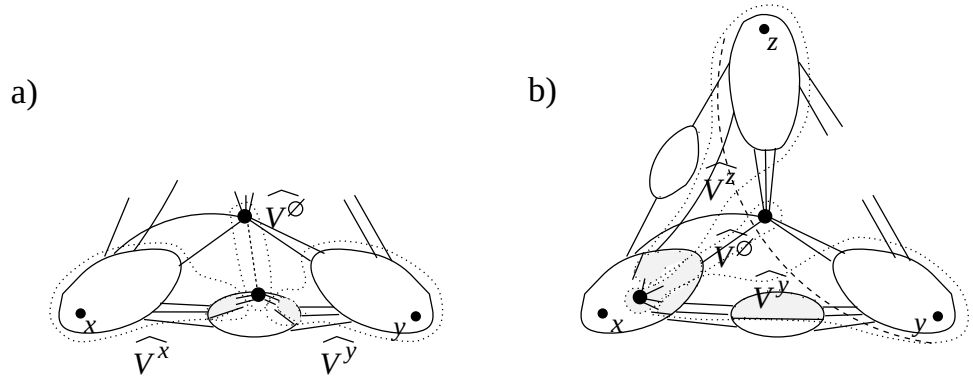


FIG. 4.10. *Dynamics of* $\mathcal{F}_{\{a,b,c\}}$ *in the symmetric case,* III.

in the same way as $\mathcal{W}_{\{x,z\},y}$ in the previous case, we establish that $\widehat{V}_y = V_y \cup \mathcal{R}_x(U_2)$ and $\widehat{V}_z = V_z \cup \mathcal{R}_x(U_1)$. Therefore, $\widehat{V}^x = V^x \cup \mathcal{R}_x(U_1) \cup \mathcal{R}_x(U_2)$, while V^{xy} and V^{xz} are cut down correspondingly.

Case 6. If $U_1 = V^\varnothing$, $U_2 \in V^{xy}$ (see Figure 4.10(a)), we use the strip $\mathcal{W}_{x,y}$. Once again $U^{\text{new}} = \{U_1, U_2\}$. Both V_x and V_y grow to $V_x \cup \mathcal{R}_x(U_2)$ and $V_y \cup \mathcal{R}_y(U_2)$, respectively. Therefore, $\widehat{V}^x = V^x \cup \mathcal{R}_x(U_2) \setminus U_2$, $\widehat{V}^x = V^x \cup \mathcal{R}_x(U_2) \setminus U_2$, and $\widehat{V}^\varnothing = V^\varnothing \cup U_2$.

Case 7. If $U_1 \in V^x$, $U_2 = V^\varnothing$ (see Figure 4.10(b)), we consider the strip $\mathcal{W}_{x,\{y,z\}}$ with terminals U_x, $x \in U_x$, and $U_{yz} = V_x$, $y, z \in U_{yz}$, $V^\varnothing \subset U_{yz}$. By section 4.2, the cone $\mathcal{R}_{yz}(U_1)$ is added to U_{yz}, resulting in $\widehat{V}_x = V_x \cup \mathcal{R}_{yz}(U_1)$. Evidently, strips $\mathcal{W}_{\{x,z\},y}$ and $\mathcal{W}_{\{x,y\},z}$ do not change; hence V_y and V_z are stable. Using the sets $\widehat{V}_x$, $\widehat{V}_y = V_y$, and $\widehat{V}_z = V_z$ in accordance to the definition of a cell, we get $U = \widehat{V}^\varnothing = V^\varnothing \cup (V^x \cap \mathcal{R}_{yz}(U_1))$, $\widehat{V}^y = V^y \cup (V^{xy} \cap \mathcal{R}_{yz}(U_1))$, $\widehat{V}^z = V^z \cup (V^{xz} \cap \mathcal{R}_{yz}(U_1))$, and each one of V^x, V^{xy}, V^{xz} is lessened by the corresponding part of $\mathcal{R}_{yz}(U_1)$.

A special subcase occurs when $U_1 = U_x$. Then $\mathcal{W}_{x,\{y,z\}}$ vanishes, and we arrive at the asymmetric case for $\mathcal{F}_{\{y,x,z\}}$. In terms of section 4.3, $\widehat{U}_x = V_x \cup V^\varnothing$, $\widehat{L}_x = V_z$, $\widehat{R}_x = V_y$, $\widehat{V}_{yz} = V_{yz}$, and $\widehat{\mathcal{W}}_{yz} = \mathcal{W}_{yz}$.

Case 8. The case $U_1 \in V^x$, $U_2 \in V^{yz}$ (see Figure 4.11(a)) is equivalent to a composition of the two previous cases: apply the transformation of Case 6 and the transformation of Case 7 with x replaced by y.

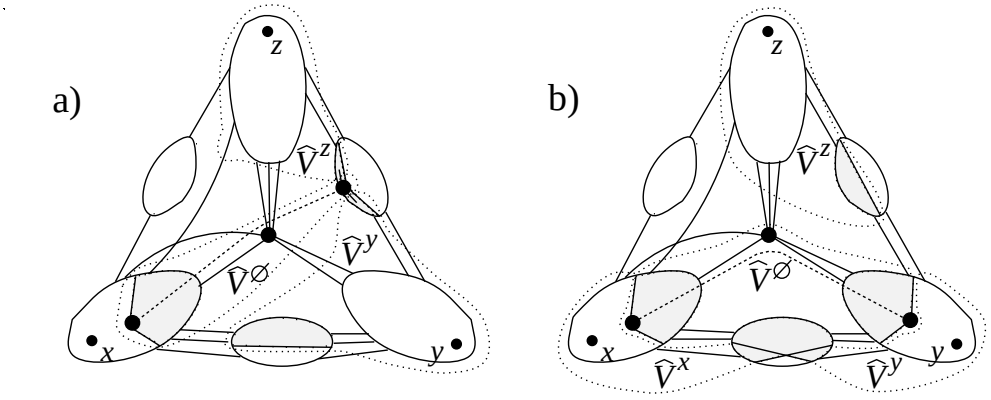


FIG. 4.11. *Dynamics of* $\mathcal{F}_{\{a,b,c\}}$ *in the symmetric case,* IV.

To see this, assume first that $V^\varnothing \neq \varnothing$ and choose an arbitrary vertex $w \in V^\varnothing$. Then the transformation of $\mathcal{F}_{\{a,b,c\}}$ equals the sum of the transformations caused by adding *two* edges: (u_1, w) and (w, u_2). Indeed, addition of each one of these edges affects only one of the strips $\mathcal{W}_{yz}$ and $\mathcal{W}_{x,\{y,z\}}$, and their dynamics are exactly the same as in Cases 6 and 7, respectively.

The case $V^\varnothing = \varnothing$ can be reduced to the previous one by the following trick. We add to G a new vertex w and three new edges (a, w), (b, w), (c, w). It is easy to see that S-mincuts of the new graph G^* correspond bijectively to S-mincuts of G; more precisely, $(X, \bar{X})$ corresponds to $(X, \bar{X} \cup \{w\})$ if and only if $|X \cap S| = 1$. Hence, all the cells of $\mathcal{F}^*_{\{a,b,c\}}$ coincide with the corresponding cells of $\mathcal{F}_{\{a,b,c\}}$ except for $(V^*)^\varnothing = \{w\}$. Since $(V^*)^\varnothing$ is now nonvoid, the transformation taking $\mathcal{F}^*_{\{a,b,c\}}$ to $\widehat{\mathcal{F}}^*_{\{a,b,c\}}$ is exactly as described above; to obtain $\widehat{\mathcal{F}}_{\{a,b,c\}}$ it suffices just to delete w.

Case 9. The last case occurs when $U_1 \in V^x$, $U_2 \in V^y$ (see Figure 4.11(b)). It can be considered as a composition of two Case 7's: apply the transformation of Case 7 as described and the same transformation with x replaced by y. In this situation the two transformations are not independent: their joint action forces the set $V' = V^{xy} \cap \mathcal{R}_{yz}(U_1) \cap \mathcal{R}_{xz}(U_2)$ to be added to both V_x and V_y and thus to move from V^{xy} to $V^\varnothing$. The validity of the above description can be observed in the same way as in the previous case; the two edges added in this case are the edges between U_1 and $V^\varnothing$ and between $V^\varnothing$ and U_2.

The only special subcase not yielded by double application of Case 7 occurs if $U_1 = U^x$, $U_2 = U^y$. It results in vanishing of both strips $\mathcal{W}_{x,\{y,z\}}$ and $\mathcal{W}_{y,\{x,z\}}$, and thus leads to degeneration of $\mathcal{F}_{\{a,b,c\}}$ to $\widehat{\mathcal{W}}_{\{x,y\},z} = \widehat{\mathcal{F}}_{x,z} = \widehat{\mathcal{F}}_{y,z}$; evidently, $\widehat{\mathcal{W}}_{\{x,y\},z}$ coincides with $\mathcal{W}_{\{x,y\},z}$.

4.5. An arbitrary subset S with the path structure. Let $S = \{a = b_0, b_1, \ldots, b_{r-1}, b_r = c\}$; assume that there exist exactly r distinct partitions of S by S-mincuts and each of them is of the form $(\{a, b_1, \ldots, b_{i-1}\}, \{b_i, \ldots, b_{r-1}, c\})$, $1 \leqslant i \leqslant r$. This means that the skeleton structure (in the sense defined in section 4.1) for this case is the path $(a, b_1, \ldots, b_{r-1}, c)$ (see Figure 4.12). Similarly to the case of the path (a, b, c) (section 4.3), all S-mincuts are represented by the (a, c)-strip $\mathcal{W}_{a,c}$. We define the structure of local orientability on the graph $\mathcal{F}_S$ by keeping that of $\mathcal{W}_{a,c}$ at all the units, except for the units U_{b_i} containing b_i, $1 \leqslant i \leqslant r-1$, which are now regarded as

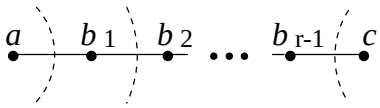


FIG. 4.12. *Path skeleton structure.*

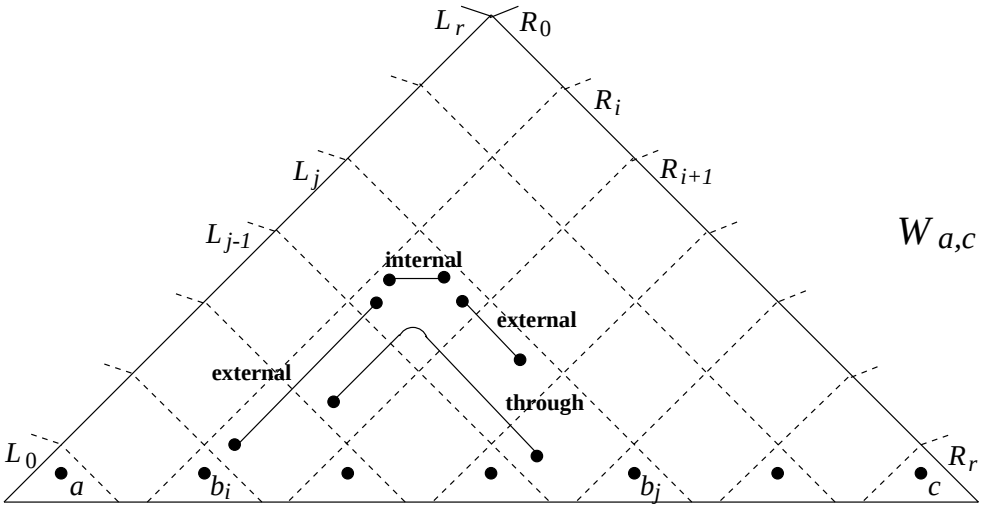


FIG. 4.13. *Cell structure for the path case.*

terminals. As in section 4.3, $\mathcal{F}_S$ is acyclic and balanced and all S-mincuts of G are transversal cuts in $\mathcal{F}_S$.

For each i, $0 \leqslant i \leqslant r-1$, let $(L_i, \bar{L}_i)$ be the $\{a, b_1, \ldots, b_i\}$-tight $(\{a, b_1, \ldots, b_i\}, \{b_{i+1}, \ldots, b_{r-1}, c\})$-mincut. Similarly, for each i, $1 \leqslant i \leqslant r$, let $(R_i, \bar{R}_i)$ be the $\{c, b_{r-1}, \ldots, b_i\}$-tight $(\{c, b_{r-1}, \ldots, b_i\}, \{b_{i-1}, \ldots, b_1, a\})$-mincut. Besides, we define L_r and R_0 to be the entire set of units of $\mathcal{F}_S$. The properties of the cuts $(L_i, \bar{L}_i)$ and $(R_i, \bar{R}_i)$ are similar to those of $(L_b, \bar{L}_b)$ and $(R_b, \bar{R}_b)$ (see Lemma 4.1) and of $\mathcal{C}_x$ (see Lemma 4.4(i) and Lemma 4.7). We omit the proof, since it is similar to that of Lemma 4.1; for illustration, see Figure 4.13.

LEMMA 4.9. (i) *For any pair i, j, $0 \leqslant i < j \leqslant r$, one has $L_i \subset L_j$ and $R_i \supset R_j$.*

(ii) *For any i, $0 \leqslant i \leqslant r$, L_i and R_i are the reachability cones of the unit U_{b_i} in $\mathcal{W}_{a,c}$, and hence $L_i \cap R_i$ coincides with this unit.*

(iii) *The sides of the star of U_{b_i} in $\mathcal{W}_{a,c}$ are induced in $\mathcal{F}_S$ by edge-sets of $(L_i, \bar{L}_i)$ and $(R_i, \bar{R}_i)$.*

(iv) *For any pair i, j, $0 \leqslant i < j \leqslant r$, the contraction of L_i and R_j in $\mathcal{F}_S$ into two terminals gives $\mathcal{F}_{\{b_i, \ldots, b_j\}}$.*

By Lemma 4.9(iii), the strip $\mathcal{W}_{a,c}$ can be restored from $\mathcal{F}_S$. Observe that the sets $\{b_i, \ldots, b_j\}$, $i < j$, have the path structure similar to that of S. Therefore, by Lemma 4.9(iii) and (iv), the strips $\mathcal{W}_{b_i, b_j}$ can be restored from $\mathcal{F}_S$ as well. A unit which is not contracted to a terminal in the construction described in Lemma 4.9(iv) is said to be *distinguished* by $\mathcal{W}_{b_i, b_j}$.

The two nested families $\{L_i\}$ and $\{R_i\}$ define a cell structure on the set of units.

Namely, the *cell* Q^{ij}, $0 \leqslant i \leqslant j \leqslant r$, is defined as the intersection $(L_j \setminus L_{j-1}) \cap (R_i \setminus R_{i+1})$, where $L_{-1} = R_{r+1} = \varnothing$ (compare with the definition preceding the 3-Star Lemma). Observe that Q^{ii} is exactly the unit containing b_i. Part (iv) of Lemma 4.9 implies that the units of the same cell are distinguished by the same "partial" strips $\mathcal{W}_{b_p,b_q}$. More exactly, the following statement holds.

LEMMA 4.10. *The units of the cell Q^{ij} are distinguished by the strip $\mathcal{W}_{b_p,b_q}$ if and only if the paths $(b_i,\dots,b_j)$ and $(b_p,\dots,b_q)$ in the skeleton structure have at least one common edge.*

Proof. According to (iv) above, to obtain $\mathcal{W}_{b_p,b_q}$ we contract the cells Q^{ij} with $j \leqslant p$ or $i \geqslant q$, and the assertion follows. □

Edges between cells and, generally, mutual reachability of their units are subject to the following restriction.

LEMMA 4.11. *Let U_1 and U_2 be mutually reachable in $\mathcal{F}_S$, $U_l \in Q^{i_l,j_l}$, $l = 1, 2$, and $i_1 < i_2$. Then*

(i) $j_1 \leqslant j_2$;

(ii) *if U_1 is a stretched unit and U_3 lies in the same reachability cone of U_1 as U_2, $U_3 \in Q^{i_3,j_3}$, then $i_1 \leqslant i_3$ and $j_1 \leqslant j_3$.*

Proof. (i) Assume that $i_1 < i_2$, $j_2 < j_1$, and there exists a coherent path P_2 between U_1 and U_2. Evidently, U_1 is not a terminal. Let us extend P_2 behind U_1 up to a terminal U_{b_p}. W.l.o.g. we may assume that $p > i_1$ (the case $p < j_1$ is treated in the same way). Let us consider the S-mincut $(X, \bar{X})$ with $X = L_{j_1} \cap \bar{R}_{i_1+1}$. Evidently, the endpoints U_2 and U_{b_p} of the extended path lie outside this cut, while an intermediate unit U_1 lies inside this cut. So, the extended path intersects the transversal cut $(X, \bar{X})$ at least twice, a contradiction.

(ii) Let U_{b_p} be as above. From (i) we get immediately that either $p \geqslant j_1$ or $p \leqslant i_1$. If $p \geqslant j_1$, then we set $i = \min\{i_2, j_1\}$; now both U_{b_p} and U_2 belong to R_i, while U_1 does not (since $i_1 < i_2$), a contradiction. Hence, $p \leqslant i_1$.

Let us now extend a coherent path P_3 between U_1 and U_3 behind U_3 up to a terminal U_{b_q}. (Since U_3 is not necessarily a stretched unit, it may happen that $U_{b_q} = U_3$.) Observe that P_3 can be extended up to U_{b_p}; we denote the obtained coherent path by P_3'. Once more we get from (i) that either $q \geqslant j_1$ or $q \leqslant i_i$. If $q \leqslant i_1$, then the endpoints U_{b_q} and U_{b_p} of P_3' belong to L_{i_1}, while its intermediate unit U_1 does not, a contradiction. Hence, $q \geqslant j_1$.

Assume now that $i_3 < i_1$; in this case $U_3 \neq U_{b_q}$. Let us consider the part of P_3' between U_1 and U_{b_q}. Both its endpoints belong to R_{i_1}, while its intermediate unit U_3 does not, a contradiction.

Assume now that $i_3 = i_1$ and $j_3 < j_1$; again we have $U_3 \neq U_{b_q}$. Let us consider the same path as in the previous case. Both its endpoints do not belong to L_{j_3}, while its intermediate unit U_3 does, a contradiction.

Therefore, $i_1 = i_3$ implies $j_1 \leqslant j_3$. Finally, $i_1 < i_3$ implies the same inequality. □

Remark. Observe that the first assertion of the above lemma can be regarded as a generalization of Lemma 2.1(i).

Let us now extend the definition of the cell structure to the edges of $\mathcal{F}_S$. Let $e = (U_1, U_2)$ be an edge of $\mathcal{F}_S$, and assume that $U_1 \in Q^{i_1 j_1}$ and $U_2 \in Q^{i_2 j_2}$. We put $i = \min\{i_1, i_2\}$, $j = \max\{j_1, j_2\}$, and say that e belongs to the cell Q^{ij}. Observe that by Lemma 4.11 there are three types of edges belonging to a cell Q^{ij}: internal (both endpoints belong to Q^{ij}), external (one endpoint belongs to Q^{ij} and the other one either to $Q^{ij'}$, $i \leqslant j' < j$, or to $Q^{i'j}$, $i < i' \leqslant j$), and through (one endpoint belongs

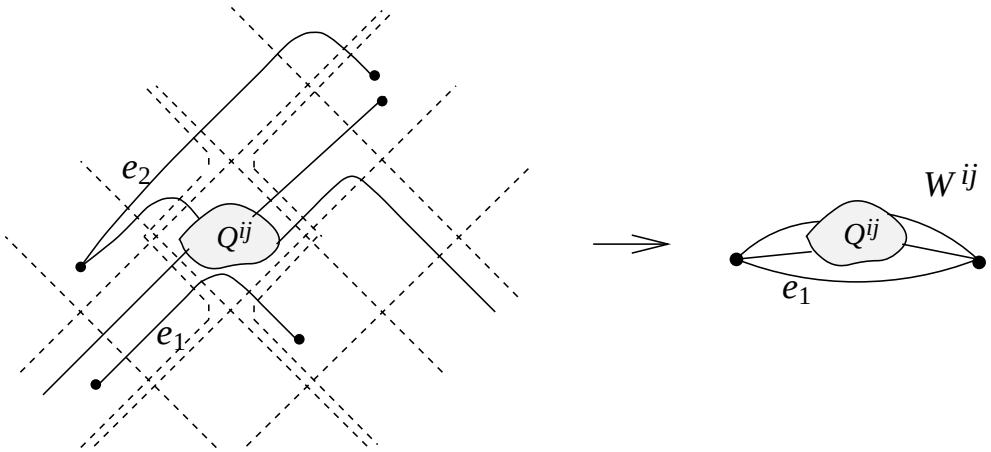


FIG. 4.14. *To the definition of the strip* $\mathcal{W}^{ij}$.

to $Q^{ij'}$, $i \leqslant j' < j$, and the other one to $Q^{i'j}$, $i < i' \leqslant j$); for an illustration, see Figure 4.13.

This definition can be justified as follows. Insert a dummy vertex u into the edge e, thus dividing it into e_1 and e_2. Evidently, if an S-mincut $(X, \bar{X})$ of the initial graph is crossed by e, then upon insertion of u it is replaced by two S-mincuts of the modified graph: $(X \cup \{u\}, \bar{X})$ and $(X, \bar{X} \cup \{u\})$. Conversely, let $(Y, \bar{Y})$, $u \in \bar{Y}$, be an S-mincut of the modified graph crossed, say, by e_1. Then $(Y \cup \{u\}, \bar{Y} \setminus \{u\})$ is another S-mincut of the modified graph, and it is crossed by e_2. Finally, if an S-mincut is not crossed by e, then it remains intact in the modified graph, and vice versa. It follows from the above discussion that the only difference between the initial $\mathcal{F}_S$ and the modified one is the new unit U that consists of the single dummy vertex u and belongs to the cell Q^{ij}; U, together with the incident edges $e_1 = (U_1, U)$ and $e_2 = (U_2, U)$, replaces the edge e in $\mathcal{F}_S$. This dummy unit, in a sense, represents the edge e, and thus justifies the above definition. Observe that this definition is stable with respect to insertion of dummy vertices. Namely, if e belongs to the cell Q^{ij}, then both e_1 and e_2 belong to the same cell.

For any pair i, j, $0 \leqslant i \leqslant j \leqslant r$, excluding cases $i = j = 0$ and $i = j = r$, we build from $\mathcal{F}_S$ a locally orientable graph $\mathcal{W}^{ij}$ with the set of nonterminals Q^{ij} (see Figure 4.14). We delete $\bigcup\{Q^{pq} : p < i \leqslant j < q\}$ and $\bigcup\{Q^{pq} : i < p \leqslant q < j\}$. (Observe that by Lemma 2.1 there are no edges between Q^{ij} and the deleted sets.) Next, we contract the sets $\bigcup\{Q^{pq} : p \leqslant i,\ q \leqslant j,\ (p,q) \neq (i,j)\}$ and $\bigcup\{Q^{pq} : p \geqslant i,\ q \geqslant j,\ (p,q) \neq (i,j)\}$ into two terminals a' and c'. Finally, we delete the edges between a' and c' that do not belong to Q^{ij}. Observe that this definition is consistent with the definition of $\mathcal{W}^{ac}$ given in section 4.3 ($\mathcal{W}^{02}$, in our new notation), because all the edges between a' and c' in $\mathcal{F}_{\{a,b,c\}}$ belong to Q^{02}. Evidently, all the edges that belong to Q^{ij} participate in $\mathcal{W}^{ij}$. Besides, each external edge participates in one more such graph, and each through edge in two more such graphs. Extensions of cuts of $\mathcal{W}^{ij}$ to S-mincuts are defined similarly to the case analyzed in section 4.3. The following statement is an analogue of Lemmas 4.2 and 4.8.

LEMMA 4.12. (i) *The graph* $\mathcal{W}^{ij}$ *is a strip of width* λ^{ij}; *if* $j = i$, *then* $\lambda^{ij} > \lambda_S/2$, *while if* $j \geqslant i + 2$, *then* $\lambda^{ij} < \lambda_S/2$.

(ii) *All its transversal cuts, and only they, are extendible to* S*-mincuts.*

(iii) *For any sequence* $i_1 < i_2 < \cdots < i_l \leqslant j_l < \cdots < j_2 < j_1$, *any set of* l *transversal cuts of* $\mathcal{W}^{i_1j_1}, \ldots, \mathcal{W}^{i_lj_l}$ *has a mutual extension to an* S*-cut. Moreover, if* $j_l \geqslant i_l + 2$, *then* $\lambda^{i_1j_1} + \cdots + \lambda^{i_lj_l} < \lambda_S/2$.

(iv) *Let* $i_1 < i_2 < \cdots < i_l \leqslant j_l < \cdots < j_2 < j_1$ *and let a* 2*-partition of* $V \setminus \{Q^{i_1j_1} \cup \cdots \cup Q^{i_lj_l}\}$ *enter a mutual extension of a set of transversal cuts of* $\mathcal{W}^{i_1j_1}, \ldots, W^{i_lj_l}$. *Then it enters some mutual extension of any set of such cuts.* (*Informally, all sets of transversal cuts for such a sequence of strips are interchangeable in* S*-mincuts.*)

Proof. (i) The fact that $\mathcal{W}^{ij}$ is a strip is proved similarly to Lemma 4.2(i) taking into account that the sets $X_a = \bigcup\{Q^{pq} : p \leqslant i,\ q \leqslant j,\ (p,q) \neq (i,j)\}$ and $\bar{X}_c = \bigcup\{Q^{pq} : p \geqslant i,\ q \geqslant j,\ (p,q) \neq (i,j)\}$ define transversal cuts $\mathcal{C}_a = (X_a, \bar{X}_a)$ and $\mathcal{C}_c = (X_c, \bar{X}_c)$ (shown by a double line on Figure 4.14). The inequality for the case $j = i$ follows from the fact that the cardinality of the cut $(U_{b_i}, V \setminus U_{b_i})$ exceeds λ_S. In the case $j \geqslant i + 2$ we see that all the edges of $\mathcal{F}_S$ incident to a'-type terminals in the strips $\mathcal{W}^{ij}$ and $\mathcal{W}^{i+1,i+1}$ belong to the edge-set of $\mathcal{C}_a$. Therefore, $\lambda^{ij} + \lambda^{i+1,i+1} \leqslant c(\mathcal{C}_a) = \lambda_S$. Since $\lambda^{i+1,i+1} > \lambda_S/2$, we are done.

(ii) Let X_a and $\bar{X}_c$ be defined as above. Given any transversal cut $\mathcal{C}' = (a' \cup Z, \bar{Z} \cup c')$ of $\mathcal{W}^{ij}$, let us consider the edge-set of the 2-partition $\mathcal{C} = (X_a \cup Z, \bar{Z} \cup \bar{X}_a)$ of V. As compared with the edge-set of $\mathcal{C}_a$, it loses the edges between X_a and Z and gets the edges between Z and $\bar{Z} \cup \bar{X}_c$. However, the same holds for the cut $\mathcal{C}'$ as compared with the cut of $\mathcal{W}^{ij}$ that separates a'. Since, by (i), the cuts in the second pair have equal cardinalities, the same is true for the first pair. Thus $\mathcal{C}$ is a minimum S-cut.

Conversely, let $(X, \bar{X})$ be an S-mincut separating X_a from $\bar{X}_c$. By the same reasons as above, it generates a 2-partition of cardinality λ^{ij} in $\mathcal{W}^{ij}$ that is a minimum (i.e., transversal) cut of $\mathcal{W}^{ij}$.

(iii) Let $\mathcal{C}'_k$ be a transversal cut of $\mathcal{W}^{i_kj_k}$, $1 \leqslant k \leqslant l$. Denote by $\mathcal{C}_k$ the extension of $\mathcal{C}'_k$ defined as in the proof of (ii) and consider the S-mincut $\mathcal{C} = \mathcal{C}_1 \cup \cdots \cup \mathcal{C}_l$. Inequalities $i_1 < i_2 < \cdots < i_l \leqslant j_l < \cdots < j_2 < j_1$ guarantee that $\mathcal{C}$ is an extension for any of the $\mathcal{C}'_k$ and hence is a mutual extension of the whole set $\{\mathcal{C}'_1, \ldots, \mathcal{C}'_k\}$. The same inequalities guarantee that no edge is counted more than once in the sum $\lambda^{i_1j_1} + \cdots + \lambda^{i_lj_l}$; the rest of the proof of the inequality $\lambda^{i_1j_1} + \cdots + \lambda^{i_lj_l} < \lambda_S/2$ follows the proof of (i).

(iv) Assume that $l = 1$. Let $\mathcal{C}_0$ be an extension of some transversal cut of $\mathcal{W}^{i_1j_1}$ and let $\mathcal{C}'$ be an arbitrary transversal cut of $\mathcal{W}^{i_1j_1}$. Take $\bar{X}_c$ as above and put $X^*_c = X_c \setminus Q^{i_1j_1}$; then $\mathcal{C}^* = (X^*_c, \bar{X}^*_c)$ is an S-mincut. Now define the cut $\mathcal{C}$ of G as in the proof of (ii); then $(\mathcal{C}_0 \cap \mathcal{C}^*) \cup \mathcal{C}$ is an extension of $\mathcal{C}'$ that divides $V \setminus Q^{i_1j_1}$ in the same way as $\mathcal{C}_0$. For $l > 1$ the proof applies with evident minor changes. □

Dynamics of $\mathcal{F}_S$, $S = \{a = b_0, \ldots, b_r = c\}$, in the case of the path structure are induced by those of $\mathcal{F}_{\{a,c\}} = \mathcal{W}_{a,c}$, in the same way as it was done for $\mathcal{F}_{\{a,b,c\}}$ in the asymmetric subcase (see section 4.3). To be more precise, let $U_1 \in Q^{i_1j_1}$, $U_2 \in Q^{i_2j_2}$ (we may assume w.l.o.g. that $j_1 \leqslant j_2$); we call i_k, j_k the *coordinates* of U_k, $k = 1, 2$. Certain elements of S "fall inside" U^{new} when $j_1 \leqslant i_2$; in this case U^{new} is a terminal and it contains all the b_k such that $j_1 \leqslant k \leqslant i_2$ (see Figure 4.15(a)).

Concerning the dynamics of the sets L_k (those of R_k are similar), Lemma 4.3 can be generalized as follows.

LEMMA 4.13. *The sets* L_k, $j_1 \leqslant k < j_2$, *and only they, are extended* (*at least by* U_2). *The new part of* $\widehat{L}_k$ *is the union of the units* $U' \in Q^{i'j'}$ *such that* $j' > k$ *and* $U' \in \mathcal{R}_a(U_2)$.

The proof of Lemma 4.13 is similar to that of Lemma 4.3 (see Figure 4.15(b)).

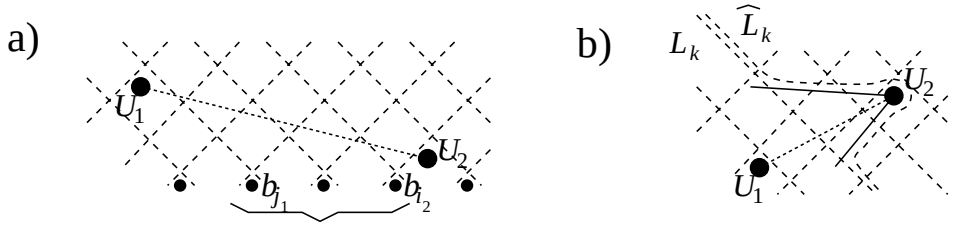


FIG. 4.15. *Dynamics in the case of the path structure.*

In the case when $U^{\rm new}$ is not a terminal, its new cell and the new cells of the other units can be defined via the characterization of the sets $\widehat{L}_k$, $\widehat{R}_l$ given in Lemma 4.13. We use the following notation. Let $j_{\min} = \min\{j_1, j_2\}$ and $i_{\max} = \max\{i_1, i_2\}$. The unit $U(j)$ is the one of U_1, U_2 with the j-coordinate not less than that of the other one, and $U(i)$ is the one of them with the i-coordinate not greater than that of the other one.

COROLLARY 4.14. (i) *The new unit $U^{\rm new}$ belongs to the cell $\widehat{Q}^{i_{\max} j_{\min}}$.*

(ii) *Let a unit $U' \in Q^{ij}$ move to the cell $\widehat{Q}^{i^* j^*}$.*

If $j > j_{\min}$ and $U' \in \mathcal{R}_a(U(j))$, then $j^* = j_{\min}$, otherwise $j^* = j$.

If $i < i_{\max}$ and $U' \in \mathcal{R}_c(U(i))$, then $i^* = i_{\max}$, otherwise $i^* = i$.

Moreover, an analogue of Corollary 4.14 is valid also in the case when $U^{\rm new}$ is a terminal if we previously merge the cells with the identified indices in the range $[j_1, \ldots, i_2]$.

5. The flesh and the skeleton of the connectivity carcass.

5.1. The system of units, or the flesh of the connectivity carcass. Let us consider an arbitrary bunch $\mathcal{B}$ of S-cuts, and let $(S_{\mathcal{B}}, \bar{S}_{\mathcal{B}})$ be the corresponding partition of S (see section 2 for all the necessary definitions). In fact, $\mathcal{B}$ is the set of all minimum $(S_{\mathcal{B}}, \bar{S}_{\mathcal{B}})$-cuts; thus the structure of $\mathcal{B}$ can be represented by the strip $\mathcal{W}_{S_{\mathcal{B}}, \bar{S}_{\mathcal{B}}}$. For brevity, in what follows this strip is denoted by $\mathcal{W}_{\mathcal{B}}$, the corresponding quotient mapping by $\delta_{\mathcal{B}}$, and the units of $\mathcal{W}_{\mathcal{B}}$ are called $\mathcal{B}$-units. Therefore, the following statement holds.

FACT 5.1. *Any bunch of S-cuts $\mathcal{B}$ is represented by the strip $\mathcal{W}_{\mathcal{B}}$ and the mapping $\delta_{\mathcal{B}}$ so that the $\delta_{\mathcal{B}}$-inducing provides a bijection between the transversal cuts of $\mathcal{W}_{\mathcal{B}}$ and the S-cuts in $\mathcal{B}$.*

Evidently, for the bunch $\bar{\mathcal{B}}$ of cuts opposite to cuts in $\mathcal{B}$, the strip $\mathcal{W}_{\bar{\mathcal{B}}}$ coincides with $\mathcal{W}_{\mathcal{B}}$.

By the definition, the partition of V into S-units is a refinement of its partition into $\mathcal{B}$-units. The following statement shows that only the two terminal $\mathcal{B}$-units can be refined.

LEMMA 5.2. *Each nonterminal $\mathcal{B}$-unit is an S-unit.*

Proof. The proof is similar to that of Lemma 4.4(i). Let U be a nonterminal $\mathcal{B}$-unit subdivided by an S-mincut $\mathcal{C}$. Evidently, there exist vertices $x \in S_{\mathcal{B}}$ and $y \in \bar{S}_{\mathcal{B}}$ separated by $\mathcal{C}$; assume w.l.o.g. that x lies inside $\mathcal{C}$. By the definition of $\mathcal{B}$-units, the bunch $\mathcal{B}$ contains cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ separating $S_{\mathcal{B}}$ from U and U from $\bar{S}_{\mathcal{B}}$, respectively. Observe that all the three cuts $\mathcal{C}$, $\mathcal{C}_1$, and $\mathcal{C}_2$ are minimum (x, y)-cuts. Let us consider the cut $\mathcal{C}_1 \cup (\mathcal{C} \cap \mathcal{C}_2)$. It contains $S_{\mathcal{B}}$ inside, contains $\bar{S}_{\mathcal{B}}$ outside, and thus belongs to $\mathcal{B}$. However, it subdivides U, a contradiction. ∎

In what follows we say that the S-units described in Lemma 5.2 are *distinguished* by the bunch $\mathcal{B}$ or by the bunch strip $\mathcal{W}_{\mathcal{B}}$ (compare with the definition preceding Lemma 4.10).

By the definition, the star of any nonterminal vertex of the locally orientable graph $\mathcal{W}_{\mathcal{B}}$ is partitioned into two sets of equal cardinality. Thus, by Lemma 5.2, the star of the corresponding vertex of $\mathcal{F}_S$ is halved. The crucial observation is that the arising 2-partition does not depend on the choice of $\mathcal{B}$ (see Lemma 4.4(iii) for a particular case of the same statement).

LEMMA 5.3. *If an S-unit is distinguished by several bunch strips, then the 2-partitions of its star in all these strips coincide.*

Proof. Let U be an S-unit distinguished by bunches $\mathcal{B}$ and $\mathcal{B}'$. W.l.o.g., the 2-partition $(S_{\mathcal{B}'}, \bar{S}_{\mathcal{B}'})$ divides $S_{\mathcal{B}}$. Therefore, for any $x \in \bar{S}_{\mathcal{B}}$ one can choose $y \in S_{\mathcal{B}}$ separated from x by this 2-partition. Similarly to the proof of Lemma 4.4(iii), both strips $\mathcal{W}_{\mathcal{B}}$ and $\mathcal{W}_{\mathcal{B}'}$ are contractions of the same strip $\mathcal{W}_{x,y}$. (In this case Lemma 3.8 is used repeatedly.) Hence, the 2-partitions at U in these strips are inherited from $\mathcal{W}_{x,y}$ and thus coincide. □

Let us provide $\mathcal{F}_S$ with a canonical structure of a locally orientable graph (see definitions in section 3). To the star of any S-unit distinguished by at least one bunch strip we assign the partition into sides indicated in Lemma 5.3. To the star of any other unit we assign the trivial 2-partition. These 2-partitions are said to be *inherent*. The quotient graph $\mathcal{F}_S$ endowed with the structure of a locally oriented graph as described above is said to be the *flesh* of the connectivity carcass of S.

As for general locally orientable graphs, the units of the former type are called stretched, and those of the latter type, terminals. In addition, a unit is called *heavy* if it intersects S. Observe that only terminal units (but, in general, not all of them) can be heavy. Below we prove several basic properties of $\mathcal{F}_S$.

In what follows we illustrate certain general concepts on the graph shown in Figure 5.1(a). Large black circles denote vertices in S; several S-mincuts are shown by dashed lines; nontrivial S-units are encircled by dotted lines. The corresponding flesh is presented in Figure 5.1(b).

According to the definitions above, $\mathcal{F}_S$ is obtained by "gluing together" all the bunch strips. Conversely, each bunch strip can be reconstructed from $\mathcal{F}_S$ with the help of contractions. Indeed, let $\mathcal{B}$ be an arbitrary bunch of S-mincuts. The intersection of all cuts in $\mathcal{B}$ is called the *tight* cut of $\mathcal{B}$; the union of all cuts in $\mathcal{B}$ is called the *loose* cut of $\mathcal{B}$. (Evidently, each of them separates a terminal of $\mathcal{W}_{\mathcal{B}}$ from all other units.) Observe that the tight cut of $\mathcal{B}$ is just the $S_{\mathcal{B}}$-tight $(S_{\mathcal{B}}, \bar{S}_{\mathcal{B}})$-mincut (for the definition, see section 2 immediately after Fact 2.2), while the loose cut of $\mathcal{B}$ is the opposite of the $\bar{S}_{\mathcal{B}}$-tight $(\bar{S}_{\mathcal{B}}, S_{\mathcal{B}})$-mincut. The following statement is a generalization of Lemma 4.1(iii) and Lemma 4.4(i).

LEMMA 5.4. *For any bunch $\mathcal{B}$, the strip $\mathcal{W}_{\mathcal{B}}$ is obtained from $\mathcal{F}_S$ by contracting all the units inside the tight cut of $\mathcal{B}$ and all the units outside the loose cut of $\mathcal{B}$ into terminals.*

Proof. The proof follows immediately from Lemmas 5.2 and 5.3 (for details, see the proof of Lemma 4.4(i)). □

Parts (i) and (ii) of the next statement provide a generalization of Lemma 4.5.

THEOREM 5.5. (i) *The flesh $\mathcal{F}_S$ is a balanced acyclic locally orientable graph.*

(ii) *Each S-mincut is transversal in $\mathcal{F}_S$.*

(iii) *Let $\tilde{n}$ and $\tilde{m}$ be the numbers of vertices and edges of $\mathcal{F}_S$, respectively. Then $\tilde{n} \leqslant n$, $\tilde{m} \leqslant m$, $\tilde{m} = O(\lambda_S \tilde{n})$.*

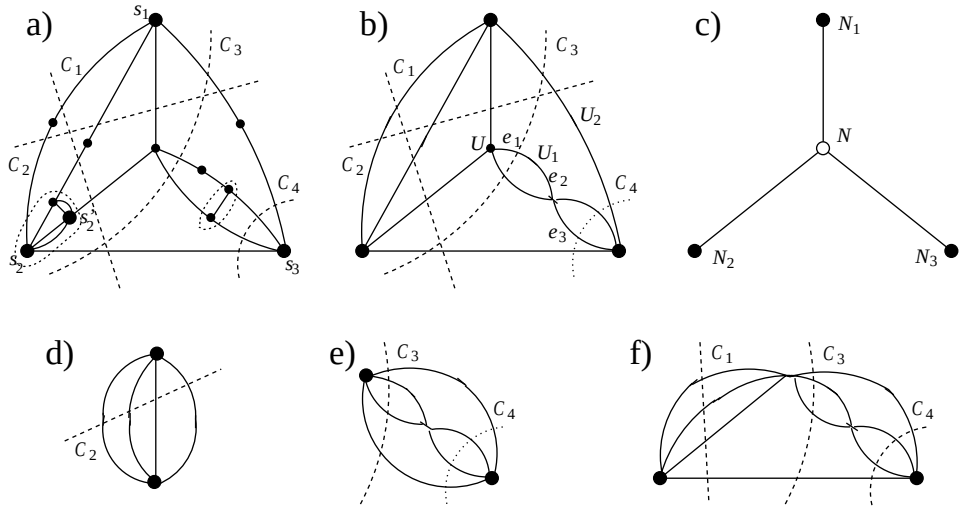


FIG. 5.1. *Connectivity carcass of a graph and its contractions to strips.*

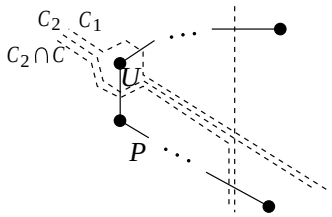


FIG. 5.2. *To the proof of Theorem* 5.5.

Proof. (i) The fact that $\mathcal{F}_S$ is balanced is clear, since inherent partitions at stretched units of $\mathcal{F}_S$ come from those in bunch strips (see Lemma 5.3). Acyclicity of $\mathcal{F}_S$ follows from part (ii) of this theorem exactly in the same way as in the proof of Lemma 4.5.

(ii) Assume to the contrary that the claim is violated by some pair G, S and the corresponding flesh $\mathcal{F}_S$. Then there exists a minimal counterexample, namely, an S-mincut $\mathcal{C}$ and a coherent path P in $\mathcal{F}_S$ such that P intersects the edge-set of $\mathcal{C}$ at least twice and the number of units in P is minimal among all such pairs $\mathcal{C}, P$ (see Figure 5.2). By minimality, the endpoints of P lie outside $\mathcal{C}$, and all the inner units of P lie inside $\mathcal{C}$ (up to flipping of $\mathcal{C}$). Let U be an arbitrary inner unit of P; since P is a coherent path, U is a stretched unit. Therefore, there exists a bunch that distinguishes U. Let $\mathcal{C}_1$ be the S-mincut in this bunch defined by a cone $\mathcal{R}$ of U in the corresponding strip and $\mathcal{C}_2$ be the S-mincut in this bunch defined by $\mathcal{R} \setminus U$. Up to taking the other cone of U, we may assume that both pairs $\mathcal{C}_1, \mathcal{C}$ and $\mathcal{C}_2, \mathcal{C}$ satisfy the conditions of Lemma 2.1. Thus, by Fact 2.2, both $\mathcal{C}_1 \cap \mathcal{C}$ and $\mathcal{C}_2 \cap \mathcal{C}$ are S-mincuts; moreover, they belong to the same bunch and their inner parts differ exactly by U.

Let us traverse P from an endpoint up to U and take the edge entering U and the

next edge of P. By Fact 3.3, exactly one of the above edges intersects the edge-set of $\mathcal{C}_2 \cap \mathcal{C}$; let U' be the other endpoint of this edge. Since U' lies inside $\mathcal{C}_2 \cap \mathcal{C}$, it lies inside $\mathcal{C}$ as well, and thus cannot be an endpoint of P. Let P' be the subpath of P leaving U by the edge (U, U') and ending at the corresponding endpoint of P. Since both U and this endpoint of P lie outside $\mathcal{C}_2 \cap \mathcal{C}$, the pair $\mathcal{C}_2 \cap \mathcal{C}, P'$ contradicts the minimality of the pair $\mathcal{C}, P$.

(iii) The first two inequalities are evident. Let us show that $\tilde{m} \leqslant \lambda_S(\tilde{n}-1)$. Since any two units are separated by an S-mincut, the size of a minimum cut between them in $\mathcal{F}_S$ is at most λ_S. Let us consider the Gomory–Hu tree of $\mathcal{F}_S$ (see [GH]). Recall that its vertices correspond bijectively to the units, and thus any of its edges defines a cut of $\mathcal{F}_S$. By [GH], this cut is a minimum cut between some two units, and hence its size is at most λ_S. Since there are $\tilde{n}-1$ such cuts and each edge of $\mathcal{F}_S$ takes part in at least one of them, their total number $\tilde{m}$ does not exceed $\lambda_S(\tilde{n}-1)$, as required. □

The flesh can be considered, in a sense, as a generalization of a strip. By the way, at early stages we called our structure "branching dag." The meaning of such a nickname will become more clear in the next subsection (see also section 4.4).

5.2. The system of bunches, or the skeleton of the connectivity carcass: Odd case. For the case $S = V$, the S-mincuts are the (globally) minimum cuts of G, and any bunch consists of exactly one S-mincut. Recall (see the introduction) that the structure of all minimum cuts of G is represented by a cactus tree (see [DKL]); if the graph connectivity is odd, it is, in fact, a tree. The construction is based on the Crossing Lemma [Bi, DKL]. A generalization of this lemma to the S-mincuts and the existence of a cactus tree representation of all 2-partitions of S by S-mincuts were stated (without proof) independently in [Na, W93] (see also [DV1]). The generalization of the lemma is straightforward (see below); for a proof of the existence of a cactus tree representation, see [DN, Nu].

As usual, two 2-partitions of a set are called crossing if they divide the set into four nonvoid parts. We consider a more delicate notion: two S-cuts $\mathcal{C}$ and $\mathcal{C}'$ (which are 2-partitions of V) are said to be S-crossing if they divide S into four nonvoid parts.

S-Crossing Lemma. *Let $\mathcal{C}$ and $\mathcal{C}'$ be S-crossing S-mincuts; then*

(i) $c(V_{\mathcal{C}} \cap V_{\mathcal{C}'}, V_{\mathcal{C}} \cap \bar{V}_{\mathcal{C}'}) = c(V_{\mathcal{C}} \cap \bar{V}_{\mathcal{C}'}, \bar{V}_{\mathcal{C}} \cap \bar{V}_{\mathcal{C}'}) = c(\bar{V}_{\mathcal{C}} \cap \bar{V}_{\mathcal{C}'}, \bar{V}_{\mathcal{C}} \cap V_{\mathcal{C}'}) = c(\bar{V}_{\mathcal{C}} \cap V_{\mathcal{C}'}, V_{\mathcal{C}} \cap V_{\mathcal{C}'}) = \lambda_S/2$;

(ii) $c(V_{\mathcal{C}} \cap V_{\mathcal{C}'}, \bar{V}_{\mathcal{C}} \cap \bar{V}_{\mathcal{C}'}) = c(V_{\mathcal{C}} \cap \bar{V}_{\mathcal{C}'}, \bar{V}_{\mathcal{C}} \cap V_{\mathcal{C}'}) = 0$.

Hence, the 2-partitions $\mathcal{C} \cap \mathcal{C}'$, $\mathcal{C} \cap \bar{\mathcal{C}}'$, $\bar{\mathcal{C}} \cap \bar{\mathcal{C}}'$, $\bar{\mathcal{C}} \cap \mathcal{C}'$ are S-mincuts as well.

The proof coincides with that of the Crossing Lemma, with an additional observation that in our case all 2-partitions involved are actually S-cuts.

It follows immediately from part (i) of the S-Crossing Lemma that for λ_S odd there are no S-crossing S-mincuts. Let us show that whenever this S-crossing-free property holds, we can represent the system of bunches of S-mincuts by a tree (for an example, see Figure 5.1(c)). Recall that each bunch is uniquely characterized by the corresponding 2-partition of S; let us concentrate on these 2-partitions. Observe that the S-crossing-free property for S-mincuts is equivalent to the crossing-free property for these 2-partitions of S (i.e., that no two such 2-partitions are crossing).

Theorem 5.6. *For any set X and any crossing free family F of 2-partitions of X,*

(i) *there exist a unique tree $\mathcal{H}$ with the node set $\mathcal{N}$ and a mapping $\varphi : X \to \mathcal{N}$ such that φ-inducing provides a bijection between the cuts of $\mathcal{H}$ and the 2-partitions*

in F;

(ii) *the size of $\mathcal{H}$ is linear in the number of subsets that X is cut into by the members of F.*

This statement can be considered as folklore; however, for a proof see [DW, section 4.1].

COROLLARY 5.7. *For any $G = (V, E)$ and any $S \subseteq V$ with the S-crossing-free property the system of bunches of S-mincuts is represented by a tree, which we denote by $\mathcal{H}_S$; its node set is denoted by $\mathcal{N}_S$. The corresponding mapping $S \to \mathcal{N}_S$ is denoted by φ_S; the bunches of S-mincuts correspond bijectively to the cuts of $\mathcal{H}_S$ via φ_S-inducing. Moreover, the cardinality of $\mathcal{N}_S$ is linear in the number of $(\lambda_S + 1)$-connectivity classes in S.*

The tree $\mathcal{H}_S$ is said to be the *skeleton* of the connectivity carcass of the set S. Its edges are called *structural edges* (to distinguish them from the edges of the graph G). Any cut of $\mathcal{H}_S$ is defined by a single structural edge.

Observe that the image $\varphi_S(S)$ does not necessarily coincide with $\mathcal{N}_\mathcal{S}$. The nodes N for which $\varphi_S^{-1}(N) = \varnothing$ are called *empty* nodes (e.g., node N on Figure 5.1(c)). It follows from the construction that the degree of each empty node is at least three. It is easy to see that the preimages of the nonempty nodes in $\mathcal{N}_\mathcal{S}$ are exactly the classes of $(\lambda_S + 1)$-connectivity in S.

By Corollary 5.7, for any S with λ_S odd, the system of bunches of S-mincuts is represented by the skeleton $\mathcal{H}_S$, which is a tree. For λ_S even, this is also the case if the S-crossing-free property holds. In general, analogues of Theorem 5.6 and Corollary 5.7 are valid, in which the tree is replaced by a cactus tree. It means that each bunch is represented by a cut of this cactus tree, which is defined either by a structural edge not belonging to any cycle, or by a pair of structural edges belonging to the same cycle. For details see [DV1].

In the rest of the paper we assume that the skeleton is a tree, thus completely covering the odd case.

6. Structure and properties of the connectivity carcass.

6.1. Tight cuts. Let ε be an arbitrary structural edge of $\mathcal{H}_S$ and N be one of its endpoints. By Theorem 5.6, this edge defines two mutually opposite bunches of S-mincuts; these two bunches are represented by a strip (see Fact 5.1), which we denote $\mathcal{W}(\varepsilon)$. (For example, Figures 5.1(d) and (e) show the strips $\mathcal{W}(N, N_1)$ and $\mathcal{W}(N, N_3)$, respectively.) Consider the cut of $\mathcal{H}_S$ corresponding to ε and containing N inside. The intersection of all S-mincuts of the bunch represented by this cut of $\mathcal{H}_S$ is the tight cut of this bunch. This cut is said to be the *N-tight cut in direction ε* and is denoted by $\mathcal{C}(N, \varepsilon)$ (e.g., the cut $\mathcal{C}_3$ on Figure 5.1(a) is the N-tight cut in direction (N, N_1)).

A *path* in $\mathcal{H}_S$ is an alternating sequence of distinct nodes and structural edges; we consider the opposite sequence as the same path traversed in the opposite direction. The unique path between two arbitrary nodes M, N of the skeleton is denoted by $[M, N]$. For any two structural edges ε' and ε'' there exists a unique path that starts by one of them and ends by the other one; we denote this path by $[\varepsilon', \varepsilon'']$.

It turns out that tight cuts are monotonous along paths in the skeleton. More precise, the following generalization of Lemma 4.9(i) holds; it makes use of the domination relation $\prec$ between cuts defined at the beginning of section 2.

LEMMA 6.1. *For any path $P = (N_0, \varepsilon_1, N_1, \ldots, \varepsilon_r, N_r)$ in the skeleton $\mathcal{H}_S$, $r \geqslant 2$, one has $\mathcal{C}(N_0, \varepsilon_1) \prec \mathcal{C}(N_{r-1}, \varepsilon_r)$.*

Proof. Observe that the sets $S \cap V_{\mathcal{C}(N_0,\varepsilon_1)}$ and $S \cap V_{\mathcal{C}(N_{r-1},\varepsilon_r)}$ are nonempty, since the cuts in question are S-cuts. Moreover, $S \cap V_{\mathcal{C}(N_0,\varepsilon_1)}$ is contained in $S \cap V_{\mathcal{C}(N_{r-1},\varepsilon_r)}$, since both sets are φ_S-preimages of subtrees of $\mathcal{H}_S$ and the subtree of the former is contained in that of the latter; the set inclusion is strict since the partitions of S by distinct bunches are distinct.

Let a and b be arbitrary vertices in $S \cap V_{\mathcal{C}(N_0,\varepsilon_1)}$ and $S \cap \bar{V}_{\mathcal{C}(N_r,\varepsilon_r)}$, respectively. Both $\mathcal{C}(N_0,\varepsilon_1)$ and $\mathcal{C}(N_{r-1},\varepsilon_r)$ are minimum (a,b)-cuts, so their intersection $\mathcal{C}$ is a minimum (a,b)-cut as well. By the set inclusion proved above, the cut $\mathcal{C}$ belongs to the bunch defined by ε_1, and thus coincides with $\mathcal{C}(N_0,\varepsilon_1)$, since the latter cut is tight. This coincidence, together with the strict inclusion $S \cap V_{\mathcal{C}(N_0,\varepsilon_1)} \subset S \cap V_{\mathcal{C}(N_{r-1},\varepsilon_r)}$, implies the assertion of the lemma. □

COROLLARY 6.2. *For any path $P = (N_0, \varepsilon_1, N_1, \ldots, \varepsilon_r, N_r)$ in the skeleton $\mathcal{H}_S$, $r \geqslant 2$, one has $V_{\mathcal{C}(N_0,\varepsilon_1)} \cap V_{\mathcal{C}(N_r,\varepsilon_r)} = \varnothing$ and $V_{\mathcal{C}(N_1,\varepsilon_1)} \not\subseteq V_{\mathcal{C}(N_{r-1},\varepsilon_r)}$.*

Remark. Let ε' and ε'' be two arbitrary structural edges and $\mathcal{C}'$ and $\mathcal{C}''$ be two tight cuts in directions ε' and ε'', respectively. Observe that Lemma 6.1 and Corollary 6.2 cover all the possible choices for $\mathcal{C}'$ and $\mathcal{C}''$.

6.2. Reconstruction of pairwise strips. According to Lemma 5.4, one can obtain the strip representation for any bunch of S-mincuts just by a contraction of $\mathcal{F}_S$. This fact holds also in a more general situation, when one is interested in the strip representation $\mathcal{W}_{s_1,s_2}$ for an arbitrary pair of vertices $s_1, s_2 \in S$, or, more generally, in the strip $\mathcal{W}_{S_1,S_2}$ for an arbitrary pair of disjoint subsets $S_1, S_2 \subset S$, provided the cardinality of (S_1,S_2)-mincuts equals λ_S. Observe that in this case the (S_1,S_2)-mincuts, which are represented by $\mathcal{W}_{S_1,S_2}$, are exactly the S-mincuts separating S_1 from S_2. In this subsection we describe the contractions of $\mathcal{F}_S$ that give the corresponding strips. The reader can use for illustration Figure 5.1(d) ($S_1 = \{s_1\}$, $S_2 = \{s_2, s_2', s_3\}$), Figure 5.1(e) ($S_1 = \{s_3\}$, $S_2 = \{s_1, s_2\}$), Figure 5.1(f) ($S_1 = \{s_2\}$, $S_2 = \{s_3\}$).

Let T_1 and T_2 be two edge-disjoint subtrees of $\mathcal{H}_S$. The *link* of T_1 and T_2 is defined as the unique path with one endpoint in T_1 and the other one in T_2 and having no edges in common with T_1 and T_2. We denote the link of T_1 and T_2 by $\mathcal{L}(T_1,T_2)$. Observe that the link of two distinct nodes M and N is just the path $[M,N]$, the link of two distinct edges ε' and ε'' extended by these edges themselves gives exactly the path $[\varepsilon',\varepsilon'']$, and the link of two subtrees intersecting by a node is this very node.

Let S_1 and S_2 be disjoint subsets of S. By Theorem 5.6, a bunch of S-mincuts separates S_1 from S_2 if and only if the deletion of the corresponding edge of $\mathcal{H}_S$ produces two subtrees containing $\varphi_S(S_1)$ and $\varphi_S(S_2)$, respectively. Let $\mathcal{T}(S_1)$ and $\mathcal{T}(S_2)$ be the minimal subtrees of the tree $\mathcal{H}_S$ containing $\varphi_S(S_1)$ and $\varphi_S(S_2)$, respectively. For brevity, we write $\mathcal{L}(S_1,S_2)$ instead of $\mathcal{L}(\mathcal{T}(S_1),\mathcal{T}(S_2))$. The following statement follows easily from Theorem 5.6 and Lemma 6.1.

FACT 6.3. (i) *If $\mathcal{T}(S_1) \cap \mathcal{T}(S_2) \neq \varnothing$, then there are no S-mincuts separating S_1 from S_2.*

(ii) *Otherwise, the edges of $\mathcal{H}_S$ corresponding to the bunches of S-mincuts separating S_1 from S_2 form the link $\mathcal{L}(S_1,S_2)$.*

(iii) *Under the assumptions of* (ii), *let N_1 be the endpoint of $\mathcal{L}(S_1,S_2)$ lying in $\mathcal{T}(S_1)$ and ε_1 be the edge of $\mathcal{L}(S_1,S_2)$ incident to N_1. Then the S_1-tight (S_1,S_2)-mincut coincides with the N_1-tight cut in direction ε_1.*

Now we can prove the main result of this subsection, which is a generalization of Lemma 4.1(i),(ii), Lemma 4.7, and Lemma 4.9 (ii)–(iv). For an illustration, see Figure 6.1.

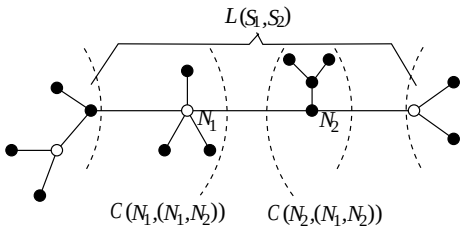


FIG. 6.1. *To the proof of Theorem* 6.4.

THEOREM 6.4. *Let S_1 and S_2 be two subsets of S such that $\mathcal{T}(S_1) \cap \mathcal{T}(S_2) = \varnothing$. The strip $\mathcal{W}_{S_1,S_2}$ is obtained as follows.*

The underlying graph of $\mathcal{W}_{S_1,S_2}$ is obtained from $\mathcal{F}_S$ by contractions of certain subsets of units to a single heavy unit. These subsets correspond bijectively to the nodes of the link $\mathcal{L}(S_1, S_2)$. The subset U_N that corresponds to a node N is equal to the intersection of the inner sides of N-tight cuts in directions of the edges of $\mathcal{L}(S_1, S_2)$ incident to N.

The inherent partition at any noncontracted unit remains the same. The inherent partition at a new unit U_N is induced by the edge-sets of the N-tight cuts defining U_N.

Proof. Let us delete all the edges of the path $\mathcal{L}(S_1, S_2)$. The trees of the forest thus obtained correspond bijectively to the nodes of $\mathcal{L}(S_1, S_2)$: a node N corresponds to the tree $\mathcal{T}_N$ that contains N. Denote by Σ_N the inverse image of the set of all the nodes of $\mathcal{T}_N$ under the mapping φ_S. Observe that Σ_N is a nonempty subset of S. Indeed, if N is an endpoint of the link, then Σ_N is just the intersection of a side of a certain S-mincut with S and is thus nonempty. If N is an inner node of the link, then $\Sigma_N = \varnothing$ would imply that the cuts of $\mathcal{H}_S$ defined by the two structural edges incident to N φ_S-induce the same 2-partition of S, in contradiction to Theorem 5.6 (compare with the proof of Lemma 6.1). Besides, by Fact 6.3(ii), the S-mincuts separating S_1 from S_2 do not divide Σ_N. Therefore, in any case all the set Σ_N lies in a single heavy unit of the strip $\mathcal{W}_{S_1,S_2}$, which we denote by U'_N. Observe that if N is an endpoint of the link, then U'_N is a terminal of $\mathcal{W}_{S_1,S_2}$ (since Σ_N contains one of S_1 and S_2).

Let now N_1 and N_2 be two neighboring nodes in the link such that N_1 and $\varphi_S(S_1)$ lie on the same side of the edge (N_1, N_2) in $\mathcal{H}_S$. Let us consider the set of all $(S_1 \cup \Sigma_{N_1}, S_2 \cup \Sigma_{N_2})$-mincuts (all of them are represented in $\mathcal{W}_{S_1,S_2}$). It is easy to see that the subtrees $\mathcal{T}(S_1 \cup \Sigma_{N_1})$ and $\mathcal{T}(S_2 \cup \Sigma_{N_2})$ contain nodes N_1 and N_2, respectively. By Fact 6.3(ii), this set of cuts is represented by the strip $\mathcal{W}(N_1, N_2)$ defined by the structural edge (N_1, N_2). Therefore, by Lemma 3.8, to get the strip $\mathcal{W}(N_1, N_2)$, it is sufficient to contract the corresponding cones of the nodes U'_{N_1} and U'_{N_2} in $\mathcal{W}_{S_1,S_2}$.

By Facts 3.6(iii) and 6.3(ii) and Lemma 6.1, these cones are just the inner sides of the tight cuts $\mathcal{C}(N_1, (N_1, N_2))$ and $\mathcal{C}(N_2, (N_1, N_2))$, respectively. Since the intersection of the two opposite cones of any unit in a strip coincides with this unit, we thus get that the units U'_N constructed in the proof coincide with the sets U_N defined in the formulation of the theorem.

Observe that, by Lemma 5.4, one can obtain $\mathcal{W}(N_1, N_2)$ directly from $\mathcal{F}_S$ by contracting the inner sides of the same tight cuts $\mathcal{C}(N_1, (N_1, N_2))$ and $\mathcal{C}(N_2, (N_1, N_2))$. Therefore, there is a bijection between the units of $\mathcal{F}_S$ and $\mathcal{W}_{S_1,S_2}$ that lie between

pairs of the above-described tight cuts; moreover, this bijection preserves the inherent partitions. Let us consider the set of units in $\mathcal{W}_{S_1,S_2}$ that do not participate in that bijection. One can deduce from Lemma 6.1 and Corollary 6.2 that each such unit U lies either inside the tight cut $\mathcal{C}(N,(N,M))$, where N is an endpoint of the link and (N,M) belongs to the link, or in the intersection of the inner sides of the two tight cuts $\mathcal{C}(N,(N,N_1))$ and $\mathcal{C}(N,(N,N_2))$, where N is an inner node of the link and N_1 and N_2 are the neighbors of N in the link. In the first case, U is one of the terminal units of $\mathcal{W}_{S_1,S_2}$ (by Fact 6.3(iii)). In the second case, U lies in the intersection of the two opposite cones of the unit U_N in $\mathcal{W}_{S_1,S_2}$, and thus coincides with U_N. The claim concerning the inherent partition at U_N follows immediately from the above construction and Fact 3.3. □

Remarks. (1) Let us distinguish a subset of coherent paths in $\mathcal{F}_S$ by preventing their inner units from "falling inside" any unit U_N, $N \in \mathcal{L}(S_1,S_2)$. By the construction above, these paths correspond naturally to the coherent paths of $\mathcal{W}_{S_1,S_2}$ without inner units U_N, $N \in \mathcal{L}(S_1,S_2)$.

(2) Observe that for any path $P = (N_0,\varepsilon_1,\dots,\varepsilon_r,N_r)$ in $\mathcal{H}_S$, the link of the subsets of S lying inside $\mathcal{C}(N_0,\varepsilon_1)$ and $\mathcal{C}(N_r,\varepsilon_r)$ is exactly P, since there are no empty nodes of degree at most two in $\mathcal{H}_S$. The strip corresponding to these two subsets is denoted by $\mathcal{W}(P)$.

6.3. The projection mapping. Let U be an arbitrary unit of the flesh $\mathcal{F}_S$. We say that U is *projected* to an edge ε of the skeleton $\mathcal{H}_S$ if U is a nonterminal unit of $\mathcal{W}(\varepsilon)$ (for example, unit U_2 in Figure 5.1(b) is a stretched unit of the strip $\mathcal{W}(N_1,N)$ shown in Figure 5.1(d), and thus U_2 is projected to the edge (N_1,N)). In other words, U is projected to $\varepsilon = (N_1,N_2)$ if it lies outside the N_1-tight and N_2-tight cuts in direction ε.

THEOREM 6.5. (i) *For any stretched unit U of the flesh, the set $\pi_S(U)$ of structural edges to which U is projected is nonempty and is the edge-set of a path in the skeleton.*

(ii) *For any terminal unit U of the flesh, the set of structural edges to which U is projected is empty, and there exists a unique node $\pi_S(U)$ of the skeleton such that U belongs to all $\pi_S(U)$-tight cuts.*

Proof. (i) The set $\pi_S(U)$ is nonempty by the definition of a stretched unit. Let us show that there exists a path in the skeleton containing all the structural edges in $\pi_S(U)$. If there is only one such edge, then the statement is trivial; thus in what follows we assume that $|\pi_S(U)| \geqslant 2$. Let us take a path $P = (N_0,\varepsilon_1,N_1,\dots,N_{r-1},\varepsilon_r,N_r)$ that is inclusion-maximal among all paths with both end-edges in $\pi_S(U)$. Suppose, to the contrary, that there exists an edge $\varepsilon \in \pi_S(U)$ that does not belong to P (see Figure 6.2(a)). Then the link of ε and P meets P in an inner node N_i (otherwise P could be extended). We denote by ε' the edge of this link incident to N_i. By Lemma 6.1, U lies outside the cuts $\mathcal{C}(N_i,\varepsilon_i)$, $\mathcal{C}(N_i,\varepsilon_{i+1})$, and $\mathcal{C}(N_i,\varepsilon')$. However, this contradicts the following generalization of part (i) of the 3-Star Lemma.

LEMMA 6.6. *Let N be a node of $\mathcal{H}_S$, ε_1, ε_2, ε_3 be structural edges incident to N, $\mathcal{C}_i = \mathcal{C}(N,\varepsilon_i)$, $i \in \{1,2,3\}$. Then $\bigcap_i \bar{V}_{\mathcal{C}_i} = \varnothing$.*

Proof. Let us choose a vertex $s_i \in S$ in such a way that $s_i \in \bar{V}_{\mathcal{C}_i}$, $i = 1,2,3$. Vertices s_i are distinct, since they belong to inverse images (under φ_S) of disjoint subtrees of $\mathcal{H}_S$. Observe that $\mathcal{T}(\{s_j,s_k\})$ is a path containing both ε_j and ε_k, but not containing ε_i. Thus, by Fact 6.3(iii), the $\{s_j,s_k\}$-tight $(\{s_j,s_k\},s_i)$-mincut coincides with $\mathcal{C}_i$. Therefore, $\mathcal{C}_i$ is exactly the cut $\mathcal{C}_{s_i}$ as defined in section 4.4, and the assertion of the lemma follows from part (i) of the 3-Star Lemma. □

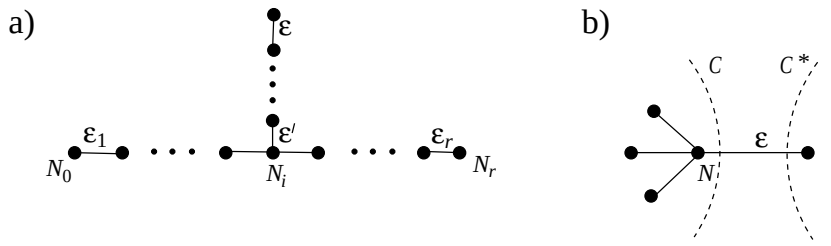


FIG. 6.2. *To the proofs of Theorem* 6.5 *and Lemma* 6.6.

Remark. Clearly, the remaining assertions of the 3-Star Lemma can be generalized to our case as well.

It remains to show that each edge $(N_{i-1}, \varepsilon_i, N_i)$ in P belongs to $\pi_S(U)$. Indeed, U lies outside both $\mathcal{C}(N_1, \varepsilon_1)$ and $\mathcal{C}(N_{r-1}, \varepsilon_r)$ and thus, by Lemma 6.1, outside both $\mathcal{C}(N_i, \varepsilon_i)$ and $\mathcal{C}(N_{i-1}, \varepsilon_i)$, as required.

(ii) Now let U be a terminal unit of the flesh. Then, for each structural edge ε, U lies inside exactly one of the two tight cuts in direction ε. Let us consider the collection $\mathcal{I}$ of all tight cuts that contain U inside and have inclusion-minimum inner sides. Let $\mathcal{C}'$ and $\mathcal{C}''$ be two arbitrary cuts in $\mathcal{I}$, ε' and ε'' be the corresponding structural edges, and let $[\varepsilon', \varepsilon''] = (N_0, \varepsilon', N_1, \ldots, N_{r-1}, \varepsilon'', N_r)$. Then, by Lemma 6.1 and the minimality of the cuts in question, one of the following possibilities holds: either $\mathcal{C}' = \mathcal{C}(N_0, \varepsilon')$ and $\mathcal{C}'' = \mathcal{C}(N_r, \varepsilon'')$ or $\mathcal{C}' = \mathcal{C}(N_1, \varepsilon')$ and $\mathcal{C}'' = \mathcal{C}(N_{r-1}, \varepsilon'')$.

In the first case, the tight cut $\mathcal{C}(N_1, \varepsilon')$ contains U inside (by Lemma 6.1), and thus both tight cuts in direction ε' contain U inside, a contradiction. In the second case, let us assume that there exists an edge $\varepsilon \in P(\varepsilon', \varepsilon'')$ distinct from ε' and ε''. Then, by Lemma 6.1 and the minimality of $\mathcal{C}'$ and $\mathcal{C}''$, both tight cuts in direction ε do not contain a terminal U inside, a contradiction. Therefore, $\mathcal{C}'$ and $\mathcal{C}''$ are tight cuts of the same node N ($N = N_1 = N_{r-1}$). One can prove easily that any other cut in $\mathcal{I}$ is also an N-tight cut.

Moreover, any N-tight cut belongs to $\mathcal{I}$. Indeed, assume that $\mathcal{C}$ is the N-tight cut in direction ε and $\mathcal{C} \notin \mathcal{I}$ (see Figure 6.2(b)). Denote by $\mathcal{C}^*$ the other tight cut in direction ε. Then U lies either inside $\mathcal{C}$ or inside $\mathcal{C}^*$. In the first case, there exists a cut in $\mathcal{I}$ dominated by $\mathcal{C}$. (For the definition of domination, see the beginning of section 2.) By the above reasoning, this cut is an N-tight cut, and we thus get two N-tight cuts with nested inner sides, which contradicts Corollary 6.2. In the second case, by Lemma 6.1, $\mathcal{C}^*$ is dominated by all the N-tight cuts except for $\mathcal{C}$, and thus by all the cuts in $\mathcal{I}$, which contradicts the minimality of these cuts. Therefore, we can set $\pi_S(U) = N$. □

Thus, the *projection* mapping π_S is defined, assigning to each unit U the path of Theorem 6.5 (observe that a single node is itself a path); the endpoints of this path are called the *coordinates* of U in $\mathcal{H}_S$ (compare with the definition preceding Lemma 4.13). For an example see Figures 5.1(b) and (c): the coordinates of the unit U_1 are N and N_3, those of U_2 are N_1 and N_3, and those of U are N and N.

The triple $(\mathcal{F}_S, \mathcal{H}_S, \pi_S)$ is called the *connectivity carcass* of S. According to Theorems 5.5(iii) and 5.6, its size is $O(\min\{\tilde{m}, \lambda_S \tilde{n}\}) = O(\min\{m, \lambda_S n\})$.

Let us consider a cut of the skeleton $\mathcal{H}_S$ defined by a structural edge and repre-

senting a bunch $\mathcal{B}$. If we delete this edge from the skeleton, it falls into two connected components $\mathcal{H}_S(\mathcal{B})$ and $\bar{\mathcal{H}}_S(\mathcal{B})$ containing $\varphi_S(S_{\mathcal{B}})$ and $\varphi_S(\bar{S}_{\mathcal{B}})$, respectively.

THEOREM 6.7. *The set of vertices lying inside the tight (resp., outside the loose) cut of $\mathcal{B}$ is the union of all units U of $\mathcal{F}_S$ such that $\pi_S(U) \subseteq \mathcal{H}_S(\mathcal{B})$ (resp., $\pi_S(U) \subseteq \bar{\mathcal{H}}_S(\mathcal{B})$).*

Proof. Let us denote by ε the structural edge corresponding to $\mathcal{B}$, and by N and $\bar{N}$ its endpoints lying in $\mathcal{H}_S(\mathcal{B})$ and $\bar{H}_S(\mathcal{B})$, respectively. Then the tight cut of $\mathcal{B}$ is $\mathcal{C}(N, \varepsilon)$, and the loose cut of $\mathcal{B}$ is $\bar{\mathcal{C}}(\bar{N}, \varepsilon)$. Now let U be an arbitrary unit of $\mathcal{F}_S$. If $\varepsilon \in \pi_S(U)$, then, by definition, U lies outside both $\mathcal{C}(N, \varepsilon)$ and $\mathcal{C}(\bar{N}, \varepsilon)$. Otherwise, $\varepsilon \notin \pi_S(U)$; by Theorem 6.5 we may assume w.l.o.g. that $\pi_S(U) \subseteq \mathcal{H}_S(\mathcal{B})$. Let us prove that U lies inside $\mathcal{C}(N, \varepsilon)$.

Assume to the contrary that this is not the case; then U lies inside $\mathcal{C}(\bar{N}, \varepsilon)$, since otherwise the projection of U would contain ε. If U is a stretched unit, then there exists a structural edge $\varepsilon' \in \pi_S(U) \subseteq \mathcal{H}_S(\mathcal{B})$. Let $[\varepsilon', \varepsilon] = (\bar{N}', \varepsilon', N', \ldots, N, \varepsilon, \bar{N})$. Then U lies outside $\mathcal{C}(N', \varepsilon')$ and inside $\mathcal{C}(\bar{N}, \varepsilon)$, in a contradiction to Lemma 6.1. Now let U be a terminal unit, and $N' = \pi_S(U) \subseteq \mathcal{H}_S(\mathcal{B})$ be its projection. We consider the link $(N', \varepsilon_1, \ldots, N)$ of N' and ε in $\mathcal{H}_S$. By Theorem 6.5(ii), U lies inside $\mathcal{C}(N', \varepsilon_1)$, which, by Lemma 6.1, contradicts to the fact that U lies outside $\mathcal{C}(N, \varepsilon)$. □

It follows readily from Theorem 6.7 that the projections of distinct terminal units are distinct. (It suffices to take for $\mathcal{B}$ the bunch of an arbitrary cut separating one of these units from the other.) Hence, π_S provides an *injection* of the set of terminal units into the set of nodes of $\mathcal{H}_S$.

Theorem 6.7 provides another point of view on the construction of Theorem 6.4.

COROLLARY 6.8. *A unit U_N is the union of all units U of $\mathcal{F}_S$ such that $\pi_S(U) \subseteq \mathcal{T}_N$. A unit U of $\mathcal{F}_S$ does not fall inside any unit U_N, $N \in \mathcal{L}(S_1, S_2)$, if and only if $\pi_S(U) \cap \mathcal{L}(S_1, S_2) \neq \varnothing$. A unit U_N is reachable from U in $\mathcal{W}_{S_1,S_2}$ if and only if N is not inner for $\pi_S(U) \cap \mathcal{L}(S_1, S_2)$.*

The following statement reveals an intimate relation between the projection of a stretched unit and its reachability cones. Let U be a stretched unit, $\mathcal{R}_1$ and $\mathcal{R}_2$ be the reachability cones of U, $\mathcal{N}_1$ and $\mathcal{N}_2$ be the sets of nodes that are the projections of the terminals belonging to $\mathcal{R}_1$ and $\mathcal{R}_2$, respectively, and $\mathcal{T}_1$ and $\mathcal{T}_2$ be the minimum subtrees of $\mathcal{H}_S$ containing $\mathcal{N}_1$ and $\mathcal{N}_2$, respectively. Observe that each reachability cone of U contains at least one terminal, and hence both $\mathcal{T}_1$ and $\mathcal{T}_2$ are nonempty.

THEOREM 6.9. *The projection of U coincides with the link of $\mathcal{T}_1$ and $\mathcal{T}_2$.*

Proof. Let ε be an arbitrary structural edge in the projection $\pi_S(U)$ and $\mathcal{B}$ be one of the two opposite bunches represented by ε. It is easy to see that all the terminals that belong to $\mathcal{R}_1$ lie inside the tight cut of $\mathcal{B}$, while those in $\mathcal{R}_2$ lie outside the loose cut of $\mathcal{B}$, up to flipping of $\mathcal{B}$. Hence, by Theorem 6.7, $\mathcal{T}_1 \subseteq \mathcal{H}_S(\mathcal{B})$ and $\mathcal{T}_2 \subseteq \bar{\mathcal{H}}_S(\mathcal{B})$. It follows immediately that the trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are disjoint, and that $\pi_S(U) \subseteq \mathcal{L}(\mathcal{T}_1, \mathcal{T}_2)$.

Assume now that $\pi_S(U) \neq \mathcal{L}(\mathcal{T}_1, \mathcal{T}_2)$; then there exists a subpath $P = (N', \varepsilon', N, \varepsilon'', N'')$ of the link such that ε' belongs to $\pi_S(U)$, while ε'' does not. By Corollary 6.8, U_N is reachable from U in $\mathcal{W}(P)$. Hence, by the remark after Theorem 6.4, there exists a coherent path in $\mathcal{F}_S$ that leads from U to the intersection of the inner sides of $\mathcal{C}(N, \varepsilon')$ and $\mathcal{C}(N, \varepsilon'')$. Let us extend this path to a terminal U^* of $\mathcal{F}_S$ (see Figure 6.3).

If the extended path leaves the inner side of the N-tight cut in direction ε for some $\varepsilon \neq \varepsilon', \varepsilon''$, then it cannot return back, and, by Theorem 6.7, the projection of U^* lies in the subtree of $\mathcal{H}_S$ that is separated from N by ε. However, this projection belongs either to $\mathcal{T}_1$ or to $\mathcal{T}_2$, and hence one of the edges ε' and ε'' does not belong

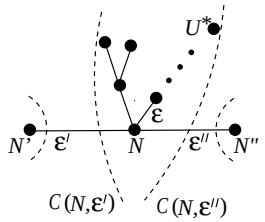


FIG. 6.3. *To the proof of Theorem* 6.9.

to $\mathcal{L}(\mathcal{T}_1, \mathcal{T}_2)$, a contradiction.

Otherwise, U^* lies inside all the N-tight cuts, and thus, by Theorem 6.5, it is projected to N, which gives the same contradiction as above. □

By Theorem 6.9, each of $\mathcal{T}_1$ and $\mathcal{T}_2$ contains exactly one coordinate of U. Therefore, we can label the reachability cones of any stretched unit (and thus the parts of the inherent partition) by its coordinates; the reachability cone of U that contains the terminals whose projections coincide with or lie "behind" the coordinate N is denoted by $\mathcal{R}_N(U)$. Moreover, we can now redenote the trees $\mathcal{T}_i$: the tree that contains N is denoted by $\mathcal{T}_N(U)$.

6.4. The cell structure. For any path in the skeleton with endpoints M and N we define the *cell* Q^{MN} as the set of units with the coordinates M, N (compare with definitions preceding the 3-Star Lemma and Lemma 4.10); by definition, the cell Q^{NM} coincides with Q^{MN}. A cell of type Q^{MM} is called *terminal*; it is either empty or consists of exactly one terminal unit (with projection M).

Let us introduce a visibility relation, which is, in a sense, a relaxation of the reachability relation studied in section 3. We say that a cell $Q^{M_2N_2}$ is *visible* from a nonterminal cell $Q^{M_1N_1}$ *in direction* N_1 if both $P_1 = [M_1, N_1]$ and $P_2 = [M_2, N_2]$ are subpaths of the same path P, so that $P = [M_1, N_2]$ or $P = [M_1, M_2]$. We denote this path P by $[P_1, P_2]$ (which is consistent with the previous use of notation $[\cdot, \cdot]$). It is easy to see that if $Q^{M_2N_2}$ is visible from $Q^{M_1N_1}$ in both directions, then $Q^{M_2N_2} = Q^{M_1N_1}$. The following statement, which is a generalization of Lemma 4.11, shows that the visibility relation is indeed a relaxation of the reachability relation.

THEOREM 6.10. *Let U_1 be a stretched unit with coordinates M_1 and N_1, $U_2 \in \mathcal{R}_{N_1}(U_1)$, and let Q_1 and Q_2 be the cells containing U_1 and U_2, respectively. Then Q_2 is visible from Q_1 in direction N_1.*

Proof. If U_2 is a terminal, then the assertion follows immediately from Theorem 6.9. Now let U_2 be a stretched unit with coordinates M_2 and N_2, and let w.l.o.g. $U_1 \in \mathcal{R}_{M_2}(U_2)$. Since $\mathcal{R}_{N_2}(U_2) \subseteq \mathcal{R}_{N_1}(U_1)$, we get $\mathcal{T}_{N_2}(U_2) \subseteq \mathcal{T}_{N_1}(U_1)$; hence the path $[N_1, N_2]$ lies entirely in $\mathcal{T}_{N_1}(U_1)$, and, by Theorem 6.9, intersects with $[M_1, N_1]$ exactly by N_1. Therefore, N_1 lies on $[M_1, N_2]$. Since U_2 is stretched, we can prove in a similar way that M_2 lies on $[M_1, N_2]$, as required. □

By Theorem 6.10, when we consider several units lying on the same coherent path, we may assume that their coordinates are named consistently. In other words, if $\pi_S(U_1) = [M_1, N_1]$, $\pi_S(U_2) = [M_2, N_2]$, and $U_2 \in \mathcal{R}_{N_1}(U_1)$, then we assume that U_1 belongs to $\mathcal{R}_{M_2}(U_2)$, and hence $[\pi_S(U_1), \pi_S(U_2)] = [M_1, N_2]$.

The following statement can be considered as the main relation between coherent

paths in the flesh and the skeleton.

COROLLARY 6.11. *Let U lie on a coherent path P from U_1 to U_2; then $M \in [M_1, M_2]$ and $N \in [N_1, N_2]$.*

Remark. It follows immediately from Corollary 6.11 that when one builds the strip $\mathcal{W}([M_1, N_2])$ (as described in Theorem 6.4), no intermediate unit of P is subject to contraction.

We now can extend the definition of the projection to the edges of $\mathcal{F}_S$, similar to what was done in section 4.5. Let $e = (U_1, U_2)$ be an edge of $\mathcal{F}_S$ and assume that $U_1 \in Q^{M_1N_1}$, $U_2 \in Q^{M_2N_2}$. We then say that the *projection* of e equals $[M_1, N_2]$; in other words, we define $\pi_S(U_1, U_2) = [\pi_S(U_1), \pi_S(U_2)]$. As in section 4.5, this definition can be justified by inserting a dummy vertex in the edge e; it is easy to see that the projection of the unit that contains such a vertex is exactly $[M_1, N_2]$. As before, there are three types of edges belonging to a cell Q^{MN}: internal (both endpoints belong to Q^{MN}), external (exactly one endpoint belongs to Q^{MN}), and through (both endpoints do not belong to Q^{MN}).

For any proper path $[M, N]$, $M \neq N$, we build from $\mathcal{F}_S$ a locally orientable graph $\mathcal{W}^{MN}$ with the set of nonterminals Q^{MN} (which may be as well empty). We delete all the cells nonvisible from Q^{MN} (observe that by Theorem 6.10 there are no edges between Q^{MN} and the deleted sets) and contract all the cells visible from Q^{MN} only in direction M and those visible only in direction N into two terminals $\tilde{M}$ and $\tilde{N}$, respectively. Finally, we delete all the edges between $\tilde{M}$ and $\tilde{N}$ that do not belong to Q^{MN}. An S-mincut $\mathcal{C}$ of G that separates the preimages of the terminals, and thus generates an $(\tilde{M}, \tilde{N})$-mincut $\tilde{\mathcal{C}}$ in $\mathcal{W}^{MN}$, is called an *extension* of $\tilde{\mathcal{C}}$ (compare with definitions preceding Lemmas 4.2 and 4.8). The following result is a generalization of Lemmas 4.2, 4.8, and 4.12.

THEOREM 6.12. (i) *The graph $\mathcal{W}^{MN}$ is a strip; moreover, if $[M, N]$ contains at least two structural edges, then its width λ^{MN} does not exceed $\lambda_S/2$.*

(ii) *All its transversal cuts, and only they, are extendable to S-mincuts. Such an extension can be found in any bunch corresponding to an edge in $[M, N]$ and in no other bunch.*

(iii) *Let $\{Q^{M_iN_i}\}$ be a set of pairwise nonvisible cells such that all the paths $[M_i, N_i]$ contain at least one common edge. Then any set of transversal cuts of $\mathcal{W}^{M_iN_i}$ has a mutual extension to an S-mincut. Moreover, if all the paths $[M_i, N_i]$ contain at least two common edges, then $\sum_i \lambda^{M_iN_i} \leqslant \lambda_S/2$.*

(iv) *If a 2-partition of $V \setminus \bigcup_i Q^{M_iN_i}$ enters a mutual extension of some sample of transversal cuts, one from each of $\mathcal{W}^{M_iN_i}$, then it enters some mutual extension of any such sample. (Informally, samples of transversal cuts of these strips are interchangeable in S-mincuts.)*

Proof. We start from the following general observation. Let $\varepsilon' = (M', N')$ be an arbitrary structural edge of $[M, N]$. (We assume that $[M, M']$ and $[N, N']$ do not intersect.) Let us consider the strip $\mathcal{W}(\varepsilon')$. Its terminals correspond naturally to M' and N'. Evidently, any unit of Q^{MN} remains stretched in $\mathcal{W}(\varepsilon')$; its cones and the sides of its star are labeled by M' and N'. Let $\mathcal{R}_{M'}$ denote the union over $U \in Q^{MN}$ of the cones $\mathcal{R}_{M'}(U)$ in $\mathcal{W}(\varepsilon')$; $\mathcal{R}_{N'}$ is defined similarly.

Let us delete $\bar{\mathcal{R}}_{M'} \cap \bar{\mathcal{R}}_{N'}$ and contract $\bar{\mathcal{R}}_{M'} \cap \mathcal{R}_{N'}$ and $\mathcal{R}_{M'} \cap \bar{\mathcal{R}}_{N'}$ into two new terminals $\tilde{M}'$ and $\tilde{N}'$, respectively. It is easy to see that each direct edge between $\tilde{M}'$ and $\tilde{N}'$ corresponds to a through edge in Q^{MN}; the converse is not true, since the endpoints of a through edge do not necessarily belong to $\mathcal{R}_{M'}$ or $\mathcal{R}_{N'}$. Thus, the number of direct edges between $\tilde{M}'$ and $\tilde{N}'$ does not exceed the number of through

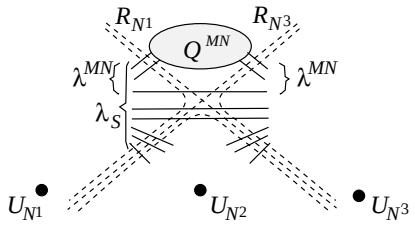


FIG. 6.4. *To the proof of inequality* $\lambda^{MN} < \lambda_S/2$.

edges in Q^{MN}. If it is strictly less, we add complementary direct edges in order to equalize these quantities. Let us denote the obtained locally orientable graph by $\tilde{\mathcal{W}}^{MN}$. We claim that $\tilde{\mathcal{W}}^{MN}$ coincides with $\mathcal{W}^{MN}$.

First of all, $Q^{MN} = \mathcal{R}_{M'} \cap \mathcal{R}_{N'}$. Indeed, the inclusion $Q^{MN} \subseteq \mathcal{R}_{M'} \cap \mathcal{R}_{N'}$ is trivial. On the other hand, if $U \in \mathcal{R}_{M'} \cap \mathcal{R}_{N'}$, then there exist $U', U'' \in Q^{MN}$ such that U lies on a coherent path from U' to U''. Hence, by Corollary 6.11, $\pi_S(U) = [M, N]$ and so $U \in Q^{MN}$.

Let us consider an arbitrary edge $e = (U, U^*)$ such that $U \in Q^{MN}$ and $U^* \notin Q^{MN}$. While constructing $\tilde{\mathcal{W}}^{MN}$ and $\mathcal{W}^{MN}$, this edge is not deleted (by Lemma 2.1 and Theorem 6.10, respectively) and goes from U to a terminal. In $\tilde{\mathcal{W}}^{MN}$ this terminal corresponds to the label of the side at U in $\mathcal{F}_S$ that contains e. By Theorems 6.7 and 6.10, the same holds true for $\mathcal{W}^{MN}$. Therefore, the stars of the corresponding terminals in $\tilde{\mathcal{W}}^{MN}$ and $\mathcal{W}^{MN}$ coincide, and thus these locally orientable graphs coincide as well.

(i) By the construction of $\tilde{\mathcal{W}}^{MN}$, it is a balanced acyclic two-terminal locally orientable graph; hence, by part (i) of the Strip Lemma, it is a strip.

Let $[M, N]$ contain at least two structural edges, and let N^1, N^2, N^3 be any three consequent nodes in $[M, N]$. We consider the strip $\mathcal{W}([N^1, N^3])$ and define the cones $\mathcal{R}_{N^1}$ and $\mathcal{R}_{N^3}$ in it and the strip $\tilde{\mathcal{W}}^{MN} \equiv \mathcal{W}^{MN}$ in the same way as above. By Theorem 6.10 and the construction of $\mathcal{W}([N^1, N^3])$ in Theorem 6.4, the contracted unit U_{N^2} belongs to $\bar{\mathcal{R}}_{N^1} \cap \bar{\mathcal{R}}_{N^3}$. Since $\mathcal{C}^1 = (\mathcal{R}_{N^1}, \bar{\mathcal{R}}_{N^1})$ and $\mathcal{C}^3 = (\mathcal{R}_{N^3}, \bar{\mathcal{R}}_{N^3})$ are S-mincuts, there are no edges between $\mathcal{R}_{N^1} \cap \mathcal{R}_{N^3}$ and $\bar{\mathcal{R}}_{N^1} \cap \bar{\mathcal{R}}_{N^3}$ (by Lemma 2.1). Next, since $\mathcal{C}^1 \cap \bar{\mathcal{C}}^3$ is an S-mincut, one has $c(\mathcal{R}_{N^1} \cap \bar{\mathcal{R}}_{N^3}, \bar{\mathcal{R}}_{N^1} \cap \bar{\mathcal{R}}_{N^3}) = \lambda_S - \lambda^{MN}$; similarly, $c(\bar{\mathcal{R}}_{N^1} \cap \mathcal{R}_{N^3}, \bar{\mathcal{R}}_{N^1} \cap \bar{\mathcal{R}}_{N^3}) = \lambda_S - \lambda^{MN}$ (see Figure 6.4). Hence, $c(\bar{\mathcal{R}}_{N^1} \cap \bar{\mathcal{R}}_{N^3}, \mathcal{R}_{N^1} \cup \mathcal{R}_{N^3}) = 2(\lambda_S - \lambda^{MN})$. However, since $U_{N^2} \cap S \neq \varnothing$, one has $c(\bar{\mathcal{R}}_{N^1} \cap \bar{\mathcal{R}}_{N^3}, \mathcal{R}_{N^1} \cup \mathcal{R}_{N^3}) \geqslant \lambda_S$. Thus $\lambda^{MN} < \lambda_S/2$, as required.

(ii) Existence of an extension for any $(M', N') \in [M, N]$ is proved in the same way as it was done in Lemma 4.12(ii), with the set X_a replaced by $\mathcal{R}_{M'} \cap \bar{\mathcal{R}}_{N'}$ and $\bar{X}_c$ by $\mathcal{R}_{N'} \cap \bar{\mathcal{R}}_{M'}$.

Any extension of a transversal cut of $\mathcal{W}^{MN}$ separates the preimages of $\tilde{M}$ from those of $\tilde{N}$ and hence the set of terminals "behind" M from the set of terminals "behind" N. By Fact 6.3(ii) and the second remark after Theorem 6.4, the bunch of such a cut must be defined by an edge in $[M, N]$.

(iii) Let $\varepsilon' = (M', N')$ be a common edge of all the $[M_i, N_i]$. We consider the strip $\mathcal{W}(\varepsilon')$ and build the sets $\mathcal{R}^i_{M'}$ and $\mathcal{R}^i_{N'}$ as above. Let $\mathcal{R}^*_{M'}$ be the union of $\mathcal{R}^i_{M'}$ over all i, and let $\mathcal{R}^*_{M'}$ be defined similarly. Let $e = (U, U^*)$ be an edge such that

$U \in Q^{M_iN_i}$, $U^* \notin Q^{M_iN_i}$, and $U^* \notin \mathcal{R}^i_{M'}$; then $U^* \notin \mathcal{R}^*_{M'}$. Indeed, assume to the contrary that $U^* \in \mathcal{R}^j_{M'}$, $j \neq i$; then, by Theorem 6.10, the cell $Q^{M_iN_i}$ is visible from $Q^{M_jN_j}$, a contradiction. The rest of the proof of the existence of a mutual extension is similar to that of Lemma 4.12(iii), as above. The inequality for $\sum_i \lambda^{M_iN_i}$ follows immediately from Lemma 4.12(iii) applied to the strip $\mathcal{W}(\cap_i[M_i, N_i])$. One should note, however, that some nodes of $(\cap_i[M_i, N_i])$ can be empty, and hence the inequality of type $\lambda_{ii} > \lambda_S/2$ for such a node is replaced by $\lambda_{ii} \geqslant \lambda_S/2$. As a consequence, we get a nonstrict inequality for $\sum_i \lambda^{M_iN_i}$.

(iv) The proof is similar to that of Lemma 4.12(iv). □

6.5. Other properties of the connectivity carcass. Assume that we are interested in pointing out an S-mincut separating two given vertices of G that lie in distinct units U_1 and U_2. The following statement is obtained easily.

LEMMA 6.13. (i) *Let $\pi_S(U_1)$ and $\pi_S(U_2)$ be distinct* nodes *of $\mathcal{H}_S$. Consider any cut of $\mathcal{H}_S$ separating them, and let $\mathcal{B}$ be any of the corresponding bunches. Then the tight and the loose cuts of $\mathcal{B}$ separate U_1 from U_2.*

(ii) *Let one of the projections, say, $\pi_S(U_1)$, contain a structural edge $\varepsilon_1 = (M_1, N_1)$, and let M, N be the coordinates of U_1 in the corresponding order. Then one of the sets $\mathcal{R}_M(U_1) \cup V^{\mathcal{C}(M_1,\varepsilon_1)}$ and $\mathcal{R}_N(U_1) \cup V^{\mathcal{C}(N_1,\varepsilon_1)}$ defines an S-mincut separating U_1 from U_2.*

Proof. (i) The proof follows from Fact 6.3(ii).

(ii) It is clear from the structure of $\mathcal{W}_{\varepsilon_1}$ that both sets mentioned define S-mincuts; on the other hand, their intersection is exactly $U_1 \neq U_2$, and the result follows. □

Consider now a pair s, t of non-(λ_S+1)-connected vertices in S. Let us decompose the connectivity carcass with respect to this pair.

Let $U_i = U_{N_i}$, $0 \leqslant i \leqslant k$, be the heavy units of $\mathcal{W}_{s,t}$, $s \in U_0$, $t \in U_k$ (see Theorem 6.4 for details); it is easy to prove that U_i are numbered according to any topological order of $\mathcal{W}_{s,t}$. For any i as above we put $\Sigma_i = S \cap U_i$ (which coincides with Σ_{N_i} defined in the proof of Theorem 6.4) and contract the set S in two different ways: S^i is obtained by contraction of $S \setminus \Sigma_i$ into a single vertex s^i, and $\tilde{S}$ is obtained by contraction of each of Σ_i into a single vertex $\tilde{s}_i$. The graphs G^i and $\tilde{G}$ are defined similarly as the results of the above two contractions applied to G. Note that the connectivity of $\tilde{S}$ in $\tilde{G}$ is exactly λ_S, while the connectivity of S^i in G^i can be greater than λ_S.

It can be shown easily that the connectivity carcass of $\tilde{S}$ in $\tilde{G}$ has the same type as one considered in section 4.5. Its skeleton is the path $[N_0, N_k]$ and its flesh is obtained from $\mathcal{W}_{s,t}$ by turning the heavy units U_i, $0 \leqslant i \leqslant k$, into terminals.

Let I denote the set of indices such that $\lambda_{S^i}(G^i) = \lambda_S$.

LEMMA 6.14. (i) *The skeleton $\mathcal{H}_S$ is obtained from the path $[N_0, N_k] = \mathcal{H}_{\tilde{S}}(\tilde{G})$ and the skeletons $\mathcal{H}_{S^i}(G^i)$, $i \in I$, by identifying N_i with the node $N^i = \varphi_{S^i}(s^i)$ of $\mathcal{H}_{S^i}(G^i)$.*

(ii) *The units of the flesh $\mathcal{F}_S$ are*

- *the units of $\mathcal{F}_{S^i}(G^i)$, $i \in I$, except for the unit U^i containing s^i;*
- *the nonheavy units of $\mathcal{W}_{s,t}$;*
- *the terminal units $U^{(i)} = U^i \cap V_{\mathcal{C}(N_i,(N_i,N_{i-1}))} \cap V_{\mathcal{C}(N_i,(N_i,N_{i+1}))} \neq \varnothing$, $i \in I$, and $U^{(i)} = V_{\mathcal{C}(N_i,(N_i,N_{i-1}))} \cap V_{\mathcal{C}(N_i,(N_i,N_{i+1}))} \neq \varnothing$, $i \notin I$ (for $i = 0$ and $i = k$ only one of the above two cuts is taken), if nonempty.*

Moreover, if a unit other than $U^{(i)}$ participates in a more than one flesh as above, then it participates in $\mathcal{F}_{\tilde{S}}(\tilde{G})$ and at most two of $\mathcal{F}_{S^i}(G^i)$.

(iii) *The projection of any unit other than $U^{(i)}$ is the concatenation of its projections in the connectivity carcasses that contain it.*

Proof. Observe that each S-mincut of G either partitions one of Σ_i, $i \in I$, and does not partition its complement in S or separates some of Σ_i from the others according to the skeleton $\mathcal{H}_S$. More exactly, each bunch of S-mincuts of G coincides either with a bunch of $\tilde{S}$-mincuts of $\tilde{G}$ or with a bunch of S^i-mincuts of G^i for some $i \in I$. This implies assertion (i).

Let U be a terminal unit of $\mathcal{F}_S$ corresponding to a node N of $\mathcal{H}_S$. Recall that U is the intersection of all N-tight cuts. If N belongs to the subtree $\mathcal{H}_i$ of $\mathcal{H}_S$ hanging on the path $[N_0, N_k]$ at N_i and $N \neq N_i$, then these cuts are the N-tight cuts of the connectivity carcass of S^i in G^i. Therefore, U is a terminal unit of this partial connectivity carcass. If $N = N_i$, $i \in I$, then all the N-tight cuts are of the same type, except for one or two cuts corresponding to the edges of $[N_0, N_k]$ incident to N_i. Clearly, U_i is a unit of $\mathcal{F}_S$ for any $i \notin I$. Thus we get assertion (ii) for terminals.

Let U be a stretched unit. It is distinguished by all bunches that correspond to the edges of its projection. The projection is contained either in one of $\mathcal{H}_i$, or in $\mathcal{H}_i \cup [N_0, N_k]$ for some i, or in $\mathcal{H}_i \cup [N_i, N_j] \cup \mathcal{H}_j$. The rest of assertion (ii) follows, as well as assertion (iii). □

7. Incremental transformations of the connectivity carcass.

7.1. Transformations of the components of the connectivity carcass. In this section we describe the transformations of all the three components of the connectivity carcass caused by insertion of a new edge (u_1, u_2) that preserves the value of λ_S. In what follows U_i stands for the unit containing u_i, Q_i for the cell containing U_i, and P_i for the projection $\pi_S(U_i)$, $i = 1, 2$. If $U_1 = U_2$, then the carcass does not change. So in what follows we assume that $U_1 \neq U_2$. If P_1 and P_2 are edge-disjoint, we denote by L_1 and L_2, respectively, the endpoints of the link $\mathcal{L} = \mathcal{L}(P_1, P_2)$. Otherwise, we set $\mathcal{L} = \varnothing$ and denote by $[M, N]$ the intersection of the projections; furthermore, in this case we assume w.l.o.g. that if $P_1 = [M_1, N_1]$ and $P_2 = [M_2, N_2]$, then the paths $[M_1, M_2]$ and $[N_1, N_2]$ do not intersect and $M \in [M_1, M_2]$, $N \in [N_1, N_2]$. In both cases we denote by $\mathcal{T}$ the minimal subtree of $\mathcal{H}_S$ containing both P_1 and P_2. Recall that we put a hat over any notation (e.g., $\widehat{\mathcal{H}}_S$ or $\widehat{\mathcal{F}}_S$) to denote the corresponding object after edge insertion.

The main concern of this section is to distinguish, in terms of the connectivity carcass, the S-mincuts that do not separate U_1 from U_2, since these are exactly the S-mincuts that are preserved under the insertion.

The transformations of the skeleton under edge insertion are as follows.

THEOREM 7.1. *If $\mathcal{L}$ contains at least one structural edge, then all the nodes and structural edges of $\mathcal{L}$ are contracted to a single new node; otherwise the skeleton remains the same.*

Proof. Let $\widehat{\mathcal{H}}_S$ be obtained from $\mathcal{H}_S$ by contraction of $\mathcal{L}$ into a new node N (or $\widehat{\mathcal{H}}_S = \mathcal{H}_S$ if $\mathcal{L}$ does not contain any edges). We define $\widehat{\varphi}_S$ as follows: $\widehat{\varphi}_S = \varphi_S$ if $\varphi_S \notin \mathcal{L}$, and $\widehat{\varphi}_S = N$ otherwise.

Let ε be an arbitrary structural edge in $\mathcal{H}_S$.

If $\varepsilon \notin \mathcal{T}$, then, by Theorem 6.7, both U_1 and U_2 lie in the same terminal of the strip $\mathcal{W}(\varepsilon)$. Hence, in this case the new edge does not affect any cut of this strip.

If $\varepsilon \in \mathcal{T}$, but $\varepsilon \notin \mathcal{L}$, then at least one of the units U_1 and U_2 is a stretched unit in $\mathcal{W}(\varepsilon)$. Hence, in this case at least one tight cut of $\mathcal{W}(\varepsilon)$ is not affected by the new edge, and thus the corresponding 2-partition of S is still an S-mincut.

Finally, if $\varepsilon \in \mathcal{L}$, then, by Theorem 6.7, U_1 and U_2 lie in distinct terminals of $\mathcal{W}(\varepsilon)$. Hence, in this case no cuts represented by $\mathcal{W}(\varepsilon)$ survive, and thus the corresponding 2-partition of S ceases to be an S-mincut.

Thus, by Theorem 5.6, $\widehat{\mathcal{H}}_S$ is indeed the skeleton of the graph $G \cup (u_1, u_2)$. □

The transformations of the flesh under edge insertion are as follows.

THEOREM 7.2. *The new flesh is obtained by contraction of a certain subset of units of $\mathcal{F}_S$ into a single new unit U^{new}. This subset contains*

(i) *U_1 and U_2;*

(ii) *all the units lying on coherent paths between U_1 and U_2;*

(iii) *all the units U such that $\pi_S(U) \subseteq \mathcal{L}$;*

(iv) *in the case when an endpoint K of P_i is the unique common point of P_i and $P_j \cup \mathcal{L}$, $i, j \in \{1, 2\}$, $i \neq j$, all the units U such that $U \in \mathcal{R}_K(U_i)$ and $\pi_S(U) \subseteq P_i \cup \mathcal{L}$.*

Proof. First, let us prove the following statement concerning the initial graph G.

CLAIM A. *If a unit U satisfies at least one of the conditions of Theorem* 7.2, *then any S-mincut that separates U_1 from U separates also U_1 from U_2.*

Indeed, if $U = U_2$, then the claim is trivial.

Let U belong to a coherent path between U_1 and U_2. Let $\mathcal{C}$ be an S-mincut that separates U_1 from U but fails to separate U_1 from U_2; then we get a coherent path with both endpoints on one side of $\mathcal{C}$ and an intermediate unit U on the other side, a contradiction.

Let $\pi_S(U) \subseteq \mathcal{L}$, and let $\mathcal{C}$ be an S-mincut separating U_1 from U. By Theorem 6.7, the structural edge ε representing $\mathcal{C}$ in $\mathcal{H}_S$ lies either in $\mathcal{L}$ or in P_1. If $\varepsilon \in \mathcal{L}$, then U_1 and U_2 are separated by any cut represented by ε (see the proof of Theorem 7.1). If $\varepsilon \in P_1$, then U_1 is a stretched unit of $\mathcal{W}(\varepsilon)$, while both U and U_2 lie in the same terminal of $\mathcal{W}(\varepsilon)$, and the claim follows.

Assume now that an endpoint K of P_1 is the unique common point of P_1 and $P_2 \cup \mathcal{L}$, and besides, $U \in \mathcal{R}_K(U_1)$ and $\pi_S(U) \subseteq P_1 \cup \mathcal{L}$. Let $\mathcal{C}$ and ε be as in the previous case; then, by Theorem 6.7, ε belongs to $P_1 \cup \mathcal{L}$. If $\varepsilon \in \mathcal{L}$, we proceed as above. If $\varepsilon \in P_1 \cap \pi_S(U)$, then both U_1 and U are stretched units of $\mathcal{W}(\varepsilon)$. Moreover, U_2 lies in one of the terminals of $\mathcal{W}(\varepsilon)$, and, by Theorem 6.10, U lies in the cone of U_1 containing this terminal. Thus, $\mathcal{C}$ separates U_1 from U_2. If ε lies in P_1, but not in $\pi_S(U)$, then, by Theorem 6.10, both U and U_2 lie in the same terminal of $\mathcal{W}(\varepsilon)$, and again $\mathcal{C}$ separates U_1 from U_2.

Finally, if an endpoint K of P_2 is the unique common point of P_2 and $P_1 \cup \mathcal{L}$, and besides, $U \in \mathcal{R}_K(U_2)$ and $\pi_S(U) \subseteq P_2 \cup \mathcal{L}$, one has to interchange the roles of U_1 and U_2 in the above reasoning.

Therefore, Claim A is proved, and as an immediate corollary we get that all the units listed in Theorem 7.2 are indeed contracted into a single new unit.

Let us now prove another statement concerning the initial graph G.

CLAIM B. *If a unit U violates all the conditions of Theorem* 7.2 *and $U' \neq U$ is an arbitrary unit, then there exists an S-mincut that separates U from U' and does not separate U_1 from U_2.*

Indeed, let $P = \pi_S(U)$, let Q be the cell containing U, and assume first that U is a stretched unit.

Suppose that there exists a structural edge $\varepsilon \in P$ that does not belong to $\mathcal{T}$. Then U is a stretched unit in $\mathcal{W}(\varepsilon)$, while both U_1 and U_2 lie in the same terminal of $\mathcal{W}(\varepsilon)$ (by Theorem 6.7). Therefore, any S-mincut that separates U from U' in $\mathcal{W}(\varepsilon)$ satisfies the assertion of the claim.

From now on we may assume that P lies entirely in $\mathcal{T}$. By the assumptions of the Theorem, P does not lie entirely in $\mathcal{L}$; hence, there exists an edge $\varepsilon \in P$ that belongs (w.l.o.g.) to P_1. If ε belongs also to P_2, then all the three units U, U_1, and U_2 are stretched in $\mathcal{W}(\varepsilon)$. If U_1 and U_2 lie in the opposite cones of U in $\mathcal{W}(\varepsilon)$, then U lies on a coherent path from U_1 to U_2, which is prohibited by the assumptions of the claim. So, there exists a cone $\mathcal{R}$ of U in $\mathcal{W}(\varepsilon)$ that contains neither U_1 nor U_2. If $\mathcal{R}$ does not contain U' as well, then the cut of $\mathcal{W}(\varepsilon)$ defined by $\mathcal{R}$ satisfies the assertion of the claim. Otherwise, we use instead of $\mathcal{R}$ the cone $\mathcal{R}'$ of U' in $\mathcal{W}(\varepsilon)$ that is contained in $\mathcal{R}$.

If $\varepsilon \notin P_2$, then in $\mathcal{W}(\varepsilon)$ both U and U_1 are stretched, while U_2 lies in a terminal. Let $\mathcal{R}$ be the cone of U in $\mathcal{W}(\varepsilon)$ that contains the other terminal. If U_1 does not belong to $\mathcal{R}$, then the cut of $\mathcal{W}(\varepsilon)$ defined by $\mathcal{R}$ (or by $\mathcal{R}'$, as above, if $U' \in \mathcal{R}$) satisfies the assertion of the claim.

Now let $U_1 \in \mathcal{R}$; this means, in particular, that U belongs to some cone of U_1 in $\mathcal{W}(\varepsilon)$, and thus in $\mathcal{F}_S$, say, $U \in \mathcal{R}_K(U_1)$. By Theorem 6.10, this means that Q is visible from Q_1 in direction K. Therefore, if ε_1 is the edge of P_1 incident to K, then $\varepsilon_1 \in P$. If $\varepsilon_1 \in P_2$, we proceed in the same way as above; so, we may assume that $\varepsilon_1 \notin P_2$. Let us consider the strip $\mathcal{W}(\varepsilon_1)$. Both U and U_1 are stretched in $\mathcal{W}(\varepsilon_1)$; moreover, U lies in a cone $\mathcal{R}_1$ of U_1, while U_2 lies in a terminal. If this terminal does not belong to $\mathcal{R}_1$, we consider the cone $\mathcal{R}$ of U that lies inside $\mathcal{R}_1$. It is easy to see that the cut of $\mathcal{W}(\varepsilon_1)$ defined by $\mathcal{R}$ (or by $\mathcal{R}'$, as above, if $U' \in \mathcal{R}$) satisfies the assertion of the claim.

Finally, let the terminal containing U_2 in $\mathcal{W}(\varepsilon)$ lie in $\mathcal{R}_1$. It follows immediately from Theorem 6.7 that in this case K is the unique common point of P_1 and $P_2 \cup \mathcal{L}$. By the assumptions of the claim, P does not lie entirely in $P_1 \cup \mathcal{L}$; hence, there exists an edge ε_2 of P such that $\varepsilon_2 \in P_2$. Evidently, in $\mathcal{W}(\varepsilon_2)$ both U and U_2 are stretched, while U_1 lies in a terminal. Let U lie outside the cone of U_2 in $\mathcal{W}(\varepsilon_2)$ that contains this terminal, and let $\mathcal{R}$ be the cone of U in $\mathcal{W}(\varepsilon_2)$ that contains the other terminal; then the cut of $\mathcal{W}(\varepsilon_2)$ defined by $\mathcal{R}$ (or by $\mathcal{R}'$, as above, if $U' \in \mathcal{R}$) satisfies the assertion of the claim. Otherwise, we use the same reasoning as above to see that one of the endpoints of P_2 is the unique common point of P_2 and $P_1 \cup \mathcal{L}$, and hence $\mathcal{T}$ is just a path. Therefore, Q_1 and Q_2 are visible from Q in opposite directions. Thus, the coherent path from U_1 to U in $\mathcal{W}(\varepsilon_1)$ can be concatenated with a coherent path from U to U_2 in $\mathcal{W}(\varepsilon_2)$, and we get that U lies on a coherent path from U_1 to U_2, in a contradiction with the assumptions of the claim.

Assume now that U is a terminal unit, and let $P' = \pi_S(U')$. Observe that P from now on is a node of $\mathcal{H}_S$.

If $P \cap P' = \varnothing$, then, by the assumptions of the claim, there exists a structural edge ε that belongs to the link of P and P' and does not lie in $\mathcal{L}$. Evidently, the units U and U' lie in the opposite terminals of the strip $\mathcal{W}(\varepsilon)$. On the other hand, the units U_1 and U_2 cannot lie in the opposite terminals in such a strip. Therefore, one of the tight cuts in direction ε satisfies the assertion of the claim.

Now let $U' = U_1$ and P be a node of P_1. If P is not an endpoint of P_1, then at least one of the two P-tight cuts in directions of the edges of P_1 incident to P does not contain U_2 (otherwise P would belong to $\mathcal{L}$). If P is an endpoint of P_1, then the P-tight cut in direction of the edge of P_1 incident to P does not contain U_2 (for the same reason as above).

This completes the proof of Claim B, since if U' is stretched and satisfies the assumptions of the claim, then we can use the same reasoning as above interchanging

the roles of U and U', whereas if U' violates the assumptions of the claim, then any cut that separates U from U_1 separates also U from U', by Claim A.

As an immediate corollary of Claim B we get that all the units not listed in Theorem 7.2 remain uncontracted, and thus the Theorem is proved. □

The transformations of the projections under edge insertion are as follows.

THEOREM 7.3. (i) *The projection of* U^{new} *is either the new node obtained by the contraction of* $\mathcal{L}$ (*if* $\mathcal{L}$ *contains at least one structural edge*), *or* $P_1 \cap P_2$ *otherwise.*

(ii) *The intersection with* $\mathcal{L}$ *is deleted from the projection of any noncontracted unit.*

(iii) *For each unit* $U \in \mathcal{R}_{N_1}(U_1)$, *the common part of* $\pi_S(U)$ *and the path between* M_1 *and* M (*or* L_1) *is deleted from* $\pi_S(U)$. *Similar transformations are applied to* $\mathcal{R}_{M_1}(U_1)$, $\mathcal{R}_{N_2}(U_2)$, $\mathcal{R}_{M_2}(U_2)$.

Proof. Assume first that U is an arbitrary stretched unit of the initial graph, and let ε be an arbitrary structural edge of $\pi_S(U)$.

If $\varepsilon \notin \mathcal{T}$, then both U_1 and U_2 lie in the same terminal of $\mathcal{W}(\varepsilon)$, and thus the new edge does not affect any cut of $\mathcal{W}(\varepsilon)$. Hence, ε is preserved in the new projection of U.

If $\varepsilon \in \mathcal{L}$, then U_1 and U_2 lie in the opposite terminals of $\mathcal{W}(\varepsilon)$, and thus all the cuts of $\mathcal{W}(\varepsilon)$ do not survive. Hence, ε is deleted from the new projection of U.

If $\varepsilon \in P_1 \cap P_2$, then both U_1 and U_2 are stretched units of $\mathcal{W}(\varepsilon)$. Therefore, both tight cuts of $\mathcal{W}(\varepsilon)$ remain unaffected by the new edge, and hence ε is preserved in the new projection of U.

The only remaining case is when ε belongs to exactly one of the P_1 and P_2; assume w.l.o.g. that $\varepsilon \in [M_1, K]$, where $K = M$ if $P_1 \cap P_2 \neq \varnothing$, and $K = L_1$ otherwise. Then U_1 is a stretched unit of $\mathcal{W}(\varepsilon)$, while U_2 lies in the terminal of $\mathcal{W}(\varepsilon)$ that belongs to $\mathcal{R}_{N_1}(U_1)$ (or, more exactly, to the cone of U_1 in $\mathcal{W}(\varepsilon)$ that lies in $\mathcal{R}_{N_1}(U_1)$). Therefore, if $U \notin \mathcal{R}_{N_1}(U_1)$, then ε is preserved in the new projection of U, while otherwise it is deleted from the new projection.

Assume now that U is a terminal of $\mathcal{F}_S$; evidently, the new edge cannot convert it to a stretched unit, and thus all we have to do is to establish that the new projection of U cannot shift.

By Theorem 6.5(ii), U lies in the intersection of all the K-tight cuts, where $K = \pi_S(U)$. It follows easily from Theorem 7.1 that if ε is incident to K and $\varepsilon \notin \mathcal{L}$, then the new K-tight cut in direction ε dominates the old one. Therefore, if $K \notin \mathcal{L}$, or $K = \mathcal{L}$, then K remains to be the projection of U in the new skeleton.

Finally, let $K \in \mathcal{L}$ and $K \neq \mathcal{L}$; we denote by ε_i and ε_i' the edges of the path $[K, L_i]$ incident to K and L_i, respectively ($i = 1, 2$). By Lemma 6.1, any L_i-tight cut, except for the L_i-tight cut in direction ε_i', dominates the K-tight cut in direction ε_i. By Theorem 7.1, all the edges incident to L_i except for ε_i' are preserved in the new skeleton, and are incident to its new node. Evidently, the tight cut of the new node in any of these directions dominates the L_i-tight cut in the same direction. Thus, the U^{new} lies in the intersection of all the tight cuts of the new node. □

Remark. It follows from Theorems 7.2 and 7.3 that if a unit must be contracted, then its new projection obtained formally via Theorem 7.3 coincides with that of U^{new}.

Finally, let us describe the transformation of the inherent 2-partition under edge insertion.

THEOREM 7.4. (i) *The inherent* 2*-partitions at noncontracted units remain the same.*

(ii) *If $P_1 \cap P_2$ does not contain any edges, then the inherent 2-partition at U^{new} is trivial. Otherwise, the inherent 2-partition for U^{new} is glued from those for contracted units. Namely, the part of the star of U_1 labeled by N_1 (resp., M_1) is glued with that for U_2 labeled by N_2 (resp., M_2). For other units, two parts are glued together if the corresponding reachability cones contain the same unit out of U_1 and U_2.*

Proof. (i) Follows immediately from Theorem 7.3, since the 2-partition at a unit U of the flesh is either trivial or inherited from the 2-partition at the same unit of any strip $\widehat{\mathcal{W}}(\varepsilon)$, $\varepsilon \in \pi_S(U)$.

(ii) The first part follows immediately from Theorem 7.3.

Let $P_1 \cap P_2$ now contain at least one edge ε. By Lemma 5.3, it is enough to prove the statement for the 2-partition at U in the strip $\widehat{\mathcal{W}}(\varepsilon)$. It follows from Theorems 7.1–7.3 that to get $\widehat{\mathcal{W}}(\varepsilon)$ from $\mathcal{W}(\varepsilon)$ one has just to insert the edge (U_1, U_2) into $\mathcal{W}(\varepsilon)$ and to transform it accordingly. This transformation was already considered in section 4.2, so the statement follows. □

7.2. Local reachability cones. The transformations of the connectivity carcass described in Theorems 7.2 and 7.3 are formulated in terms of reachability cones. Therefore, to maintain the carcass, we need to maintain the cones as well. However, reachability cones in an acyclic locally orientable graph behave in a more complicated way than those in a dag. In particular, insertion of an edge may cause a reduction of a reachability cone. Indeed, if a stretched unit belonging to a cone is contracted into a terminal (see Theorem 7.2), then any coherent path passing through this unit cannot be traced behind it in the modified flesh. To avoid these difficulties, we introduce a simpler structure, called a local reachability cone, which preserves all the useful properties of reachability cones but is easier to handle.

Let $\mathcal{R}$ be a reachability cone of an arbitrary unit U. We define the *local reachability cone* $\mathcal{R}^{\mathrm{loc}}$ as the set of units $U' \in \mathcal{R}$ such that $\pi_S(U) \cap \pi_S(U')$ contains at least one edge. (In particular, the local cone of a terminal is empty.) One can check easily that local reachability cones of U are obtained from the corresponding reachability cones of U in the strip $\mathcal{W}(\pi_S(U))$ by deleting the terminals. Observe that reachability cones in the assumptions of Theorems 7.2 and 7.3 can be replaced by the corresponding local reachability cones.

Indeed, for any unit U whose projection is changed according to Theorem 7.3(iii), the intersection $\pi_S(U) \cap P_1$ contains at least one edge, and hence such a unit belongs to $\mathcal{R}^{\mathrm{loc}}_{N_1}(U_1)$ (or, similarly, to $\mathcal{R}^{\mathrm{loc}}_{M_1}(U_1)$, $\mathcal{R}^{\mathrm{loc}}_{N_2}(U_2)$, or $\mathcal{R}^{\mathrm{loc}}_{M_2}(U_2)$).

The units U distinguished by condition (iv) of Theorem 7.2 satisfy $U \in \mathcal{R}_K(U_i)$ and $\pi_S(U) \subseteq P_i \cup \mathcal{L}$. However, if $\pi_S(U) \cap P_i$ does not contain any edges, the latter inclusion implies $\pi_S(U) \subseteq \mathcal{L}$, and hence U is already distinguished by condition (iii). Otherwise $\pi_S(U) \cap P_i$ contains at least one edge, and this together with the former inclusion implies $U \in \mathcal{R}^{\mathrm{loc}}_K(U_i)$.

Finally, let us consider the units distinguished by condition (ii) of Theorem 7.2. Evidently, these units form the intersection of the two reachability cones $\mathcal{R}(U_1)$ and $\mathcal{R}(U_2)$. (For the sake of simplicity, we do not specify the labels of the cones.) Let us prove that it suffices to consider only the units lying in the intersection $\mathcal{R}^{\mathrm{loc}}(U_1) \cap \mathcal{R}^{\mathrm{loc}}(U_2)$. Indeed, if $\mathcal{R}(U_1) \cap \mathcal{R}(U_2) = \varnothing$, then $\mathcal{R}^{\mathrm{loc}}(U_1) \cap \mathcal{R}^{\mathrm{loc}}(U_2) = \varnothing$, and the assertion is trivial. Otherwise, by Theorem 6.10, the minimum tree containing P_1 and P_2 is a path, and the projection of any unit $U \in \mathcal{R}(U_1) \cap \mathcal{R}(U_2)$ belongs to this path. If $\pi_S(U)$ intersects both P_1 and P_2 at least by an edge, then $U \in \mathcal{R}^{\mathrm{loc}}(U_1) \cap \mathcal{R}^{\mathrm{loc}}(U_2)$. If $\pi_S(U)$ intersects both P_1 and P_2 at most by a node, then $\pi_S(U) \subseteq \mathcal{L}$, and U is distinguished by condition (iii) of Theorem 7.2. Finally, if $\pi_S(U)$ intersects at least

by an edge exactly one of P_1 and P_2, then the assumptions of Theorem 7.2(iv) are satisfied, and U is distinguished by condition (iv).

8. Construction and incremental maintenance of the connectivity carcass.

8.1. Construction of the connectivity carcass. We build the connectivity carcass by a recursive algorithm based straightforwardly on Lemma 6.14; in what follows we use the notation of this lemma.

First of all, we choose an arbitrary vertex $s \in S$ and find maximal flows from s to all the other vertices $t \in S$; this allows us to define λ_S as the minimal value of these flows. All the vertices $t \in S$ such that the value of a maximal flow from s to t exceeds λ_S are contracted together with s; in what follows s denotes this new vertex. We then choose an arbitrary one of the remaining vertices of S (denoted by t) and execute the following procedure.

We build the strip $\mathcal{F} = \mathcal{W}_{s,t}$, as explained in section 3.3. To build the skeleton $\mathcal{H} = \mathcal{H}_{\tilde{S}}(\tilde{G})$ we find a topological order of the units in $\mathcal{F}$. Let $U_0 \ni s, U_1, \ldots, U_k \ni t$ be the heavy units in this order; we define $\mathcal{H}$ as a k-edge path $(N_0, N_1, \ldots, N_k)$, $k \geqslant 1$.

To find the corresponding projections $\pi = \pi_{\tilde{S}}$, we execute DFS in $\mathcal{F}$ in direction U_k first from U_{k-1}, then from U_{k-2}, and so on up to U_0. In any such execution we assign N_i as a coordinate to the units found from U_i and backtrack each time when we discover a unit already visited in the previous search. To get the other coordinate we repeat the same process in direction U_0 first from U_1, then from U_2, and so on up to U_k.

Next, for each i, $0 \leqslant i \leqslant k$, we turn the unit U_i into a terminal. If U_i contains vertices of S distinct from s and t, we do the following. We build the set S' by taking all the vertices of S that do not belong to U_i, together with s and t, if they are not yet included in S'. We contract all the vertices of S' into a new vertex s^i, thus obtaining the new graph G^i and the new set S^i in it. We then scan the vertices t' in $S^i \setminus s^i$ and find a maximum flow from s^i to t'. If its value exceeds λ_S, then we just contract t' to s^i. Otherwise, we stop scanning and build the connectivity carcass $(\mathcal{H}^i, \mathcal{F}^i, \pi^i)$ of S^i in G^i by a recursive execution of the same procedure, with $s = s^i$, $t = t'$. This carcass is then merged with the current triple $(\mathcal{H}, \mathcal{F}, \pi)$, as follows.

The skeleton $\mathcal{H}$ is merged with $\mathcal{H}^i$ by identifying the node $N_i \in \mathcal{H}$ with the node $\pi^i(s^i) \in \mathcal{H}^i$.

The new terminal unit corresponding to N_i is built according to Lemma 6.14; it participates in $\mathcal{F}_S$ if the result is a nonempty set.

To merge the partitions of V into units of $\mathcal{F}$ and $\mathcal{F}^i$ we assume that each vertex $v \in V$ knows its units $U(v)$ and $U^i(v)$, and each unit knows its cardinality. Now, for each $v \in V$ we check the units $U(v)$ and $U^i(v)$. If their cardinalities are equal, we concatenate their projections into the new projection of $U(v)$ and cancel $U^i(v)$. Otherwise, we set the new $U(v)$ to be the smallest of the two units involved, preserve its projection, and cancel the other unit.

Notice that the sets $S^i \setminus s^i$, $0 \leqslant i \leqslant k$, do not intersect since they belong to the disjoint vertex sets U_i, respectively.

The above-described algorithm implies the following result (the proof is omitted).

THEOREM 8.1. *The connectivity carcass of an arbitrary vertex subset S can be constructed in time dominated by the complexity of $2\sigma - 2$ max-flow computations in G.*

8.2. Incremental maintenance of the connectivity carcass. Let n be the number of vertices in the initial graph G, $\tilde{\sigma} \leqslant \sigma$ be the number of (λ_S+1)-connectivity classes in S, $\tilde{n} \leqslant n$ be the initial number of flesh units, $\tilde{m} \leqslant m$, $\tilde{m} = O(\lambda_S \tilde{n})$, be the initial number of flesh edges.

We maintain the connectivity carcass under an arbitrary sequence of updates (edge insertions) not changing the connectivity of S and queries "Are vertices $v, w \in G$ separated by an S-mincut?" denoted by $\mathrm{Sep}(v, w)$, "Show an S-mincut separating $v, w \in G$," denoted by $\mathrm{Cut}(v, w)$, and "Construct the strip $\mathcal{W}_{S_1,S_2}$ for $S_1, S_2 \subset S$," denoted by $\mathrm{Strip}(S_1, S_2)$.

In order to maintain the connectivity carcass efficiently, we propose to keep track of the projections represented by the coordinates and of certain parts of reachability cones specified below. It follows from the discussion in section 7.2 that instead of a reachability cone $\mathcal{R}$ it suffices to maintain any partial subcone of $\mathcal{R}$ that contains the corresponding local reachability cone $\mathcal{R}^{\mathrm{loc}}$; we will maintain and use a certain subcone of this type named $\mathcal{R}^{\mathrm{part}}$. By Lemma 3.4, any reachability cone $\mathcal{R}$ of an arbitrary unit W, and hence $\mathcal{R}^{\mathrm{part}}$, is globally orientable. Since it is acyclic, it can be treated in the same way as a dag. In what follows we assume that W is the source in this dag. Following [I], we represent $\mathcal{R}^{\mathrm{part}}$ by a directed spanning tree rooted at W and make use of the following reachability vector of length $\tilde{n}$. Each entry of the vector takes one of the three values to distinguish the following situations: the unit is currently contained in $\mathcal{R}^{\mathrm{part}}$; the unit was never contained in $\mathcal{R}^{\mathrm{part}}$; the unit was previously deleted from $\mathcal{R}^{\mathrm{part}}$. At the initial moment we set $\mathcal{R}^{\mathrm{part}} = \mathcal{R}$; the initialization takes $O(\tilde{n}\tilde{m})$ time for all the cones together.

The skeleton is represented as follows. First, we fix an arbitrary node of the skeleton and make it the root. Second, we attach to each node a single auxiliary leaf bearing the same name. When an edge from a child to its parent is contracted, the new node acquires the name of the parent; observe that only edges of the skeleton are contracted. It follows easily by induction that at any moment the names of the auxiliary leaves attached to each skeleton node are exactly the names of the original nodes contracted into this node. The projection of any unit U is represented by the pair of auxiliary leaves corresponding to the coordinates of U; in fact, the actual projection is the path between the parents of these two leaves.

To handle the obtained rooted tree we make use of the compressed tree data structure proposed in [W92]. This data structure allows us to perform an arbitrary sequence of w nearest common ancestor (NCA) and parent queries and edge contractions in $O(w + t \log^2 t)$ time, where t is the size of the initial tree. In our case this means that we can perform NCA and parent queries in $O(1)$ amortized time with an overhead of $O(\tilde{\sigma} \log^2 \tilde{\sigma})$, since by Theorem 5.6 the size of the skeleton is $O(\tilde{\sigma})$.

The set of units is represented with the help of the standard union-find technique (see [AHU]) that allows us to perform an arbitrary sequence of w operations in $O(w + t \log t)$, where t is the size of the initial set. In our case this means that we can perform find operations in $O(1)$ worst-case time with an overhead of $O(\tilde{n} \log \tilde{n})$. Each side of a unit W is represented as a linked list of edges, and the whole such list is labeled by the corresponding coordinate of W.

When a new edge is inserted, it takes $O(1)$ worst-case time to locate the units U_1 and U_2 containing its endpoints. The nontrivial case, when U_1 and U_2 are distinct, can occur at most $\tilde{n}$ times (since this causes the contraction of these units). For each nontrivial case, we do the following.

On the first stage we analyze the relative position of the projections $\pi_S(U_1)$ and

$\pi_S(U_2)$. If they are edge-disjoint, we find their link and, according to Theorem 7.1, contract the edges of the link. Otherwise, we find their intersection and establish the correspondence between the endpoints of the intersection and the endpoints of the projections (see the beginning of section 7.1 for details). It is easy to check that in order to find the link or the intersection of two projections, and to establish the latter correspondence, it suffices to apply a constant number of NCA and parent queries. Since we spend $O(1)$ amortized time for each of $O(\tilde{n})$ queries with an overhead of $O(\tilde{\sigma}\log^2\tilde{\sigma})$ (covering all contractions), the total time for the first stage is $O(\tilde{\sigma}+\tilde{n}+\tilde{\sigma}\log^2\tilde{\sigma}) = O(\tilde{n}+\tilde{\sigma}\log^2\tilde{\sigma})$.

On the second stage we change the projections. By Theorem 7.3(i), we assign to U^{new} either the contracted node of the skeleton or the path $\pi_S(U_1)\cap\pi_S(U_2)$. Evidently, the total cost of such an assignment is $O(\tilde{n})$. Further, since projections are represented by coordinates, contractions in the skeleton do not imply explicit changes in this representation. Thus, projection changes prescribed by Theorem 7.3(ii) do not require any time.

Finally, to find units whose coordinates must be changed according to Theorem 7.3(iii), we execute DFS in the spanning tree of each of the four partial cones involved and backtrack each time when we reach a unit whose projection no more intersects a certain path in the skeleton; see Theorem 7.3(iii) for details (e.g., for the case of $\mathcal{R}_{N_1}(U_1)$ this path is either $[M_1, M]$ or $[M_1, L_1]$). The validity of such a backtracking is justified by Theorem 6.10. It is easy to see that all the operations performed on the second stage, other than the analysis of the relative position of two paths in the skeleton, require $O(1)$ amortized time per unit. The latter analysis can be performed in $O(1)$ amortized time per unit as well by means of NCA and parent queries as above.

To estimate the total amount of time required by projection changes in this case, let us assign a weight to each edge in the flesh. The weight of an edge is equal to the sum of the lengths of the projections of its endpoints. (The length of a projection is just the number of structural edges in it.) Observe that each time when an edge is scanned, the projection of its tail is truncated. Hence, the length of the projection of the tail strictly decreases, and the same occurs to the weight of the edge considered. Since the initial weight of each edge is $O(\tilde{\sigma})$, we see that the overall number of projection changes is $O(\tilde{\sigma}\tilde{n})$, and that of edge scans is $O(\tilde{\sigma}\tilde{m})$. Thus, the total amount of work on the second stage is $O(\tilde{n}+\tilde{\sigma}\tilde{n}+\tilde{\sigma}\tilde{m}+\tilde{\sigma}\log^2\tilde{\sigma}) = O(\tilde{\sigma}\tilde{m})$.

The contractions in the flesh are performed on the third stage. Evidently, the total cost of contractions described in Theorem 7.2(i) is $O(\tilde{n}\log\tilde{n})$. The set of units that must be contracted according to Theorem 7.2(ii) is the intersection of two opposite local reachability cones (see the discussion in section 7.2); to find these units we just scan the reachability vectors of the corresponding partial cones. Since each contraction itself takes $O(\log\tilde{n})$ amortized time, the total amount of time for contractions in this case is $O(\tilde{n}^2+\tilde{n}\log\tilde{n}) = O(\tilde{n}^2)$.

To find the units that must be contracted according to Theorem 7.2(iii) and (iv), we scan all the units and check for each of them whether its new projection coincides with this of the contracted unit (see the remark after Theorem 7.3). This takes $O(1)$ amortized time per unit (by using a constant number of parent queries) and thus $O(\tilde{n})$ time for the whole scan. Each contraction itself takes $O(\log\tilde{n})$ amortized time. Since each occurrence of this case leads to a contraction in the skeleton, the total number of such occurrences is $O(\tilde{\sigma})$. Hence, the total amount of work in this case is $O(\tilde{\sigma}\tilde{n}+\tilde{n}\log\tilde{n}+\tilde{\sigma}\log^2\tilde{\sigma}) = O(\tilde{\sigma}\tilde{n}+\tilde{n}\log\tilde{n})$.

To find the 2-partition at the new unit, provided it is stretched, we establish the correspondence between the labels at the sides of the stars involved and concatenate the corresponding linked lists. This can be done in $O(\tilde{n}+\tilde{\sigma}\log^2\tilde{\sigma})$ time with the same technique as above.

On the fourth stage we change partial reachability cones. Let $\mathcal{R}^{\text{part}}$ be a partial cone of some unit W. As it was described above, $\mathcal{R}^{\text{part}}$ is represented by a spanning tree rooted at W and the three-valued reachability vector. Since the set of the contracted units is already known, we scan the corresponding entries of the reachability vector and find the list $L(\mathcal{R}^{\text{part}})$ of the units in $\mathcal{R}^{\text{part}}$ that should be contracted. Since each unit is contracted at most once, the total amount of work for these scans is $O(\tilde{n}^2)$. If the list $L(\mathcal{R}^{\text{part}})$ is empty, the cone $\mathcal{R}^{\text{part}}$ is not changed. Otherwise we proceed as follows (see the proof of Theorem 8.2 for support).

If U^{new} is a terminal, we just delete all the units belonging to $L(\mathcal{R}^{\text{part}})$, together with their subtrees, from the tree representing $\mathcal{R}^{\text{part}}$, and change the reachability vector accordingly. Since each unit is deleted at most once from each tree, and the total number of trees is $O(\tilde{n})$, the total amount of work in this case is $O(\tilde{n}^2)$.

If U^{new} is not a terminal, we do the following. For each unit U belonging to the list $L(\mathcal{R}^{\text{part}})$ and distinct from W, we check whether its parent belongs to the list; if it does, we contract the edge from U to its parent, and if not, we mark this edge. If there are marked edges, we identify all their tails into a new unit W' and delete all marked edges except for an arbitrary one. Finally, if exactly one of U_1 and U_2 (say, U_1) never belonged to $\mathcal{R}^{\text{part}}$, the other one (in our case, U_2) belongs to $\mathcal{R}^{\text{part}}$, and $\pi_S(W)\cap(\pi_S(U_1)\cap\pi_S(U_2))$ contains at least one edge, we add a certain subtree to $\mathcal{R}^{\text{part}}$ by identifying its root with W', if defined, or with W otherwise. To obtain this subtree we execute DFS in the spanning tree of the corresponding partial cone of U_1 and backtrack each time when we get to a unit that belongs to $\mathcal{R}^{\text{part}}$ or was previously deleted from $\mathcal{R}^{\text{part}}$. Clearly, the total amount of work in this case is proportional to the number of edge operations, that is, contractions, marking and deletions of edges currently in $\mathcal{R}^{\text{part}}$, and scans of edges currently not in $\mathcal{R}^{\text{part}}$. Each edge in $\mathcal{R}^{\text{part}}$ is contracted or marked and deleted at most once. Besides, each time when $\mathcal{R}^{\text{part}}$ is updated, at most one edge is marked and not deleted. Finally, each scan turns an edge currently not in the entire cone $\mathcal{R}$ to an edge in $\mathcal{R}$. Moreover, such an edge never belonged to $\mathcal{R}$ before, and hence each edge is scanned at most once. Therefore, the total number of edge operations is $O(\tilde{n}\tilde{m})$.

To find whether two vertices of G are separated by an S-mincut, it suffices to find the corresponding units (in $O(1)$ worst-case time) and to check whether they do not coincide. If this is the case, the partition of S corresponding to such a cut can be obtained via Lemma 6.13 using Theorem 6.7. Finding a cut of the skeleton separating two given nodes and finding an edge of a projection takes $O(1)$ amortized time (with the help of NCA queries). To find a tight cut as in Lemma 6.13(ii), we check for all units the inclusion of Theorem 6.7 in $O(1)$ amortized time per unit.

To obtain the strip $\mathcal{W}_{S_1,S_2}$ of Theorem 6.4 we execute $O(|S_1|+|S_2|)$ NCA queries to check whether $\mathcal{T}(S_1)\cap\mathcal{T}(S_2)=\varnothing$, and if this is the case, to find the link $\mathcal{L}(S_1,S_2)$. Next, for each unit U we check whether its projection intersects the path $\mathcal{L}(S_1,S_2)$ by an edge. If this is not the case, a few NCA queries give us the endpoint $N\in\mathcal{L}(S_1,S_2)$ of the link of $\mathcal{L}(S_1,S_2)$ and $\pi_S(U)$. The unit U belongs to the contracted subset corresponding to N. To obtain the orientation, it suffices to execute DFS by coherent paths from any terminal unit of the strip.

The complexity of the above-described algorithm is given by the following statement.

THEOREM 8.2. *The connectivity carcass of an arbitrary vertex subset S can be maintained in $O(\tilde{n}\tilde{m}+u+q_{\text{sep}}+q_{\text{cut}}\tilde{n}+q_{\text{strip}}\tilde{m})$ time for an arbitrary sequence of u edge insertions preserving the value of λ_S, q_{sep} queries $\text{Sep}(v,w)$, q_{cut} queries $\text{Cut}(v,w)$, and q_{strip} queries $\text{Strip}(S_1,S_2)$. Moreover, each query $\text{Sep}(v,w)$ can be answered in $O(1)$ worst-case time.*

Proof. It follows from the discussion above that the only thing one has to check is that partial reachability cones are maintained properly, that is, that at any moment they remain intermediate between local reachability cones and ordinary ones. Since for a terminal both the local cone and its partial cone maintained by the algorithm are trivial, we assume from now on that W is a stretched unit and remains stretched in the modified flesh.

It is convenient to use the same notation as in section 7, that is, $P_1=[M_1,N_1]$ and $P_2=[M_2,N_2]$ are the projections of U_1 and U_2, respectively; if U^{new} is stretched, then $[M,N]$ is their intersection (it contains at least one edge), the paths $[M_1,M_2]$ and $[N_1,N_2]$ are disjoint, and $M\in[M_1,M_2]$, $N\in[N_1,N_2]$. Besides, $\widehat{\pi}_S(U)$ stands for the modified projection of U, provided U exists in the modified flesh, or for the projection of U^{new} otherwise. In the latter case U is one of the units constituting U^{new}, and hence is implicitly contained in it as a set of vertices, which justifies our notation. In the same sense, we say that a cone is increased upon edge insertion if the set of vertices that constitute its units increases.

Let us prove first the following statement.

LEMMA 8.3. *If an edge insertion takes a unit U out of a local cone $\mathcal{R}^{\text{loc}}$ of W, then U will never belong to $\mathcal{R}^{\text{loc}}$ again.*

Proof. We prove, moreover, that if $U\in\mathcal{R}^{\text{loc}}$ and $U\notin\widehat{\mathcal{R}}^{\text{loc}}$, then already $\widehat{\pi}_S(W)$ and $\widehat{\pi}_S(U)$ intersect at most by a node; since the projections can only shrink upon edge insertions, this would mean that U never belongs to $\mathcal{R}^{\text{loc}}$ again. Assume to the contrary that $\widehat{\pi}_S(W)$ and $\widehat{\pi}_S(U)$ intersect at least by an edge. Then $U\notin\widehat{\mathcal{R}}^{\text{loc}}$ can occur only if U^{new} is a terminal and each coherent path from W to U in $\mathcal{F}_S$ passes through at least one unit that constitutes this terminal. Let U' be such an intermediate unit on a coherent path from W to U, and let $\widehat{\pi}_S(U')=N'$. It follows immediately from Theorem 6.10 that $\widehat{\pi}_S(W)$ and $\widehat{\pi}_S(U)$ belong to distinct branches of $\widehat{\mathcal{H}}_S$ hanging at N', so they intersect at most by a node, a contradiction. ∎

Remark. Observe that the monotonicity property of Lemma 8.3 does not hold for the cone $\mathcal{R}(W)$ itself. For example, let G be a graph shown on Figure 8.1(a). Assume that $S=\{x,y,z\}$; then $\mathcal{F}_S$ is as shown on Figure 8.1(b), and hence $U\in\mathcal{R}(W)$. Upon insertion of the edge (v,y) the flesh changes as shown on Figure 8.1(c); here $U\notin\mathcal{R}(W)$. Finally, upon subsequent insertion of the edge (y,u) the flesh changes as shown on Figure 8.1(d), and again $U\subset U''\in\mathcal{R}(W)$.

It follows immediately from Lemma 8.3 that when maintaining $\mathcal{R}^{\text{part}}$, one does not need to include in it anew the units that were previously deleted from it.

Observe that exactly the same reasoning proves that if U^{new} is a terminal, then the whole subtree rooted at any unit in $L(\mathcal{R}^{\text{part}})$ does not belong to the corresponding $\widehat{\mathcal{R}}^{\text{loc}}$. Thus, the case when the new unit is a terminal is completed.

Assume now that U^{new} is stretched; clearly, in this case $\mathcal{R}$ is not decreased. First of all, observe that if $U',U''\in L(\mathcal{R}^{\text{part}})$ and there exists a coherent path from W to U'' passing through U', then all the units on this path lying between U' and U'' belong to $L(\mathcal{R}^{\text{part}})$ as well. Indeed, let U be such a unit, and consider an arbitrary

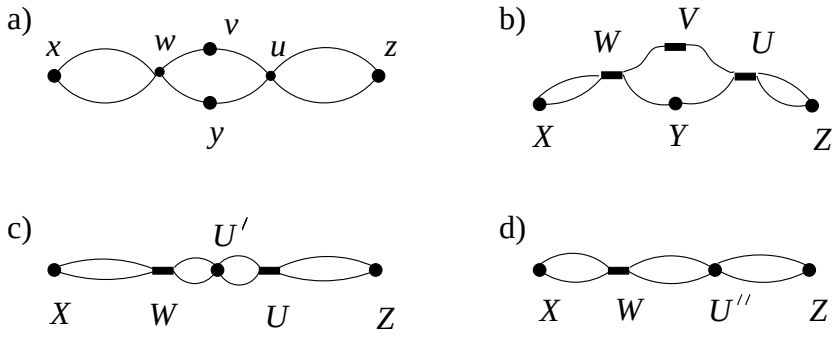


FIG. 8.1. *Nonmonotonicity of reachability cones.*

S-mincut separating U, say, from U'. It follows immediately from Theorem 5.5(ii) that the same S-mincut separates also U' from U'', and hence it no more exists after the edge insertion. Therefore, U is contracted into U^{new} as well. This means that the units in $L(\mathcal{R}^{\text{part}})$ form a set of subtrees in the tree representing $\mathcal{R}^{\text{part}}$ in such a way that no root of a subtree is a descendant of another root. It follows immediately that the contraction-marking-deletion-identifying procedure described in the algorithm is well defined, and that the tree thus obtained represents the part of $\widehat{\mathcal{R}}^{\text{part}}$ containing all the vertices constituting $\mathcal{R}^{\text{part}}$, and thus $\mathcal{R}^{\text{loc}}$. Clearly, if $\mathcal{R}^{\text{loc}}$ does not increase, we are done.

Let us consider the case when $\mathcal{R}^{\text{loc}}$ is increased. Evidently, this happens if and only if the new unit belongs to $\widehat{\mathcal{R}}^{\text{loc}}$ and at least one of U_1 and U_2 (say, U_1) does not belong to $\mathcal{R}^{\text{loc}}$. Let us prove the following statement.

LEMMA 8.4. *The conditions $U^{\text{new}} \in \widehat{\mathcal{R}}^{\text{loc}}$ and $U_1 \notin \mathcal{R}^{\text{loc}}$ are equivalent to the following three: $U_1 \notin \mathcal{R}^{\text{part}}$, $U_2 \in \mathcal{R}^{\text{part}}$, and $\pi_S(W) \cap (P_1 \cap P_2)$ contains at least one edge.*

Proof. Indeed, $U^{\text{new}} \in \widehat{\mathcal{R}}^{\text{loc}}$ implies that $\widehat{\pi}_S(W) \cap (P_1 \cap P_2)$ contains at least one edge, therefore, the same is true for $\pi_S(W) \cap (P_1 \cap P_2)$ (since projections can only shrink). Next, if $\pi_S(W) \cap (P_1 \cap P_2)$ contains at least one edge and $U_1 \notin \mathcal{R}^{\text{loc}}$, then $U_1 \notin \mathcal{R}$; indeed, $U_1 \in \mathcal{R}$ would imply together with $U_1 \notin \mathcal{R}^{\text{loc}}$ that $\pi_S(W)$ intersects P_1 at most by a node, a contradiction. Clearly, $U_1 \notin \mathcal{R}$ implies $U_1 \notin \mathcal{R}^{\text{part}}$. The same reasoning with U_1 replaced by U_2 shows that if $U_2 \notin \mathcal{R}^{\text{part}}$, then $U_2 \notin \mathcal{R}$. Finally, to get the condition $U_2 \in \mathcal{R}^{\text{part}}$, it remains to rule out the case $U_1, U_2 \notin \mathcal{R}$. It follows easily from Theorem 7.2 that the units constituting U^{new} are, apart from U_1 and U_2, exactly those lying on coherent paths between U_1 and U_2. Therefore, $U_1, U_2 \notin \mathcal{R}$ together with $U^{\text{new}} \in \widehat{\mathcal{R}}^{\text{loc}}$ would imply the existence of a unit $U \in \mathcal{R}$ lying on such a path; hence, extending a coherent path from W to U behind U we can get either to U_1 or to U_2, a contradiction.

In the other direction, $U_1 \notin \mathcal{R}^{\text{part}}$ implies $U_1 \notin \mathcal{R}^{\text{loc}}$ and $U_2 \in \mathcal{R}^{\text{part}}$ implies $U^{\text{new}} \in \widehat{\mathcal{R}}$. By Theorem 7.3, any edge contained in $\pi_S(W) \cap (P_1 \cap P_2)$ is contained also in $\widehat{\pi}_S(W) \cap (P_1 \cap P_2)$, and hence, the existence of an edge in the former intersection together with $U^{\text{new}} \in \widehat{\mathcal{R}}$ implies $U^{\text{new}} \in \widehat{\mathcal{R}}^{\text{loc}}$. □

In fact, in the algorithm, instead of checking condition $U_1 \notin \mathcal{R}^{\text{part}}$, we check whether U_1 has never belonged to $\mathcal{R}^{\text{part}}$. Indeed, if U_1 currently does not belong to $\mathcal{R}^{\text{part}}$ but has belonged to it previously, then by Lemma 8.3 we do not need to include U_1 into $\widehat{\mathcal{R}}^{\text{part}}$.

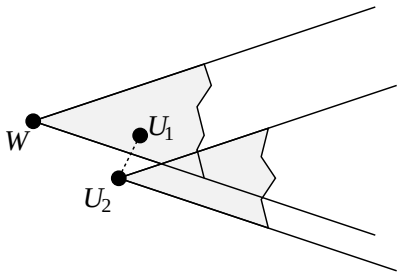


FIG. 8.2. *To the dynamics of* $\mathcal{R}^{\text{part}}$.

Let us describe now the additional units acquired by $\widehat{\mathcal{R}}^{\text{loc}}$. Assume w.l.o.g. that the cones of W under consideration are in direction N_2. (By Theorem 6.10 this makes sense, since $U_2 \in \mathcal{R}^{\text{part}}$.) Let us prove that all the additional units belong to $\mathcal{R}^{\text{loc}}_{N_1}(U_1)$ (see Figure 8.2). Indeed, all such units evidently belong to $\mathcal{R}_{N_1}(U_1)$. If $U \in \mathcal{R}_{N_1}(U_1)$ but $U \notin \mathcal{R}^{\text{loc}}_{N_1}(U_1)$, then $\pi_S(U)$ does not have edges in common with $\pi_S(U_1)$ and lies behind N_1 with respect to M_1. However, by Theorem 7.3(iii), the part of $\pi_S(W)$ lying behind N_1 with respect to M_1 is deleted in the modified carcass. Hence, $\widehat{\pi}_S(W)$ intersects $\widehat{\pi}_S(U)$ at most by a node, and thus $U \notin \widehat{\mathcal{R}}^{\text{loc}}$. The backtracking rule in the cone $\mathcal{R}^{\text{loc}}_{N_1}(U_1)$ is justified by Lemma 8.3. □

8.3. Maintenance of the cell structure. As one can easily see from the description of the algorithm in section 8.2, the most time-consuming problem is to maintain the flesh, namely, the distribution of the vertices of the initial graph among units and reachability between units. Below we propose a less detailed, cell-oriented approach. It allows us to reduce the complexity of maintenance at the expense of an increase in the reaction time for certain separation queries: we guarantee the same $O(1)$ worst-case time only in the cases when at least one of the vertices in question belongs to S. To avoid the time-consuming maintenance of reachability cones, we suggest to maintain, instead of the actual flesh, a weaker contraction of the initial flesh, called the *preflesh*; the actual flesh can be obtained from the preflesh by contraction of its strongly connected components. At any moment the partition of V into units represented by our data structure (henceforth, *preunits*) is, in a sense, intermediate: it is a refinement of the true partition, while the initial partition is a refinement of this partition. Though this approach does not guarantee the proper distribution of vertices among units, it keeps track of the distribution of vertices of G among cells.

More exactly, upon inserting a new edge we execute all the contractions as prescribed only if the resulting unit is a terminal; otherwise, we contract only the two preunits containing the endpoints of the inserted edge, as described in Theorem 7.2(i). Thus, the preflesh can be not acyclic but is a coherent locally orientable graph, and the true units are just its strongly connected components. Besides, we maintain only the 2-partitions at the preunits, but not reachability cones of any kind. The set of the preunits, the 2-partitions at each one of them, and the skeleton are represented in the same way as in the previous algorithm.

The preliminary stage of the new algorithm (distinguishing between trivial and nontrivial edge insertions) almost coincides with that of the previous one and has the same complexity $O(u)$. The only difference is that we locate not the units, but the

preunits U_1^p and U_2^p that contain the endpoints of the new edge.

The first stage of the new algorithm (update of the skeleton) coincides literally with that of the previous one and has the same complexity $O(\tilde{n} + \tilde{\sigma}\log^2\tilde{\sigma})$.

The only difference on the second stage (update of projections) is related to the case of units whose coordinates must be changed according to Theorem 7.3(iii). Instead of scanning the cones, we scan the two reachability subgraphs of U_1^p and U_2^p. It is done by executing a DFS four times (in both directions for both preunits) with backtracking each time when we reach a preunit whose projection no longer intersects a certain path in the skeleton. It is easy to see that all the reasoning involving weights of edges remains valid in this case; hence the total amount of work on the second stage is $O(\tilde{\sigma}\tilde{m})$.

The third stage of the new algorithm (update of the preflesh) is somewhat different from the corresponding stage of the previous one. If the new unit (and thus the new preunit) is not a terminal, we contract only the preunits U_1^p and U_2^p. The 2-partition at the new preunit is maintained exactly as in the previous algorithm with the help of labels, and the total amount of time for this case is $O(\tilde{n}\log\tilde{n} + \tilde{\sigma}\log^2\tilde{\sigma})$.

If the new unit (and thus the new preunit) is a terminal, we just contract all the preunits whose new projection coincides with the projection of the new terminal (see the remark after Theorem 7.3). Thus, the total amount of time in this case is $O(\tilde{\sigma}\tilde{n} + \tilde{n}\log\tilde{n} + \tilde{\sigma}\log^2\tilde{\sigma}) = O(\tilde{\sigma}\tilde{n} + \tilde{n}\log\tilde{n})$.

To find whether two vertices of G are separated by an S-mincut, we find the corresponding preunits (in $O(1)$ worst-case time). If they coincide, then such a cut does not exist. Otherwise, we check their projections. If the projections differ, then such a cut exists and is a cell cut. To find it we analyze the projections of all the preunits and verify the inclusion of Theorem 6.7. This can be done in $O(\tilde{n})$ amortized time by means of NCA queries as in the previous algorithm.

If the projections coincide, then the two preunits belong to the same nonterminal cell. In this case we do not know at once whether a cut in question exists or not; thus we answer both queries simultaneously. According to Lemma 6.13, the work to be done is split into two parts. One of them consists of finding a certain cell cut and is done exactly in the same way as in the previous case and within the same time. The other part is to find a certain reachability cone of one of our preunits. It is done by scanning the reachability subgraph of this preunit. Since, unlike the previous algorithm, we have no special data structure supporting reachability cones, this is done similarly to the scanning on the second stage. Thus, the time for both queries in this case is $O(\tilde{m})$.

Finally, to obtain the strip $\mathcal{W}_{S_1,S_2}$ of Theorem 6.4 we proceed exactly as in the previous algorithm and get this strip up to contractions of strongly connected components. To get the true strip, we replace the ordinary DFS used in the previous algorithm for obtaining orientations by the extended DFS that also finds and contracts strongly connected components. The total time remains the same.

We thus get the following result.

THEOREM 8.5. *The cell structure of the connectivity carcass of an arbitrary vertex subset S can be maintained in $O(\tilde{\sigma}\tilde{m} + \tilde{n}\log\tilde{n} + u + q\tilde{m})$ time for an arbitrary sequence of u edge insertions preserving the value of λ_S and q queries* $\mathrm{Sep}(v,w)$, $\mathrm{Cut}(v,w)$, $\mathrm{Strip}(v,w)$. *Moreover, if at least one of the vertices v, w is in S, or v and w belong to distinct cells, then the query* $\mathrm{Sep}(v,w)$ *can be answered in $O(1)$ worst-case time, and the query* $\mathrm{Cut}(v,w)$ *in $O(\tilde{n})$ amortized time.*

REFERENCES

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1976.

[Be] A. Benczur, *Augmenting undirected connectivity in $\tilde{O}(n^3)$ time*, in Proceedings of the 26th ACM Symposium on Theory of Computing (STOC'94), 1994, pp. 658–667.

[Bi] R. E. Bixby, *The minimum number of edges and vertices in a graph with edge connectivity n and m n-bonds*, Networks, 5 (1975), pp. 253–298.

[DKL] E. A. Dinic, A. V. Karzanov, and M. V. Lomonosov, *On the structure of the system of minimum edge cuts in a graph*, in Studies in Discrete Optimization, A. A. Fridman, ed., Nauka, Moscow, 1976, pp. 290–306 (in Russian; for an English review see [NV]).

[DN] Ye. Dinitz and Z. Nutov, *A 2-level cactus model for the minimum and minimum+1 edge cuts in a graph and its incremental maintenance*, in Proceedings of the 27th ACM Symposium on Theory of Computing (STOC'95), 1995, pp. 509–518.

[DV1] Ye. Dinitz and A. Vainshtein, *The connectivity carcass of a vertex subset in a graph and its incremental maintenance*, in Proceedings of the 26th ACM Symposium on Theory of Computing (STOC'94), 1994, pp. 716–725.

[DV2] Ye. Dinitz and A. Vainshtein, *Locally orientable graphs, cell structures, and a new incremental algorithm for maintaining the connectivity carcass in a graph*, in Proceedings of the 6th ACM–SIAM Symposium on Discrete Algorithms (SODA'95), 1995, pp. 302–311.

[DW] Ye. Dinitz and J. Westbrook, *Maintaining the classes of 4-edge-connectivity in a graph on-line*, Algorithmica, 20 (1998), pp. 242–276.

[FF] L. R. Ford, Jr., and D. R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.

[GI] Z. Galil and G. F. Italiano, *Maintaining the 3-edge-connected components of a graph on-line*, SIAM J. Comput., 22 (1993), pp. 11–28.

[GH] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, SIAM J. Appl. Math., 9 (1961), pp. 551–570.

[GN] D. Gusfield and D. Naor, *Extracting maximal information about sets of minimum cuts*, Algorithmica, 10 (1993), pp. 64–89.

[I] G. F. Italiano, *Amortized efficiency of a path retrieval data structure*, Theoret. Comput. Sci., 48 (1986), pp. 273–281.

[LL] J. La Poutré and J. van Leeuwen, *Maintenance of transitive closure and transitive reduction of graphs*, in Proceedings of the Workshop on Graph-Theoretics Concepts in Computer Science, Lecture Notes in Comput. Sci. 314, Springer-Verlag, Berlin, 1988, pp. 106–120.

[Na] D. Naor, *private communication*, 1991.

[Nu] Z. Nutov, *Structure of cuts and cycles in graphs. Algorithms and applications*, Ph. D. thesis, Technion, Haifa, Israel, 1996.

[NGM] D. Naor, D. Gusfield, and C. Martel, *A fast algorithm for optimally increasing the edge connectivity*, SIAM J. Comput., 26 (1997), pp. 1139–1165.

[NV] D. Naor and V. Vazirani, *Representing and enumerating edge-connectivity cuts in RNC*, in Proceedings of the 2nd Workshop on Algorithms and Data Structures (WADS'91), Lecture Notes in Comput. Sci. 519, Springer-Verlag, Berlin, 1991, pp. 273–285.

[PQ] J. C. Picard and M. Queyranne, *On the structure of all minimum cuts in a network and applications*, Math. Programming Stud., 13 (1980), pp. 8–16.

[W92] J. Westbrook, *Fast incremental planarity testing*, in Proceedings of the International Symposium on Automata, Languages and Programming (ICALP'92), Lecture Notes in Comput. Sci. 623, Springer-Verlag, New York, pp. 342–353.

[W93] J. Westbrook, *Incremental algorithms for four-edge connectivity*, in the abstract of the lecture held at the CS seminar of Bell Labs on March 10, 1993.

[WT] J. Westbrook and R. E. Tarjan, *Maintaining bridge-connected and biconnected components on-line*, Algorithmica, 7 (1992), pp. 433–464.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 809–837

FORMAL-LANGUAGE-CONSTRAINED PATH PROBLEMS*

CHRIS BARRETT†, RIKO JACOB‡, AND MADHAV MARATHE†

Abstract. Given an alphabet Σ, a (directed) graph G whose edges are weighted and Σ-labeled, and a formal language $L \subseteq \Sigma^*$, the *formal-language-constrained shortest/simple path* problem consists of finding a shortest (simple) path p in G complying with the additional constraint that $l(p) \in L$. Here $l(p)$ denotes the unique word obtained by concatenating the Σ-labels of the edges along the path p. The main contributions of this paper include the following:

(1) We show that the formal-language-constrained shortest path problem is solvable efficiently in polynomial time when L is restricted to be a context-free language (CFL). When L is specified as a regular language we provide algorithms with improved space and time bounds.

(2) In contrast, we show that the problem of finding a simple path between a source and a given destination is NP-hard, even when L is restricted to fixed simple regular languages and to very simple classes of graphs (e.g., complete grids).

(3) For the class of treewidth-bounded graphs, we show that (i) the problem of finding a regular-language-constrained simple path between source and destination is solvable in polynomial time and (ii) the extension to finding CFL-constrained simple paths is NP-complete.

Our results extend the previous results in [*SIAM J. Comput.*, 24 (1995), pp. 1235–1258; *Proceedings of the* 76*th Annual Meeting of the Transportation Research Board*, 1997; and *Proceedings of the* 9*th ACM SIGACT-SIGMOD-SIGART Symposium on Database Systems*, 1990, pp. 230–242]. Several additional extensions and applications of our results in the context of transportation problems are presented. For instance, as a corollary of our results, we obtain a polynomial-time algorithm for the best k-similar path problem studied in [*Proceedings of the* 76*th Annual Meeting of the Transportation Reasearch Board*, 1997]. The previous best algorithm was given by [*Proceedings of the* 76*th Annual Meeting of the Transportation Research Board*, 1997] and takes exponential time in the worst case.

Key words. algorithms, formal languages, computational complexity, transportation planning, World Wide Web, shortest paths, query processing, multicriteria problems

AMS subject classifications. 68Q25, 68Q45, 90B06, 68R10

PII. S0097539798337716

1. Introduction.

In many path-finding problems arising in diverse areas, certain patterns of edge/vertex labels in the labeled graph being traversed are allowed/preferred, while others are disallowed. Thus, the feasibility of a path is determined by (i) its length (or cost) under well-known measures on graphs such as distance, and (ii) its associated label. The acceptable label patterns can be specified as a formal language. For example, in transportation systems with mode options for a traveler to go from source to destination, the mode selection and destination patterns of an itinerary that a route will seek to optimize can be specified by a formal language. The problem of finding label-constrained paths also arises in other application areas such as production distribution network, VLSI design, databases queries [MW95, AMM97], etc. Here, we study the problem of finding shortest/simple paths in a network subject to certain formal language constraints on the labels of the paths obtained. We

*Received by the editors April 20, 1998; accepted for publication (in revised form) September 3, 1999; published electronically July 13, 2000.
http://www.siam.org/journals/sicomp/30-3/33771.html

†Los Alamos National Laboratory, P.O. Box 1663, MS M997, Los Alamos, NM 87545 (barrett@lanl.gov, marathe@lanl.gov). The research of these authors was supported by Department of Energy contract W-7405-ENG-36.

‡Basic Research in Computer Science, Center of the Danish Research Foundation. Department of Computer Science, University of Aarhus, Ny Munkegade, bg 540, DK-8000 Aarhus C, Denmark (rjacob@brics.dk). This research was completed while the author was staying at Los Alamos National Laboratory.

illustrate the type of problems studied here by discussing prototypical application areas.

1.1. Intermodal route planning. Our initial interest in the problem studied in this paper came from our work in the TRANSIMS[1] project at the Los Alamos National Laboratory. We refer the reader to [TR+95a, AB+97, TR+95b] for a detailed description of this project.

As a part of the intermodal route planning module of TRANSIMS, our goal is to find feasible (near optimal) paths for travelers in an intermodal network (a network with several mode choices, such as train, car, etc.) subject to certain mode-choice constraints. The mode choices for each traveler are obtained by either processing the data from the microsimulation module or by certain statistical models built from real life survey data. We refer the reader to Ben-Akiva and Lerman [BaL] for a detailed discussion and references on the theory of discrete choice analysis as applied to transportation science. The following example illustrates a prototypical problem arising in this context.

Example 1. We are given a directed labeled, weighted graph G. The graph represents a transportation network with the labels on edges representing the various modal attributes (e.g., a label t might represent a rail line). Suppose we wish to find a shortest route from s to d for a traveler. This is the ubiquitous shortest path problem. But now we are also told that the traveler wants to go from s to d using the following modal choices: either he walks to the train station, then uses trains, and then walks to his destination (office), or he would like to go all the way from home to office in a car. Using t to represent trains, w to represent walking, and c to represent a car, the traveler's mode choice can be specified as $w^+t^+w^+ \cup c^*$, where $\cup, +$, and $*$ denote the usual operators used to describe regular languages.

1.2. Searching the Web. Browsing the Web to find documents of interest as well as searching a database using queries can be interpreted as graph traversals in a certain graph. From this viewpoint, one views the Web (database) as a directed (undirected), labeled graph—the nodes are URL sites (text) and the edges are hyperlinks. A query for finding a particular URL site, for instance, proceeds by browsing the network by following links and searching by sending information retrieval requests to "index servers." A serious problem that arises in the context of using pattern-matching-based search engines is that the queries cannot exploit the topology of the document network. For example, as pointed out in [Ha88],

> Content search ignores the structure of a hypermedia network. In contrast, structure search specifically examines the hypermedia structure for subnetworks that match a given pattern.

We refer the reader to the work of [MW95, AMM97, Ha88] for a more thorough discussion on this topic. A recent paper by Abiteboul and Vianu [AV99] also discusses how regular expression constrained path queries can be used to query the Web. The following example is essentially from [AMM97].

Example 2. Let G be a graph describing a hypertext document. Suppose we want to search for job opportunities for software engineers. We first query an index server to find pages that mention the keywords "employment job opportunities" and then, from each of these pages, we could follow the local paths of lengths 0, 1, or 2 to find pages that contain the keywords "software engineer." We can state the above problem

[1] TRANSIMS is an acronym for the Transportation Analysis and Simulation System. See http://transims.tsasa.lanl.gov for details.

as finding labeled paths in G with constraints on the admissible labelings. We refer to [AMM97] for a number of additional interesting queries that can be formulated in such a framework.

2. Problem formulation. The problems discussed in the above examples can be formally described as follows: Let $G(V, E)$ be an (un)directed graph. Each edge $e \in E$ has two attributes—$l(e)$ and $w(e)$. $l(e)$ denotes the label of edge e. In this paper, the label is drawn from a (fixed) finite alphabet Σ. The attribute $w(e)$ denotes the weight of an edge. Here, we assume that the weights are nonnegative integers. Most of our positive results can in fact be extended to handle negative edge weights also (if there are no negative cycles). A path p of length k from u to v in G is a sequence of edges $\langle e_1, e_2, \ldots, e_k \rangle$, such that $e_1 = (u, v_1)$, $e_k = (v_{k-1}, v)$, and $e_i = (v_{i-1}, v_i)$ for $1 < i < k$. A path is *simple* if all the vertices in the path are distinct. Given a path $p = \langle e_1, e_2, \ldots, e_k \rangle$, the weight of the path is given by $\sum_{1 \le i \le k} w(e_i)$ and the label of p is defined as $l(e_1) \cdot l(e_2) \cdots l(e_k)$. In other words the label of a path is obtained by concatenating the labels of the edges on the path in their natural order. Let $w(p)$ and $l(p)$ denote the weight and the label of p, respectively.

DEFINITION 1 (formal-language-constrained shortest path). *Given an (un)directed labeled, weighted graph G, a source s, a destination d, and a formal language (regular, context-free, context-sensitive, etc.) L, find a shortest (not necessarily simple) path p in G such that $l(p) \in L$.*

DEFINITION 2 (formal-language-constrained simple path). *Given an (un)directed labeled, weighted graph G, a source s, a destination d, and a formal language (regular, context-free, context-sensitive, etc.) L, find a shortest simple path p in G such that $l(p) \in L$.*

For the rest of the paper we denote the formal-language-constrained shortest path problem restricted to regular, context-free, and context-sensitive languages by REG-ShP, CFG-ShP, and CSG-ShP, respectively. Similarly, we denote the formal-language-constrained simple path problem restricted to regular, context-free, and context-sensitive languages by REG-SiP, CFG-SiP, and CSG-SiP, respectively.

In general we consider the input for these problems to consist of a description of the graph (including labeling and weights) together with the description of the formal language as a grammar. By restricting the topology of the graph and/or the syntactic structure of the grammar, we get modifications of the problems. If we claim a statement to be true "for a *fixed* language," we refer to the variant of the problem, where the input consists of the graph only, whereas the language is considered to be part of the problem specification.

Note that in unlabeled networks with nonnegative edge weights, a shortest path between s and d is necessarily simple. This need not be true when we wish to find a shortest path subject to an additional constraints on the set of allowable labels. As a simple example, consider the graph $G(V, E)$ that is a simple cycle on four nodes. Let all the edges have weight 1 and label a. Now consider two adjacent vertices x and y. The shortest path from x to y consists of a single edge between them; in contrast a shortest path with label $aaaaa$ consists of a cycle starting at x and the additional edge (x, y).

3. Summary of results. We investigate the problem of formal-language-constrained path problems. A number of variants of the problem are considered and both polynomial-time algorithms as well as hardness results (NP-, PSPACE-hardness, undecidability) are proved. Two of the NP-hardness results are obtained by combining the simplicity of a path with the constraints imposed by a formal language. We

	word recognition	shortest path general graph	simple path treewidth bound	simple path grid graph
Fixed finite	FP	FP	FP	FP
Free finite	FP	FP	FP	NP-c.
Fixed LT	FP	FP	FP	NP-c.[5]
Free RL	FP	FP [1]	FP	NP-c.
Fix. 1-log-SPCE-TM	FP	undec[2]	FP	NP-c.
Fix. lin. det. CFL	FP	FP	NP-c. [4]	NP-c.
Free CFL	FP	F P[3]	NP-c.	NP-c.
Fixed CSL	PSPACE-c.	undec.	PSPACE-c.	PSPACE-c.

[1]Section 5.1, Theorem 11; [2]section 7.5; [3]section 5.3; [4]section 6.3, Theorem 26; [5]section 6.2, Theorem 20.

FIG. 1. *Summary of results on formal-language-constrained simple/shortest paths in contrast to the word recognition problems. LT, RL, CFL, CSL denote locally testable, regular, context-free, and context-sensitive languages, respectively. For regular languages, the time bounds hold for regular expressions with the operators* $(\cup, \cdot, *, 2)$. *FP states that the problem can be computed in deterministic polynomial time, even if the language specification is part of the input. The superscripts in the table and the corresponding text tell where the result is proven.*

believe that the techniques used to prove these results are of independent interest. The main results obtained in the paper are summarized in Figure 1 and include the following:

(1) We show that CFG-ShP has a polynomial-time algorithm. For REG-ShP with operators $(\cup, \cdot, *)$, we give polynomial-time algorithms that are substantially more efficient in terms of time and space. The polynomial-time solvability holds for the REG-ShP problem, when the underlying regular expressions are composed of $(\cup, \cdot, *, 2)^2$ operators. We also observe that the extension to regular expressions over operators $(\cup, \cdot, *, -)$ is PSPACE-hard.

(2) In contrast to the results for shortest paths, we show that the problem finding any *simple* paths between a source and a given destination is NP-hard, even when restricted to a very simple, fixed locally testable regular language (see section 6.2 for details) and very simple graphs (undirected grid graphs).

(3) In contrast to the results in (1) and (2) above, we show that for the class of treewidth-bounded graphs, (i) the REG-SiP is solvable in polynomial time, but (ii) CFG-SiP problem is NP-complete, even for a fixed deterministic linear context-free language (CFL). The easiness proof can be extended to one-way-log-space recognizable languages. It uses a dynamic programming method, although the tables turn out to be quite intricate.

(4) Finally, we investigate complexity of problems CSG-ShP and CSG-SiP. Using simple reductions and, in contrast to the complexity results in (1) and (2), we show that (i) CSG-SiP is PSPACE-complete but (ii) CSG-ShP is *undecidable* even for a fixed language.

(5) As an application of the theory developed here, we provide a polynomial-time algorithm for a number of basic problems in transportation science. In

[2]Operator $\square^2$ or simply 2 stands for the square operator. R^2 denotes $R \cdot R$.

section 7 we consider two examples, namely, the BEST k-SIMILAR PATH and the TRIP CHAINING problem.

The results mentioned in (1)–(4) provide a tight bound on the computational complexity (P versus NP) of the problems considered, given the following assumptions: The inclusions of classes of formal languages "finite $\subset$ locally testable" and "regular $\subset$ deterministic linear context-free" are tight, i.e., there is no natural class of languages "in between" these classes. Furthermore in this paper grid-graphs are considered to be the "easiest" class of graphs that do not have a bounded treewidth.

Preliminary versions of the algorithms outlined here have already been incorporated in the route planning module of TRANSIMS. In [JMN98] we conduct an extensive empirical analysis of these and other basic route-finding algorithms on realistic traffic network.

4. Related work. We refer the reader to the monograph by Huckenbeck [Hu97] for a comprehensive survey on path problems. References [TR+95a, AB+97, TR+95b] provide a detailed account of the TRANSIMS project. Regular-expression-constrained simple path problems were considered by Mendelzon and Wood [MW95]. The authors investigate this problem in the context of finding efficient algorithms for processing database queries (see [CMW87, CMW88, MW95]). A recent paper by Abiteboul and Vianu describes further results on related problems [AV99]. Yannakakis [Ya90, Ya95] in his keynote talk has independently outlined some of the polynomial-time algorithms given in section 5. Romeuf [Ro88] also independently considered some of the problems discussed in section 5. However, the emphasis in [Ya90] was on database theory and Romeuf [Ro88] considered only regular languages. Online algorithms for finding regular-expression-constrained paths are given in [BKV91]. Our work on finding formal-language-constrained shortest paths is also related to the work of Ramalingam and Reps [RR96]. The authors were interested in finding a minimum-cost derivation of a terminal string from one or more nonterminals of a given context-free grammar. The problem was first considered by Knuth [Ku77] and is referred to as the *grammar problem*. [RR96] gives an incremental algorithm for a version of the grammar problem and as corollaries obtain incremental algorithms for single-source shortest path problems with positive edge weights. We close this section with the following additional remarks:

(1) To our knowledge this is the first attempt at using formal language theory in the context of modeling mode/route choices in transportation science.

(2) The polynomial-time algorithms and the hardness results presented here give a boundary on the classes of graphs and queries for which polynomial-time query evaluation is possible. In [MW95] the authors state that

> Additional classes of queries/dbgraphs for which polynomial-time evaluation is possible should be identified ....

Our results significantly extend the known hardness as well as easiness results in [MW95] on finding regular-expression-constrained simple paths. For example, the only graph theoretic restriction considered in [MW95] was acyclicity. On the positive side, our polynomial-time algorithms for regular-expression-constrained simple path problems when restricted to graphs of bounded treewidth are a step toward characterizing graph classes on which the problem is easy. Specifically, it shows that for graphs with fixed-size recursive separators, the problems is easy. Examples of graphs that can be cast in this framework include chordal graphs with fixed clique size, outer planar graphs, series parallel graphs, etc. (see [Bo92] for other examples).

(3) The basic techniques extend quite easily (with appropriate time performance

bounds) to solve other (regular-expression-constrained) variants of shortest path problems. Two notable examples that frequently arise in transportation science and can be solved are (i) multiple-cost shortest paths [Ha92, BAL97] and (ii) time-dependent shortest paths [OR90]. These extensions are briefly outlined in section 7.

The rest of the paper is organized as follows. Section 4.1 contains preliminary results and basic definitions. In section 5, we present efficient algorithms for grammar-constrained shortest path problems (namely, REG-SHP and CFG-SHP). Section 6 contains our hardness/easiness results for simple paths. Section 7 outlines several extensions and applications of our basic results.

4.1. Basic definitions. We recall the basic concepts in formal language and graph theory. Additional basic definitions on topics related to this paper can be found in [HU79, GJ79, AHU, CLR, HRS76, Pa94, Ta81]. For the rest of the paper, we use $|I|$ to denote the size of an object I represented using binary notation.

DEFINITION 3. *Let Σ be a finite alphabet disjoint from $\{\varepsilon, \phi, (,), \cup, \cdot, *\}$. A regular expression R over Σ is defined as follows:*

(1) *The* empty string *"ε," the* empty set *"ϕ," and, for each $a \in \Sigma$, "a" are atomic regular expressions.*
(2) *If R_1 and R_2 are regular expressions, then $(R_1 \cup R_2)$, $(R_1 \cdot R_2)$, and $(R_1)^*$ are compound regular expressions.*

DEFINITION 4. *Given a regular expression R, the language (or the set) defined by R over Σ and denoted by $L(R)$ is defined as follows:*

(1) $L(\varepsilon) = \{\varepsilon\}$; $L(\phi) = \phi$; $\forall a \in \Sigma: L(a) = \{a\}$.
(2) $L(R_1 \cup R_2) = L(R_1) \cup L(R_2) = \{\, w \mid w \in L(R_1) \text{ or } w \in L(R_2) \,\}$.
(3) $L(R_1 \cdot R_2) = L(R_1) \cdot L(R_2) = \{\, w_1 w_2 \mid w_1 \in L(R_1) \text{ and } w_2 \in L(R_2) \,\}$.
(4) $L(R^*) = \bigcup_{k=0}^{\infty} L(R)^k$, *where* $L(R)^0 = \{\varepsilon\}$ *and* $L(R)^i = L(R)^{i-1} \cdot L(R)$.

DEFINITION 5. *A* nondeterministic finite automaton *(NFA) is a 5-tuple $M = (S, \Sigma, \delta, s_0, F)$, where*

(1) *S is a finite nonempty set of states;*
(2) *Σ is the input alphabet (a finite nonempty set of letters);*
(3) *δ is the* state transition function *from $S \times (\Sigma \cup \{\varepsilon\})$ to the power set of S;*
(4) *$s_0 \in S$ is the initial state;*
(5) *$F \subseteq S$ is the set of accepting states.*

If there is an $s \in S$ such that $\delta(s, \varepsilon)$ is a nonempty subset of S, then the automaton M is said to have ε-transitions. If M does not have any ε-transitions and $\forall\ s \in S$ and $\forall\ a \in \Sigma$ the set $\delta(s, a)$ has at most one element, then δ can be regarded as a (partial) function from $S \times \Sigma$ to S and M is said to be a *deterministic finite automaton* (DFA).

The extended transition function δ^* from $S \times \Sigma^*$ is defined in a standard manner. The size of M denoted by $|M|$ is defined to be equal to $|S||\Sigma|$.

DEFINITION 6. *Let $M = (S, \Sigma, \delta, s_0, F)$ be an NFA. The* language accepted *by M denoted $L(M)$ is the set*

$$L(M) = \{\, w \in \Sigma^* \mid \delta^*(s_0, w) \cap F \neq \phi \,\}.$$

A string w is said to be accepted by the automaton M if and only if $w \in L(M)$.

DEFINITION 7. *A* context-free grammar *(CFG) G is a quadruple (V, Σ, P, S), where V and Σ are disjoint nonempty sets of* nonterminals *and* terminals*, respectively, $P \subset V \times (V \cup \Sigma)^*$ is a finite set of* productions*, and S is the* start symbol*. A CFG G is said to be* linear *if at most one nonterminal appears on the right-hand side of any of its productions.*

DEFINITION 8 (see[Bo88, AL+91]). *Let $G = (V, E)$ be a graph. A* tree-decomposition *of G is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a family of subsets of V and $T = (I, F)$ is a tree with the following properties:*

(1) $\bigcup_{i \in I} X_i = V$.
(2) *For every edge $e = (v, w) \in E$, there is a subset X_i, $i \in I$, with $v \in X_i$ and $w \in X_i$.*
(3) *For all $i, j, k \in I$, if j lies on the path from i to k in T, then $X_i \bigcap X_k \subseteq X_j$.*

The treewidth *of a tree-decomposition $(\{X_i \mid i \in I\}, T)$ is $\max_{i \in I} |X_i| - 1$. The treewidth of G is the minimum treewidth of a tree decomposition.*

5. Shortest paths. In this section, we present polynomial-time algorithms for the problems REG-SHP and CFG-SHP.

5.1. Algorithm for REG-SHP and extensions. In this subsection, we will describe our algorithms for regular expression constrained shortest path problems. We note that regular expressions over $(\cup, \cdot, *)$ can be transformed into equivalent NFAs in $O(n)$ time [AHU], where n represents the size of the regular expression. Thus for the rest of this subsection we assume that the regular expressions are specified in terms of an equivalent NFA.

The basic idea behind finding shortest paths satisfying regular expressions is to construct an auxiliary graph (the product graph) combining the NFA denoting the regular expression and the underlying graph. We formalize this notation in the following.

DEFINITION 9. *Given a labeled directed graph G, a source s, and a destination d, define the NFA $M(G) = (S, \Sigma, \delta, s_0, F)$ as follows:*

(1) *$S = V$; $s_0 = s$; $F = \{d\}$;*
(2) *Σ is the set of all labels that are used to label the edges in G; and*
(3) *$j \in \delta(i, a)$ if and only if there is an edge (i, j) with label a.*

Note that this definition can be used to interpret an NFA as a labeled graph as well.

DEFINITION 10. *Let $M_1 = (S_1, \Sigma, \delta_1, p_0, F_1)$ and $M_2 = (S_2, \Sigma, \delta_2, q_0, F_2)$ be two NFAs. The product NFA is defined as $M_1 \times M_2 = (S_1 \times S_2, \Sigma, \delta, (p_0, q_0), F_1 \times F_2)$, where $\forall a \in \Sigma$, $(p_2, q_2) \in \delta((p_1, q_1), a)$ if and only if $p_2 \in \delta_1(p_1, a)$ and $q_2 \in \delta_2(q_1, a)$.*

It is clear that $L(M_1 \times M_2) = L(M_1) \cap L(M_2)$. Algorithm RE-Constrained-Short-Paths outlines the basic steps for solving the problem and uses the cross product construction mentioned above.

THEOREM 11. *The algorithm RE-Constrained-Short-Paths computes the exact solution for the problem* REG-SHP *with nonnegative edge weights in time $O(T(|R||G|))$. Here $T(n)$ denotes the running time of a shortest path algorithm on a graph with n nodes.*

Proof. For the correctness of the algorithm observe the following: Consider a shortest path q^* in G of cost $w(q^*)$ that satisfies the regular expression R. This path and the accepting sequence of states in the NFA imply a path q' of the same cost between (s_0, s) to (f, d) for some $f \in F$. So we know that the sought path is considered. Conversely, for each path τ in $M(G) \times M(R)$ of cost $w(\tau)$ that begins on a starting state and ends on a final state, the projection of τ to the nodes of G yields a path of same cost in G from s to d that satisfies the regular expression R.

To calculate the running time of the algorithm, observe that the size of $M(R) \times M(G)$ is $O(|R||G|)$. As the overall running time is dominated by step 4, we obtain $O(T(|R||G|))$ as a bound. ∎

ALGORITHM RE-CONSTRAINED-SHORT-PATHS:

- *Input:* A regular expression R, a directed labeled weighted graph G, a source s, and a destination d.
- 1. Construct an NFA $M(R) = (S, \Sigma, \delta, s_0, F)$ from R.
  2. Construct the NFA $M(G)$ of G.
  3. Construct $M(G) \times M(R)$. The length of the edges in the product graph is chosen to be equal to the corresponding edges in G.
  4. Starting from state (s_0, s), find a shortest path to the vertices (f, d), where $f \in F$. Denote these paths by p_i, $1 \le i \le w$. Also denote the cost of p_i by $w(p_i)$.
  5. $C^* := \min_{p_i} w(p_i)$; p^*: $w(p^*) = C^*$.
     (If p^* is not uniquely determined, we choose an arbitrary one.)
- *Output:* The path p^* in G from s to d of minimum length subject to the constraint that $l(p) \in L(R)$.

5.2. Extensions: Other regular expressions. We consider two possible extensions for the problem REG-SHP, namely, allowing the additional operators for taking the complement $(-)$ and for squaring an expression (2).

THEOREM 12. *Shortest path in the complement of an NFA or a regular expression over* $(\cup, \cdot, *)$ *is PSPACE-hard.*

Proof. Let R be such a regular expression. The question of deciding if the complement of $L(R)$ is empty (regular expression nonuniversality) is known to be PSPACE-complete (see [GJ79, problems AL1 and AL9]). Given an instance of such a problem we create a graph G with one node v. There is a loop from v to v for each symbol in the alphabet. The existence of a path from v to v with the label in the complement of the language $(\Sigma^* - L(R))$ is equivalent to the existence of such a word. □

This immediately implies the following corollary.

COROLLARY 13. REG-SHP *for regular expressions over* $(\cup, \cdot, *, -)$ *is PSPACE-hard.*

It is easy to see that regular expressions consisting of operators from $(\cup, \cdot, *, 2)$ can be represented by CFGs with rules of the form $A \to BB$. This observation together with the results of the next section (section 5.3) yields the following corollary.

COROLLARY 14. REG-SHP *for regular expressions over* $(\cup, \cdot, *, 2)$ *can be solved in polynomial time.*

5.3. Algorithm for CFG-SHP. We now extend our results in section 5.1 to obtain polynomial-time algorithms for CFL-constrained shortest path problems. Beside the applicability of the algorithm, this result stands in contrast to the hardness of simple-path problems even for a single fixed regular expression.

The algorithm for solving CFG-constrained shortest paths is based on dynamic programming. Hence we will first investigate the structure of an optimal shortest path from s to d in the graph G that is labeled according to the CFG R. Assume that R is in Chomsky normal form, i.e., all rules of the form $C \to AB$ or $C \to a$ (see [HU79] for details). Consider any such shortest path p with $l(p) = a_1 a_2 \cdots a_m$. One important property of any CFG is that nonterminals are expanded independently. In the case of a Chomsky normal form, the derivation forms a binary tree, which means that the

label of p can be decomposed into two parts l_1 and l_2 such that $l(p) = l_1 l_2$, $S \to AB$, $A \xrightarrow{*} l_1$, and $B \xrightarrow{*} l_2$.

With this structure in mind let us define the quantity $D(i, j, A)$ as the shortest path distance from i to j subject to the constraint that the label on this path can be derived starting from the nonterminal A.

These values are well defined and fulfill the following recurrence:

$$D(i,j,A) = \min_{(A \to BC) \in R} \min_{k \in V} \Big(D(i,k,B) + D(k,j,C) \Big), \tag{1}$$

$$D(i,j,a) = \begin{cases} w(i,j) & \text{if } l((i,j)) = a, \\ \infty & \text{otherwise.} \end{cases} \tag{2}$$

Observation 15. *These equations uniquely determine the function D and immediately imply a polynomial-time dynamic programming algorithm.*

Proof. Consider the case that D and D' satisfy the above recurrence but are different. Then there must be a smallest witness of this fact $a = D(i, j, X)$ and $b = D'(i, j, X)$, and $a \neq b$. Let us assume $a > b$ and consider the smallest such b. As the definition (2) is unambiguous, we know that $X = A$ is a nonterminal. As b is minimal, it is finite and satisfies (1). This implies that there must exist the witnesses $e = D(i, k, B)$ and $f = D(k, j, C)$ that establish $b = e + f$ as their sum. As all lengths in the graph are positive, both e and f are smaller than b. By the choice of a and b we know that $e = D'(i, k, B)$ and $f = D'(k, j, C)$, implying with (1) the contradiction $a \leq e + f = b$.

This discussion immediately implies a dynamic programming approach to compute the table of D. Starting with the values for links in the network, we fill the table with increasing values. This can be done with a Bellman–Ford-type algorithm. This algorithm will finally set at least one entry in the table per round. The execution time is bound by the square of the number of entries in the table times the amount of time needed to compute the two minima in (1). This is polynomial, namely, $O(|V|^5|N|^2|R|^2)$ with V being the vertices of the graph, N the nonterminals, and R the rules of the grammar in Chomsky normal form.

Another way of implementing this is to set the table by filling in the smallest values first. There a heap is used to hold the current estimates on entries, and the smallest one is finally put in the table generating or changing estimates. This implies one extract-min operation and up to $2|V||R|$ update operations for every entry. Using Fibonacci–Heaps (see [CLR] for an analysis), this sums up to $O(|V|^2|N| \cdot (\log(|V|^2|N|) + 2|V||R|))$, that is, $O(|V|^3|N||R|)$. ∎

A naive adaptation of a Floyd–Warshall-type algorithm fails, because we cannot split an optimal path in two optimal subsolutions at an arbitrary node of the path. The splitting in the above dynamic programming works only because it is done in accordance with the grammar.

6. Simple paths. Next we investigate the complexity of finding formal-language-constrained simple paths. For the ease of exposition, we present our results for directed, multilabeled graphs. The following lemma shows how to extend these results to undirected and unilabeled graphs.

Lemma 16.

(1) REG-SiP *on directed, multilabeled grids can be reduced to* REG-SiP *on directed grids.*

(2) REG-SiP *on directed grids can be reduced to* REG-SiP *on undirected grids.*

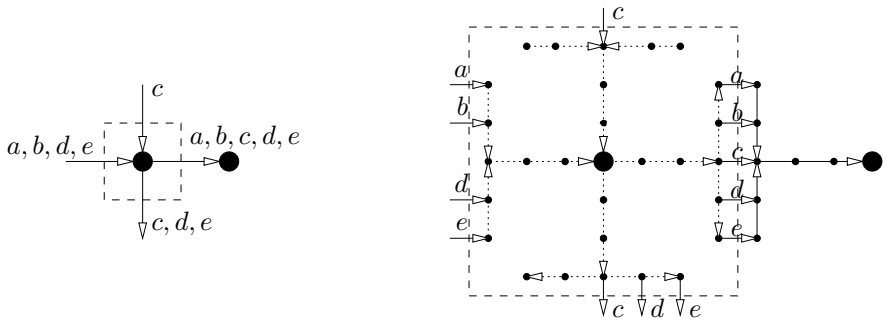


FIG. 2. *Example for replacing multilabels. All the dotted edges "inside" are labeled with* v. *Only the connection to the right is completely depicted.*

(3) *For all* $k \geq 1$, CFG-SIP *on directed graphs of treewidth* k *can be reduced to* CFG-SIP *on undirected graphs of the same treewidth.*

Proof. (1) Applying the changes illustrated in Figure 2 to all nodes of a multilabeled grid G, we obtain an unilabeled grid G'. G' is part of a grid roughly $|\Sigma|^2$ times larger than the original one. Furthermore, the alphabet needs to be extended by two symbols, one for inaccessible edges and one, the symbol v, for the edges "inside" the extended nodes.

The regular language L is replaced by the regular language L' defined as follows:

$$L' = \{\ w \in (\Sigma \cup \{v\})^* \mid w = v^* x_1 v^* x_2 v^* \cdots v^* x_n v^* \ \text{ and } \ x_1 x_2 \cdots x_n \in L\ \}.$$

Now paths in the new instance (G', L') are in one-to-one correspondence to paths in the original instance (G, L).

(2), (3) It is well known (see, for example, [HU79]) that regular and CFLs are closed under substitution. Let $\Sigma = \{a, b, \ldots, z\}$ be an alphabet. Then $\Sigma' = \{a', b', \ldots, z'\}$ is a marked copy of this alphabet. Substitution with the homomorphism $a \mapsto aa'$, $\forall\ a \in \Sigma$ yields the following: If $L \subseteq \Sigma^*$ is regular, the language

$$L' = \{\ w \in (\Sigma \cup \Sigma')^* | w = x_1 x_1' x_2 x_2' \cdots x_n x_n' \ \text{ and } \ x_1 x_2 \cdots x_n \in L\ \}$$

is also regular. Let G be the original graph. The directed edges in G labeled with a get replaced by two consecutive edges labeled with a and a', introducing a new node.

In this situation paths in G' complying with L' are in a one-to-one correspondence to paths in G complying with L. It is straightforward to extend the weight function on the edges to preserve the weights of paths. Additionally relative path length (in number of edges used) are preserved.

The proof of (2) follows from the fact that the resulting G' can be embedded in a grid using a new symbol $v \notin (\Sigma \cup \Sigma')$ as label.

For (3) let T be a tree-decomposition of treewidth k. If $k = 1$, the graph itself is a tree and replacing edges by paths of length 2 does not change the treewidth. Otherwise let T be a tree-decomposition of width $k > 1$. For every new node we create a new set of the tree-decomposition consisting of the new node and the endpoints of the edge it splits. This set is included in the tree of the decomposition by attaching it to the set that covered the split edge. As $k > 1$, sets of cardinality 3 cannot change the treewidth, yielding a new tree-decomposition of width k. ∎

6.1. Finite languages.

DEFINITION 17. *A language L is called* finite *if there are only finitely many words in L.*

Finite languages are considered one of the smallest subclasses of the regular languages.

THEOREM 18. *For any* fixed *finite language L the problem* REG-SIP *can be solved in polynomial time.*

Proof. Let k be the maximum length of a word in L. Considering all k-tuples of nodes, and checking if they form the sought path, yields a polynomial-time algorithm of running time $O(n^k)$. □

THEOREM 19. *Let $\mathcal{C}$ be a graph class (such as planar, grid, etc.) such that the* HAMILTONIAN PATH *problem is* NP*-hard when restricted to $\mathcal{C}$. Then the problem* REG-SIP *is* NP*-hard when restricted to $\mathcal{C}$ and (free) finite languages.*

Proof. Consider a fixed class of graphs $\mathcal{C}$ for which the HAMILTONIAN PATH problem is NP-hard. Then given an instance G of the HAMILTONIAN PATH problem in which $G \in \mathcal{C}$, with n nodes, we construct an instance G_1 of the regular-expression-constrained simple path problem by labeling all the edges in G by a. We now claim that there is a Hamiltonian path in G if and only if there is a simple path in G_1 that satisfies a^{n-1}, i.e., the constraining language is chosen to be the finite language $L = \{a^{n-1}\}$. □

6.2. Hardness of REG-SIP. Before formally stating the theorem and the proof we present the overall idea. We perform a reduction from 3-SAT. It will be easy to verify that the reduction can be carried out in polynomial time (logarithmic space).

Our encoding of a satisfiability question naturally decomposes into two parts. The first is to choose an assignment; the second is to check whether this assignment satisfies the given formula. Choosing the assignment will correspond to choosing a certain part of the path. The 3-CNF formula will be checked clause by clause. This requires access to the value of a variable more than once. Here this is achieved by forcing sufficiently many subpaths to be similar (in the sense that they stand for the same assignment). These "copies" of the assignment can then be used to check whether the underlying assignment satisfies the 3-CNF formula. This is now a local task, as there is one copy of the assignment for every clause.

In order to achieve sufficiently many similar copies of the assignment, the following *beads and holes argument* will be useful: Think of $n + 1$ holes forming a straight line and n beads between every two of them. Each bead is allowed to fall in one of the two holes adjacent to it. Moreover, we allow at most one bead in each hole. The state of the system after the beads fall down in the holes is a description of the contents of each hole. By reporting which hole is free, the state of the beads and holes system is described completely; since all beads left of the free hole fell in the left hole and the remaining beads fell into the right hole. We additionally know that there exists a set of at least $n/2$ consecutive beads that fell in the same direction.

The construction presented enforces an overall snake-like path, which goes up and down several times. This is schematically depicted in Figure 3. Figure 4 provides additional details about the construction and will be described in what follows. At this point it is sufficient to note that nodes on a vertical dotted line will be referred to as nodes on a column, and vertices on each horizontal dotted line will be referred to as nodes on a level (or a row). For the case of n levels (variables) and m columns the following statements provide an overall outline of the proof:

(1) Every column of the rotated grid (depicted in Figure 4) represents a member

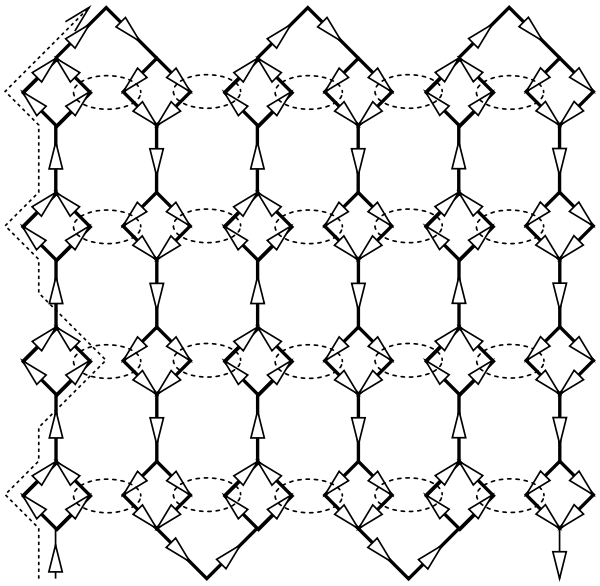


FIG. 3. *Overall shape of all feasible paths; the dotted circles stand for nodes to be identified. One possible leg in the first column is depicted as a dashed line.*

of an extended sequence of clauses. Any feasible path from S to D will use all of the column nodes. Such a path naturally decomposes into subpaths which span an entire column and are called legs.

(2) The shape of a leg uniquely corresponds to an assignment; for this, levels of the rotated grid are identified with variables.

(3) Identically shaped legs do not interfere with neighboring columns.

(4) Let p be one fixed feasible path, i.e., a path that consists of m legs, as many as there are columns. Such a path visits every node but precisely n variable nodes, one per level. Therefore, the shape of the legs of p cannot change too often (following the beads and holes argument), which in turn ensures that there will be a lot of similar legs.

(5) On each column the labeling together with the language allows us to check that the assignment represented by the leg satisfies a clause (of the extended set of clauses). For this we use three symbol of the alphabet to stand for the three literals of the clause.

We will prove the theorem for multilabeled directed grid-graphs. Lemma 16 implies that the result holds for unilabeled undirected grid graphs.

THEOREM 20. *The* REG-SIP *problem is* NP*-hard for complete multilabeled directed grids and a* fixed *regular expression.*

Proof. We present a reduction from the problem 3-SAT, which is well known to be NP-complete; see, for example, [GJ79].

Given a 3-SAT formula we construct a labeled complete directed grid, such that there exists a path from the start vertex to the destination vertex, that complies with a fixed (independent of the formula) regular expression, if and only if the formula is satisfiable.

Let $F = (X, C)$ be a 3-CNF formula, $X = \{x_1, x_2, x_3, \ldots, x_n\}$ the set of variables, and $C = \{c_1, c_2, c_3, \ldots, c_m\}$ the set of clauses.

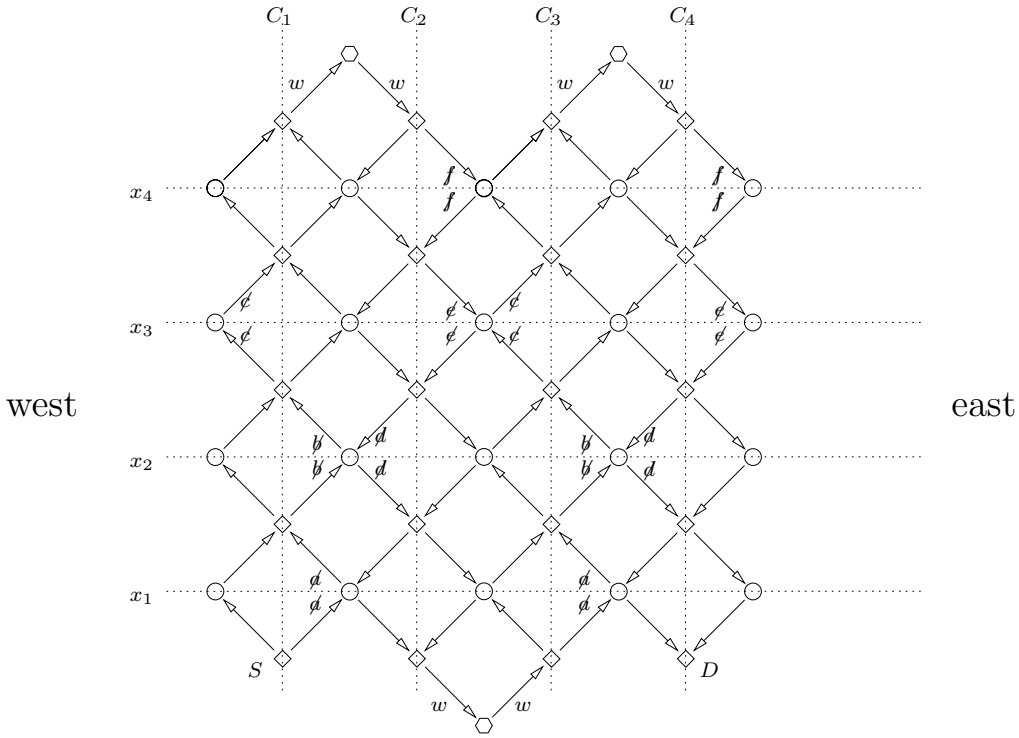


FIG. 4. *Graph corresponding to the formula* $(x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_2 \vee x_3 \vee x_4)$*, west-true, east-false, in general upward edges are labeled* a*,* b*, and* c*, downward edges are labeled* d*,* e*, and* f*. The label* $\not{c}$ *in the figure states that the corresponding edge is not labeled with* a *(by this deviating from the general rule). For simplicity the additional* $4 = (4+2)-2$ *repetitions of the basic clause-sequence are omitted.*

For the beads and holes argument we need $n + 2$ repetitions of the sequence of clauses. To do this, we construct an extended sequence $d_1, \ldots, d_M$ of clauses that consists of exactly $n+2$ copies of the original sequence of clauses ($M = m(n+2)$). It is important that the ordering of the basic sequence remains unchanged between different copies. This extended sequence of clauses is used in what follows for constructing the graph.

We will describe a grid rotated by 45 degrees against the coordinate system that we use to define levels (i.e., rows) and columns. There are three different types of vertices: (i) clause-vertices, (ii) variable-vertices, and (iii) join-vertices. The layout of these vertices is illustrated in Figure 4 by means of an example. The coordinate system essentially has for every clause C_i a vertical line with the name of the clause and for every variable x_i a horizontal line named by the variable. These lines are shown as dotted lines in Figure 4. The vertices of the grid lie in the middle of the so-formed line-segments. Clause-vertices lie on dotted vertical (clause-)lines and are drawn as rhombuses. Variable vertices lie on horizontal (variable-)lines and are drawn as circles. There are additional bottom and top lines of join-vertices that are drawn as hexagons. It is easy to see that this grid is part of a complete square grid with $(M + n + 2)^2$ nodes.

To simplify the description we assume that M is even. For the rest of the proof, by a slight abuse of notation, we use the phrase edges incident on a vertex v to mean both

the incoming and outgoing edges that have v as one of the endpoints. The start node S is the lowermost clause-node on C_1, the destination node D the lowermost clause-node on C_M. To achieve paths of the snake-like form depicted in Figure 3, we direct all edges incident on a clause-vertex on an odd-numbered column upward and all edges incident on a clause-vertex lying on even-numbered columns downward. This orientation is depicted in Figure 4. In order to enforce the overall shape of admissible paths, we label the edges incident on join-vertices with w. Furthermore, with important exceptions to be described later, upward edges are in general multilabeled with a, b, and c, where as downward edges are in general multilabeled with d, e, and f. Let us define the regular expression

$$R = \Big((a^* \cup b^* \cup c^*)ww(d^* \cup e^* \cup f^*)ww\Big)^*(a^* \cup b^* \cup c^*)ww(d^* \cup e^* \cup f^*)$$

and the corresponding language $L = L(R)$.

The final construction will have some of the labels removed; this only makes the set of feasible paths smaller. The following proposition summarizes a key property of the above construction.

PROPOSITION 21. *Let P be a simple path in the described graph from S to D complying with the regular language L. Then the following holds:* (i) *P can be partitioned into a set of legs and* (ii) *there exist m consecutive legs that have identical shape.*

Proof. (i) Call every subpath labeled with $w(d^+ \cup e^+ \cup f^+)w$ or $w(a^+ \cup b^+ \cup c^+)w$ a leg. Here $x^+ := x^*x$ is the usual shorthand used in regular expressions. As the label of P is in the given regular language, this defines a partitioning of P into M subpaths L_i. Because of the labeling of the grid, all legs have one endpoint at the lower join level and one at the upper join level. Because of this all L_i have the same length. Additionally every second vertex of a leg is a clause-node, and all these clause-nodes are in the same column. As we traverse a leg from the low-numbered end to the high-numbered end, we deviate from the vertical line passing through variable-vertices on the way. The sequence of deviations (east/west) defines the *shape* of the leg. The shape of a leg is used to infer an assignment to the variables as discussed later.

(ii) As P is simple, every node of the grid is visited at most once by P. As a result the shapes of two neighboring legs L_i and L_{i+1} are not independent. If L_i deviates at (variable-)level j to the east, L_{i+1} may not deviate to the west on that level. As there are $(M+1)n$ variable-vertices in the grid and Mn variable-vertices on P, there are exactly n variable-vertices not used by P. An averaging argument yields that there must exist m consecutive columns of variable-vertices that are all used by P. This implies that all legs in that range must have identical shape. □

These m legs will be used as multiple copies of an encoding of an assignment of truth values to the variables. The removing of some of the labels of the graph will allow us to enforce the semantics of the clauses. Given the shape of a leg we use the following rule (denoted rule **R**) to infer an assignment to the variables:

(R) If the shape deviates on level x_i to the west (east), we assign x_i the truth value TRUE (FALSE).

Note that we have m legs with identical shapes and thus we have a consistent assignment to the variables across clauses. The constraining regular expression and appropriate labels to the edges also have to ensure that the following holds. The

leg corresponding to each C_i can only have shapes such that the corresponding assignment to the variables using rule **R** makes C_i true. For an odd-numbered clause $C_{2k+1} \equiv (u_1 \vee u_2 \vee u_3)$, we remove labels from the graph in the following way: If $u_1 = x_i$ (resp., $u_1 = \neg x_i$) we remove the label a from both the edges incident on the variable-vertex x_i that is to the east (resp., west) of the column C_{2k+1}. For u_2 (resp., u_3) the label b (resp., c) is removed in the same way on the level corresponding to the variable of the literal. For even-numbered clauses d, e, and f take the role of a, b, and c.

PROPOSITION 22. *Let L_k be a leg of a path in G from S to D on column C_k complying with L and let A be the assignment corresponding to the shape of L_k using rule* **R**. *Then the labeling of L_k can be chosen to be of the form $wxx \dots xxw$ with $x \in \{a, b, c, d, e, f\}$ if and only if A evaluates C_k to TRUE.*

Proof. Let A evaluate C_k to TRUE. Then at least one of the literals of C_k is TRUE. We can choose the x accordingly (for example, $x = a$ if it is the first literal and k is odd). On the level corresponding to this positive (resp., negative) literal the path deviates to the west (resp., east), and can thus be labeled with x; independent of its shape, L_k can be labeled with x on all other levels.

Conversely, if the leg is labeled in $wxx \dots xxw$, we can focus on the literal of the clause corresponding to x (for example, the first literal if $x = a$). As the labeling at the level of that variable is available only in the correct direction, A must evaluate the clause to true. □

This completes the construction of the graph. We now prove the correctness of the reduction.

Suppose there exists a satisfying assignment A to the formula. Then we choose the shape of the path from S to D in the snake-like fashion with the additional property that we deviate to the west on level x_j if x_j is set to TRUE by A; otherwise we deviate to the east. Since A is a satisfying assignment, each clause contains at least one literal that is set true by A. So we choose the labeling on all legs L_k of the path in column C_k corresponding to this literal of C_k. This yields a simple path that complies with L.

Conversely, let there be a simple path from S to D complying with L. By Proposition 21, we know that the path has m consecutive legs of the same shape corresponding to an assignment A. By construction of the extended sequence of clauses and Proposition 22 we know that A satisfies all of the original clauses. So the formula is satisfiable. □

In fact this result stays valid for a smaller class of infinite regular languages.

DEFINITION 23 (see [Str94]). *A language L is called locally testable if there exists a k and a finite set S such that*

$$w \in L \iff (\forall v <_k w : v \in S).$$

Here $v <_k w$ stands for the fact that v is a subword of length k of w.

COROLLARY 24. *The* REG-SIP *problem is* NP*-hard for a complete multilabeled directed grid and a fixed locally testable language.*

Proof. In the proof of Theorem 20 the constraining regular language can be replaced by the regular language L defined as follows:

$$L = \left\{ v \in \Sigma^* \;\middle|\; x_1 x_2 <_2 v \to \Big(x_1 \in \{a, b, c, d, e, f\} \to x_2 = x_1 \vee x_2 = w\Big) \right\}.$$

This condition is equivalent to stating that any sequence of $y \in \{a, b, c, d, e, f\}$ may be ended only by a w. Starting from the first symbol in the word, L guarantees

that this single symbol is repeated until the next w. The labeling of the grid ensures that this w is at the top join-vertex between the columns C_1 and C_2. There the simplicity enforces another w and it follows inductively that L enforces the snake-like shape of the path as well as the uniform labeling of the legs. So it is as strong as the language used in the proof of Theorem 20.

It should be noted that Lemma 16 cannot be applied immediately in this situation. Nevertheless the result of the corollary extends. This is easy to verify using a slight modification of the graph proposed in the proof of Lemma 16 and a modification of the language. This makes use of the fact that the directionality can readily be omitted and that constructions of the form $x_i = y \rightarrow x_{i+5} = y$ are also expressible in locally testable languages. □

Note that the result of Theorem 20 immediately extends to other graph classes.

COROLLARY 25. *Let $\mathcal{C}$ be a class of graphs such that $\forall\ k > 0$ there is an instance $\mathcal{I} \in \mathcal{C}$ that contains as a subgraph a $(k \times k)$-mesh and both the graph and the subgraph are computable in time polynomial in k; then the* REG-SIP *problem is* NP*-hard for $\mathcal{C}$.*

Thus the REG-SIP *problem is* NP*-hard even for* (i) *complete cliques,* (ii) *interval graphs,* (iii) *chordal graphs,* (iv) *complete meshes,* (v) *complete hypercubes,* (vi) *permutation graphs.*

6.3. Hardness of CFG-SIP. We show that CFG-SIP, the problem of finding a simple path complying with a CFL, is NP-hard even on graphs with bounded treewidth. Before formally stating the proof, we give the overall idea. We present a reduction from 3-SAT (see, e.g., [GJ79] for definition). The basic idea is to have a path consisting of two subpaths. The first subpath uniquely chooses an assignment and creates several identical copies of it. The second subpath checks one clause at every copy of the assignment and can only reach the destination if the assignment satisfies the given formula.

Consider the language $L = \{w\#w^R\$w\#w^R \cdots w\#w^R | w \in \Sigma^*\}$. As is standard, w^R denotes the reverse of string w. At the heart of our reduction is the crucial observation that L can be expressed as the intersection of two CFLs L_1 and L_2. Consider $L_1 = \{\, w_0\#w_1\$w_1^R\#w_2\$w_2^R t \cdots w_k\$w_k^R\#w_{k+1} \,|\, w_i \in \Sigma^* \,\}$ and $L_2 = \{\, v_1\#v_1^R\$v_2\#v_2^R \cdots v_k\#v_k^R \,|\, v_i \in \Sigma^* \,\}$. To see that $L = L_1 \cap L_2$, observe that $w_0 = v_1,\ v_1^R = w_1$, and $\forall\ i,\ w_i^R = v_{i+1}, v_{i+1}^R = w_{i+1}$, establishing that $v_i = v_{i+1}$ holds $\forall\ i$, and thus $L = L_1 \cap L_2$.

Now, imagine w representing an assignment to the variables of a 3-CNF formula with a fixed ordering of the variables. For every clause of the formula, we create a copy of this assignment. L is used to ensure that all these copies are consistent, i.e., identical.

Note that we have two basic objects for performing the reduction—a CFG and a labeled graph. We will specify L_1 as a CFG and use the labeled graph and simple paths through the graph to implicitly simulate L_2. Recall that there is a straightforward deterministic pushdown automaton M for accepting L_2. Our graph will consist of an "upward chain" of vertices and a "downward chain" of vertices along with a few additional vertices. The upward chain will simulate the behavior of M when it pushes w on the stack. The "downward chain" will then simulate popping the contents of the stack and verifying that they match w^R. We will call such a gadget a "tower" as an analogy to the stack of M.

We now describe the proof in detail. For the purposes of simplicity, we prove the results for directed graphs; the extension to undirected graphs follows with Lemma 16.

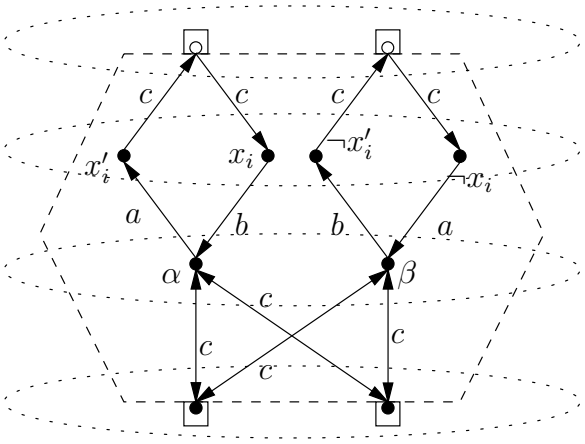


FIG. 5. *The gadget H_i used to implement the tower, forcing the assignment to be spread consistently over the graph. Dashed ellipses denote the so-called level sets.*

THEOREM 26. *The* CFG-SIP *problem is* NP-hard*, even for graphs of constant treewidth and a fixed deterministic CFL.*

Proof. Reduction from 3-SAT. Let $F(X, C)$ be a 3-CNF formula, where $X = \{x_1, \ldots, x_n\}$ denotes the set of variables and $C = \{c_1, \ldots, c_m\}$ denotes the set of clauses. Corresponding to F, we create an instance $G(V, E_1 \cup E_2)$ of CFG-SIP as follows. We will describe the reduction in two parts—first the subgraph (V, E_1) and then the subgraph (V, E_2).

The subgraph (V, E_1) is constructed as follows: Corresponding to each clause c_j, we have a tower T^j. It consists of n "simple-path stack-cell" gadgets H for each variable one. This basic gadget is depicted in Figure 5.

Consider a simple path p from one of the bottom nodes (marked by a square in the figure) to one of the top nodes. Because of the labels of this gadget, we can define the *signature* y of this path by $l(p) = cyc$ with $y \in \{a, b\}$. Let q be another simple path that has no vertex in common with p, starts at one of the top nodes, and ends in one of the bottom nodes. Then we can again properly define the signature z of this path by $l(q) = czc$.

Because of the nodes α and β, the signatures are identical, i.e., $y = z$. Furthermore, if p uses the node x_i', the node x_i is not used at all, and q has to use the node $\neg x_i$. Similarly, if p uses the node $\neg x_i'$, the node $\neg x_i$ is not used at all, and q has to use the node x_i.

These gadgets are now composed to form towers T^j, by identifying the top terminal nodes of H_i^j with the bottom terminal nodes of H_{i+1}^j. The tower has four levels corresponding to every variable. We call this a *floor* of the tower. The bottom of the tower T^j is connected to the bottom of the tower T^{j+1}. The start vertex is connected to the bottom of the tower T^1. These connections are depicted in Figure 6. Before we describe the remaining edges, we discuss the properties of a tower.

Consider a tower T^j and a simple path r labeled according to $(c(a \cup b)c)^* \# (c(a \cup b)c)^*$ that starts at one bottom vertex and reaches the other bottom vertex. Such a path has the following important properties:

(1) The path r consists of two simple subpaths p and q separated by an edge labeled with $\#$. p starts at the bottom of the tower and is labeled according to the regular expression $(c(a \cup b)c)^*$. On every floor it uses exactly one of the nodes $\{x_i', \neg x_i'\}$,

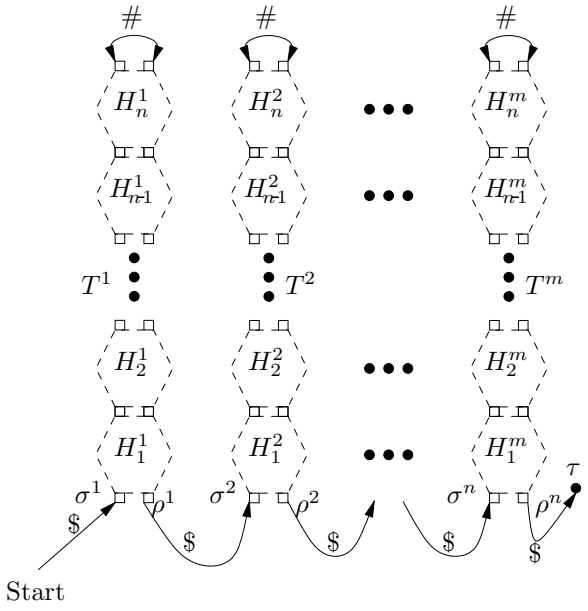


FIG. 6. *Assembling the gadgets, building the graph* G_1.

thus realizing an assignment to the variables of X. This assignment uniquely corresponds to the labeling of this path.

(2) The constraints of *simplicity* and the direction of edges imply that q has the following structure: it starts at the unused top vertex, is labeled with $l(q) = l(p)^R$, avoids already-used nodes, and reaches the unvisited bottom vertex. Furthermore q is *uniquely* determined given p.

(3) Note that for each variable x_i, the simple path r visits exactly one of the nodes in $\{x_i, \neg x_i\}$. If the path reflects an assignment in which x_i is true, then x_i is unused and $\neg x_i$ is used, and vice versa. These free nodes will be used in the second part of the reduction to verify that the chosen assignment is indeed satisfying F.

We now assemble the towers to form the graph G_1 depicted in Figure 6.

PROPOSITION 27. *Any simple path in* G_1 *starting at the start node* s *and reaching the intermediate node* τ*, with the constraint that the labeling belongs to* $\$((c(a \cup b)c)^*\#(c(a \cup b)c)^*\$)^*$*, generates the language*

$$L_2 = \{\, \$w_1\#w_1^R\$w_2\#w_2^R\$ \cdots w_n\#w_n^R\$ \mid w_i \in (c(a \cup b)c)^* \,\}.$$

Proof. The statement follows from the above. □

We now choose the constraining CFL for the path from s to τ to be

$$L_1 = \{\, \$w_1\#w_2\$w_2^R\#w_3\$w_3^R\# \ldots w_{k-1}\$w_{k-1}^R\#w_k\$ \mid k \in \mathbb{N}, w_i \in (c(a \cup b)c)^* \,\}$$

The following important lemma follows from Proposition 27 and the definition of L_1.

LEMMA 28.

(1) *Proposition* 27 *enforces that in every tower the used and unused nodes can be uniquely interpreted as an assignment to the variables of* X.
(2) L_1 *enforces that these assignments are consistent across two consecutive towers.*

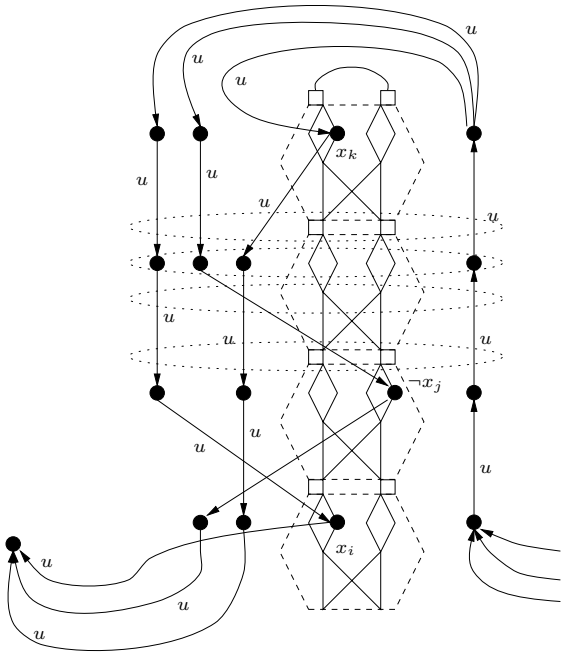


FIG. 7. *One tower of the second part of the graph, according to the clause* $(x_i \vee \neg x_j \vee x_k)$.

We now describe the subgraph (V, E_2). The label of each edge in this subgraph is u. The subgraph is composed of m subgraphs $D_1 \cdots D_m$, the subgraph D_i corresponding to clause c_i, depicted in Figure 7. Every D_i basically consists of four simple chains. The first one goes up the tower. There it splits into three downward directed paths. Each of them corresponds to a literal of c_i. The node in the tower T_i corresponding to that literal is used as part of the path. At the very bottom the three paths are joined and connected to D_{i-1}. At the boundaries D_0 is replaced by d and D_{m+1} by τ. This completes the description of the graph $G(V, E_1 \cup E_2)$.

The instance of CFG-SIP consists of the graph $G(V, E_1 \cup E_2)$ and the constraining CFL $L = L_1 \cdot u^*$. This forces the path to go through every tower using edges in $G(V, E_1)$, visit vertex τ, and then use the edges of $G(V, E_2)$ to reach d.

We now prove the correctness of the reduction. Suppose, there exists a satisfying assignment for F. Then we choose the path from s through all the towers to τ accordingly. For the remaining path to d we use the fact that for every clause of F at least one literal is true. Choosing the downward-directed subpath according to this literal we can complete a simple path to d. By construction, the label of this path complies with the CFG.

Conversely suppose there exists a simple path in G from s to d satisfying the labeling constraint. The alternation of # and \$ in L enforces that such a path visits every tower, then the vertex τ, and finally the destination. In this situation we can define an assignment A according to the path in the first tower. By Lemma 28, and the path's being simple, we know that A is consistently represented in all the towers. The second part of the path shows that A is a satisfying assignment for F.

Finally it is easy to verify that G has a constant treewidth. For this we describe a tree decomposition of the graph. The up to four nodes of one level in a gadget H_i

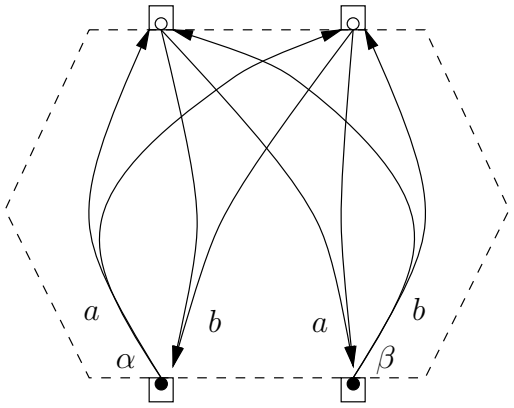


FIG. 8. *The smaller gadgets replacing the H_i used in the original construction. a stands for a_1, a_2, a_3; b for b_1, b_2, b_3. To encode the clauses some of these labels get removed, as in the proof of Theorem* 20.

and the up to four nodes of the path D_i form so-called level-sets. Two neighboring level-sets together, and two bottom level-sets of neighboring towers, respectively, form a set of the tree-decomposition. Additional sets for s, d, and τ with all the adjacent nodes are needed. This shows that the treewidth is bounded by 16. □

Actually the result of Theorem 26 can be extended to obtain the following.

COROLLARY 29. CFG-SIP *is* NP*-hard when restricted to graphs of treewidth* 3 *and a fixed* linear, deterministic *CFG.*

Proof. The proof is similar to the proof of Theorem 26, and thus we describe only the necessary modifications needed to prove the corollary. First we can replace the second part of the graph, used to verify that the assignment satisfies the formula. This is done as in the proof of Theorem 20, by replacing a (and b) by three symbols a_1, a_2, and a_3 (b_1, b_2, b_3). We modify the language such that for the context-free part all a_i are equivalent and all b_i are equivalent. Then we add a regular component to the language (take the intersection), enforcing that on one leg all the indices have to be identical. By removing certain labels we can make sure that in every tower one clause is tested.

Since we do not need the free nodes anymore, the gadget H_i depicted in Figure 5 can be replaced by a much smaller one, depicted in Figure 8. Note that replacing parallel directed links by disjoint paths of length 2 does not increase the treewidth in this situation. The additional nodes and edges can be covered in a tree-decomposition by some more leaf-sets of size 3. This does not increase the width of the tree-decomposition. In total the treewidth of the modified construction is reduced to 3.

Taking a new alphabet symbol x we define the language $H = \{\ wxw^P\#v | w \in L_1, v \in (\Sigma - \{\#\})^*\ \}$. It is easy to see that H can be specified by a linear CFG if L_1 is a regular language. Here w^P is defined to be the reverse of w with bottom marker \$ and top marker # exchanged. We take L_1 to be a regular language that forces the path to use the top marker of every tower. It is sufficient to argue that the proof of Theorem 26 can be modified such that the constraining CFL can be replaced by H.

For this we double the sequence of variables, introducing a new set of variables $\{x'_i = x_{2n-i+1}\}$ yielding the extended sequence $x_1, x_2, \ldots, x_n, x'_n, x'_{n-1}, \ldots, x'_1$. The new variables are used only while constructing the towers, but not for the evaluation

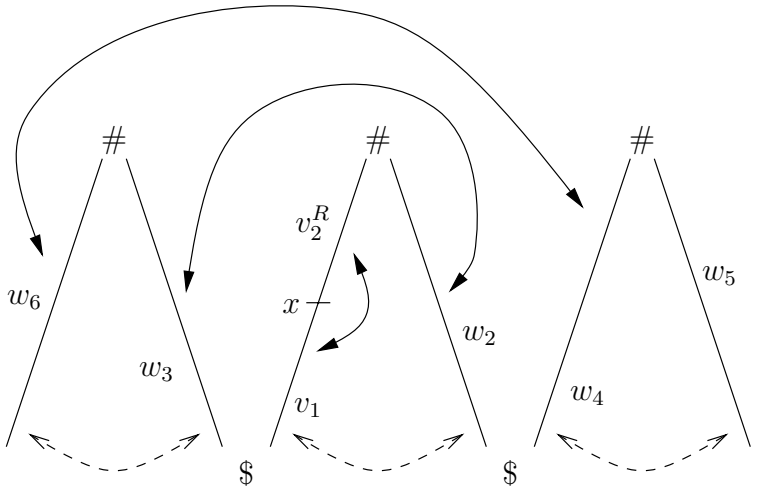


FIG. 9. *Schematic labeling of the path; $w_1 = v_1 v_2^R$; solid arrows stand for correspondence enforced by the language, dashed arrows for correspondence resulting from the graph.*

of clauses. Additionally we have to incorporate the middle marker x into the graph. Assuming the number of clauses m is odd, this means that we modify the tower $m/2$ at the level of the lowermost gadget $H_{n+1}^{m/2}$ created for the new variable x'_n. In this gadget we replace both upward edges by paths of length 2. The upper edge gets the original label, the lower edge gets labeled with x.

The proof of correctness consists of showing that the new H ensures a single assignment is represented by all the subpaths in the towers. This can be shown inductively from the middle symbol x outwards. The way the information is spread is depicted in Figure 9. The induction starts with the fact that the signature w_1 of the upward path in tower $m/2$ is forced by H to comply with $v_1 = v_2^R$, leading to the definition $w_1 = v_1 v_2^R$, with the property $w_1 = w_1^R$. So we do not have to distinguish between w_1 and w_1^R. Then the graph enforces $w_2 = w_1$, H enforces $w_3 = w_2$, and so on. This establishes that all the (lower halves of the) signatures are identical. The remaining part of the proof follows the details given for Theorem 26 closely. □

Note that the language H in the proof of Theorem 26 can also be accepted by a deterministic log-space-bounded Turing machine having a two-way input tape.

6.4. Algorithm for REG-SIP on graphs of bounded treewidth. In contrast to the above NP-hardness results we show that for graphs of bounded treewidth the problem REG-SIP is solvable in polynomial time. The class of treewidth-bounded graphs includes interval graphs and chordal graphs with *bounded clique size*, complete meshes, with fixed length or width, outer planar graphs, series parallel graphs, etc. We refer the reader to [AC+93, AL+91, Bo88, Bo92, BL+87] for other polynomial-time-solvable problems for treewidth-bounded graphs. This easiness result, along with the hardness result in the preceding section, also implies that these results are in a sense the best possible. We will use the notion of a nice tree-decomposition discussed in [Bo92].

DEFINITION 30. *A tree-decomposition $\langle \{ X_i \mid i \in I \}, T = (I, F) \rangle$ is* nice *if one can choose a root r in the tree such that*

- *T is a binary tree;*

- *if i is a leaf, then $|X_i| = 1$ (start node);*
- *if i has two children j and k, then $X_i = X_j = X_k$ (join node);*
- *if i has one child j, then there exists a vertex v such that either*
  - *$X_i = X_j \setminus \{v\}$ (forget node), or*
  - *$X_i = X_j \cup \{v\}$ (introduce node).*

We use the fact, that there exists always a nice tree-decomposition of optimal width (treewidth), and that it can be constructed in linear time [Bo92].

THEOREM 31. *The* REG-SIP *problem is solvable in polynomial time on treewidth-bounded graphs*

Proof. Given a treewidth-bounded graph G along with a nice tree-decomposition $T(I, F)$, we describe an algorithm that computes tables of (partial) shortest simple paths in a bottom-up fashion in T. Specifically, we have one table for every set X_i corresponding to the node i of T. We describe the entries of these tables and how to compute the values (i) for leaf sets and (ii) for internal nodes from the values in the tables of the child node(s). Since the number of values computed, as well as the number of tables, is polynomial in the size of G, this yields a polynomial-time algorithm.

The complicated task is to keep track of the simplicity of the path or, more precisely, the nodes used by the partial solutions represented by entries of the table. We have an entry for each type of path going through the set of nodes to which the table is attached. Since these sets are separators in G it is not necessary to keep the complete information about possible paths "behind" the separator. For the remainder of this section, we use k to denote the treewidth of the tree-decomposition T. In the following, for a set X_j corresponding to a node i in T we give a characterization of the distinct subsolutions that need to be maintained.

First we introduce the notion of an "atom": a simple path in the already "visited" part of the graph. An atom is characterized by

(1) a node in X_j the path starts;
(2) the state the automaton is in before reading in the label of the path;
(3) a node in X_j the path ends;
(4) the state of the automaton after having read in the label of the path.

We accept the special situation of one-node paths, i.e., indices with identical nodes and states.[3] There are two special "half" atoms standing for a beginning segment and an end segment of the path: There we have a node and a state identifying the end node of a simple path that starts at the source, such that its label can lead the automaton to the specified state. Similarly, for the end segment of a path we have a node and a state such that the labeling along the path gets the automaton from this state to an accepting state at the sink. Thus the subsolutions are completely described by

> the $O(k)$ atoms plus the two special half atoms together with a set of other used nodes in X_i.

We only need to allow the atoms to be empty, meaning that they do not denote any path. Noting that the total number of ways to partition a set of size k is $k^{O(k)}$ leads to an upper bound of $(2k \cdot |NFA|)^{O(k)}$ for the number of entries in the tables[4] which is polynomial in the size of the NFA for fixed treewidth k.

Note that the entry in the table where both of the special half atoms are empty stands for a complete solution in the already visited part of the graph. For every type

[3] Here we could actually account for ε moves in the automaton if we choose to.
[4] We have not attempted to optimize the size of the tables.

of partial solutions we maintain the total length of the shortest partial solution.

We now describe the tables more formally. For the leaf sets consisting of a single node v the table is easy to compute. It has entries with value 0 for the following partial solutions:

- $s \neq v \neq d$ for all states i: the length 0 path v-v, with start and end states i, no additional nodes used.
- $v = s$ the length 0-path from s to v ending in the initial state.
- $v = d$ for all accepting states $f \in F$: the length 0-path from v to d beginning in state f.

In order to compute the tables in a bottom-up fashion in T, we need to consider three possible cases depending on the type of nodes in T. Let i be a node in T.

i **is a join node:** (1) Letting j and l be the children of i, we know that $X_i = X_j = X_l$. Combine a partial solution stored in the table associated j with a partial solution stored in the table associated with l. For all pairs of types of partial solutions check if they can be combined to form another partial solution (no commonly used nodes, matching boundary node and state of the partial solution, respecting special cases for source and destination subpaths). Create the new type and compute the value of the combined solution. If the type does not yet exist in the table or the newly computed value is smaller than the old table entry, update the table entry. This is justified by the observation that the described subsolutions associated with j and k are disjoint except at the set of separator nodes X_i.

(2) Keep the componentwise min of the tables: for all types of partial solutions (their descriptions match as the two sets of the decomposition are identical), keep the smaller value. If the solution according to a type does not exist, we assume its value to be infinity. (This can also be seen as (1), where we combine the entry with an "empty subsolution" of cost 0 from the left and from the right and then choose the better one.)

i **is a forget node:** Let v be the node removed from the set in the nice tree-decomposition. We discard all partial solutions that contain a path with endpoint v. In all partial solutions, we delete v in the set of used nodes. For the resulting identical subsolutions we keep the one with minimum value.

i **is an introduce node:** Let v be the new node, e_i the edges between v and nodes in the set of the child.

(1) Set up the new node. Copy all known partial solutions. Create new partial solutions by combining all known with all solutions created according to the rules stated above for leaf sets of the form $\{v\}$.

(2) Include the incident edges one by one. Consider all possible new paths using this edge.

The correctness of the algorithm follows by noting the following:

(1) Given an entry in the root of the table for the existence of a path p of a given length, we can easily find such a path recursively from subsolutions associated with the children of the root.

(2) Conversely, let p be one of the optimal (shortest) solutions. Let X_r be the set associated with the root of T. Assume that r is a join node and let l and w be the left and right children. The argument for the other two cases is similar and thus omitted. It is easy to see that the path p can be broken into two paths p_1 and p_2 (p_2 could be empty) such that p_1 is a subsolution maintained at l and p_2 is subsolution maintained at w. □

The following generalization of the previous result holds.

COROLLARY 32. *Let L be a fixed language decidable by a one-way input tape,* log*-space-bounded nondeterministic Turing machine (NDTM). Then the problem of finding a shortest simple path from source to destination according to L is solvable in polynomial time on treewidth bounded graphs.*

Proof. The length of a simple path in the graph is bounded by n. The number of configurations of an NDTM using $\log n$ tape cells is polynomial in n. No other configurations of the machine can be used while recognizing a word of length smaller than or equal to n. As the machine reads the input tape in one direction only, we can create an NFA with states representing configuration that (i) decides L for words of length $\leq n$ and (ii) has a size polynomial in n. Using the algorithm in the proof of Theorem 31 with NFA M, we can compute the sought path in polynomial time. □

Note that $a^n b^n c^n$ and all languages accepted by pushdown automata having only one stack-symbol are examples of such languages.

6.5. Algorithm for acyclic graphs. Another well-known situation in which shortest simple paths are feasible is for acyclic graphs. This stems from the fact that all paths in an acyclic graph are simple and by this the shortest and the shortest simple path always coincide. This, together with the results of section 5, yields the following result.

COROLLARY 33. *The problem* CFG-SIP *is solvable in polynomial time on directed acyclic graphs.*

Making use of the fact that the length of a simple path is bounded by the size of the graph, we get the following.

COROLLARY 34. *The problem of finding a shortest simple simple from source to destination in a directed acyclic graph according to a formal language L is solvable in polynomial time if there exists a polynomial time computable CFL R of size polynomial in n with the following property:*

$$x \in \Sigma^*,\ |x| \leq n :\ \ (x \in L \leftrightarrow x \in L(R)).$$

7. Extensions and applications. In this section, we briefly discuss extensions and applications of our results to problems in transportation science. For many of these applications, it is possible to devise dynamic-programming-based methods to solve the problems; our aim is to convey the applicability of the general methodology proposed here.

7.1. Node labels and trip chaining. Consider the problem where the nodes have labels, instead of the edges and the constraint is on the concatenated node labels of a path. Easy transformations of the input show that all the results we develop for the edge-labeled case are also true for the node-labeled case. We can transform this kind of instance into an edge-labeled one by the following steps: The network stays the same, the edges get labeled with a new symbol. Every node gets an additional loop attached. This loop gets the label(s) of the node. Then the language has to be extended such that (i) exactly every second symbol has to be the new edge symbol and (ii) the word without the edge symbols is in the original language. It is easy to see that regular and context-free languages are closed under this operation. This shows, that easiness results for edge label constraints imply easiness results for node label constraints. If we have an edge-labeled graph, we split the edges and insert a node with the edge label, and label all the old nodes with one new symbol. The language has

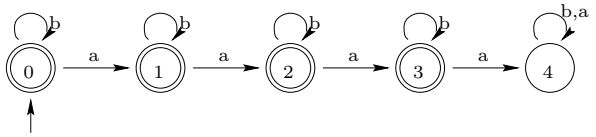


FIG. 10. *Automaton used to count shared links, accepting if less than* 4.

to be extended as above. This construction transfers edge-label-constrained hardness results to node-label-constrained results.

The node label extensions turn out to be useful in modeling transportation-related problems. In many transportation applications (e.g., TRANSIMS) one needs to find paths for travelers who need to visit a fixed sequence of location types. For instance, we might want to find a shortest path from home to home that visits some locations, e.g., ATM–gas station–supermarket–post office, in this particular order, but with the freedom to choose one of the several ATMs (gas stations, etc.) in the city. Problems of this type are referred to in the transportation literature as *trip chaining* problems. The problem cannot be solved by direct application of Dijkstra's shortest path algorithm to find best paths between two consecutive subdestinations and concatenating these paths. By treating each destination type as a node label and constructing a simple regular expression, we can select places for the subdestinations and find shortest path in networks that satisfy the precedence constraints using the polynomial result discussed in this paper.

7.2. Finding alternatives to shortest paths. There has been considerable interest in algorithms for variations of shortest paths (see [AMO93, Ta81, HNR68, OR90, Ha92, BAL97]). For example, in a recent paper by Scott, Pabon-Jimenez, and Bernstein [SJB97], the authors consider the following problem: given a graph G, a shortest path SP in G, and an integer parameter, find the shortest path $SP1$ in G that has at most k links in common with SP. Call this the BEST k-SIMILAR PATH problem. In [SJB97], the authors present an integer linear program formulation of the problem and present a heuristics based on Lagrangian relaxation. It can be verified that the heuristic takes exponential time in the worst case. Quoting [SJB97], *"The link overlap constraint thus makes finding the best path much more difficult (i.e., shortest path problems with a single constraint are* NP*-hard)"*; thus suggesting that the BEST k-SIMILAR PATH problem is likely to be NP-hard. Here we show that this problem is solvable in time $T(k|G|)$, where $T(n)$ denotes the time taken to find a shortest path on a graph with n vertices. This substantially improves upon the exponential-time algorithm given in [SJB97]. Our approach for solving the problem is based on using our algorithm for regular-expression-based shortest paths. The approach uses the fact that, given a k and symbol $a \in \Sigma$, the language consisting of all words $w \in \Sigma^*$ with no more than k occurrences of the symbol a in them is regular. Given a graph G, a shortest path SP in G and an integer parameter k, we perform the following steps:

(1) Label the edges on the shortest paths by a and all the other edges in G by b.
(2) Construct an NFA M that accepts all strings that have no more than k occurrences of a. The corresponding automaton M is shown in Figure 10 (and happens to be deterministic).
(3) Find a shortest path in G with the constraint that its label is in $L(M)$.

The proof of correctness is straightforward. We note that in this particular case, we in fact always get a simple path, since we can always remove a loop from a path without increasing its length or acceptability of the shorter string.

7.3. Handling left-turn and U-turn penalties. Suppose we are told that in our road network there is additional cost incurred when we take certain left turns. This is a common scenario in transportation science and is referred to as turn penalties [HM95]. This is not an unlikely scenario. In fact the current Dallas Fort Worth case study has exactly this situation. Specifically, several left turns in the study area are prohibited, which amounts to saying that the cost of taking that edge if it is a left turn is ∞. Moreover, the infrastructure change that is proposed for the case study intends to make a series of left turns illegal near the area of Valley View Mall and the Galleria.

A well-known reduction from the original problem to the problem of finding a shortest path in a modified network (see [AMO93] for details) can be used to solve this problem. (The basic idea is to replace each intersection by a clique of size 4. A slightly more complicated subgraph is required for directed graphs.) Instead of giving a penalty to a turn, suppose we wish to find a path in which we do not take more than, say, k left turns. This variant of the problem cannot be solved by a direct application of Dijkstra's algorithm, but is amenable to an efficient solution using the formal language approach. To do this we again replace each intersection by a clique of size 4 and then add appropriate labels to each of these edges. We then construct an automaton, which accepts strings that contain at most k labels corresponding to left turns. This can be constructed along the lines of the k-SIMILAR PATH problem. The rest of the details are straightforward. Now consider a more complicated query in which we wish to find a path such that the number of left turns is a small fraction of the total number of edges. It is easy to see that this can be written as a CFG and thus again can be solved efficiently.

7.4. Time-dependent networks and multicriteria shortest paths. In several transportation applications, it is desirable to solve shortest path problems on networks in which the edge weights are a function of time. In [OR90] Orda and Rom consider this type of problem for various waiting policies and function classes. One of their basic algorithms is a dynamic programming on functions. Combining this with our results, we obtain a polynomial-time algorithm for CFL-constrained shortest paths in time-depending networks.

As a final application, consider bicriteria and in general multicriteria shortest path problems. For instance, we might have two different weight functions on each edge a function $c(e)$ that captures the cost of using that edge and a function $t(e)$ that captures the time it takes to traverse the edge. The aim of the bicriteria shortest path problem aims at finding a minimum cost path from a source s to a destination d, that obeys a given budget bound B on the time taken to go from s to d. This problem has been studied extensively in the literature (see [MRS+95] and the references therein). Given that the cross product construction simply constructs multiple copies of the basic graph, it is easy to design a polynomial-time approximation scheme for the bicriteria shortest path problem subject to labeling constraints. The idea is a straightforward combination of the ones outlined here and those used for designing approximation schemes for the basic bicriteria problem.

7.5. Other formal languages. As expected, attempts to extend the polynomial-time algorithms to more general grammars such as the context-sensitive grammars fail to yield polynomial-time results. Intuitively, the hardness of the problems is due to the fact that *emptiness* and *recognition problems* for context-sensitive grammars are undecidable and PSPACE-complete, respectively.

Consider the problem for context-sensitive grammars. It is easy to show that

the problem (CSG-SHP) is undecidable even for a fixed log-space decidable context-sensitive language. It is easy to see that the language

$$L = \left\{ w\#^i \mid w \in \{0,1\}^*,\ \text{TM}(w) \text{ halts on an empty input using space } \log i \right\}$$

is context sensitive. It is in fact log-space computable. It is also straightforward to see that

$$L' = \{w \mid \exists i\ w\#^i \in L\}$$

is another way of stating the halting problem and is thus undecidable.

Showing that we can use CSG-SHP to decide L' establishes that CSG-SHP is itself undecidable. For $w \in \{0,1\}^*$ we construct a directed chain labeled according to w with start s and end d. We additionally put a loop from d to d labeled with $\#$. The fixed constraining language is L. Now $w \in L'$ is equivalent to the existence of a (in general not simple) path from s to d in this graph. Note that this graph is nearly a tree.

In contrast, the CSG-SIP is PSPACE-complete. Membership in PSPACE follows by observing that a simple path in a graph $G(V, E)$ can have at most $|V|$ nodes. Thus a space-bounded NDTM [HU79] can guess a simple path p and then verify that $l(p) \in L(M)$, where M is the context-sensitive grammar. The hardness is shown by reducing the problem of deciding if $w \in L(M)$ to finding a simple path in a directed chain that is labeled according to w. Let s and t denote the two endpoints of this chain. Then there is a path from s to t whose labels are in $L(R)$ if and only if $w \in L(R)$. Similar extensions hold for other formal languages. Note that for this argument the context-sensitive language can be fixed to represent an arbitrary PSPACE-complete problem.

8. Conclusions. In this paper, we have presented a general approach for modeling and solving a number of problems that seek to find paths subject to certain labeling constraints. The model was shown to be particularly useful in understanding and solving transportation science problems. The results in this paper provide a fairly tight characterization of the complexity of these problems, varying the type of considered path, the underlying grammar, and allowed graph classes. For a number of nontrivial cases this completely characterizes the boundary between easy and hard cases. Our results can also be seen investigating tradeoffs between (i) economy of descriptions of languages used to describe labeling constraints and (ii) the efficient solvability of the corresponding problems.

In [BK+99, JBM99, JMN98], we have obtained a number of additional theoretical as well as empirical results on related topics. Specifically in [JBM99], we show how applying our results on finding REG-SHP in time-dependent networks can be used to solve a number of additional problems arising in transportation science. In [BK+99, JMN98], we have carried out a detailed experimental analysis to validate the suitability of extensions of the algorithm suggested here on realistic transportation networks. We refer the reader to the Web site http://transims.tsasa.lanl.gov for comprehensive information about TRANSIMS and related documents.

The results presented here raise a number of questions for further investigation. First, it would be of interest to characterize the class of fixed regular (context-free) languages for which the regular-expression-constrained simple path problems are solvable in polynomial time. It would also be of interest to provide natural formulation of other label-constrained subgraph problems. For example, what is a natural way

to specify labeling constraints for spanning trees of a graph? A number of possible ways present themselves; the aim is to find ways that are both natural and useful in modeling practical problems.

Acknowledgments. We want to thank Harry Hunt III, Sven Krumke, Alberto Mendelzon, S. S. Ravi, R. Ravi, Dan Rosenkrantz, M. Yannakakis, Hans-Christoph Wirth, Stephen Eubank, Kai Nagel, Terence Kelly, Robert White, and Brian Bush for several useful discussions, pointers to related literature, and suggested improvements in the course of writing this paper. We also thank the anonymous referee for a careful reading of the manuscript and a number of helpful comments on the earlier draft.

REFERENCES

[AV99] S. Abiteboul and V. Vianu, *Regular path queries with constraints*, J. Comput. System Sci., 58 (1999), pp. 428–452.

[BaL] M. Ben-Akiva and S. R. Lerman, *Discrete Choice Analysis*, MIT Press Ser. Transportation Stud., MIT Press, Cambridge, MA, 1985.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows*: *Theory, Algorithms and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[AC+93] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese, *An algebraic theory of graph reductions*, J. ACM, 40 (1993), pp. 1134–1164.

[AMM97] G. Arocena, A. Mendelzon, and G. Mihaila, *Applications of a Web query language*, in Proceedings of the 6th International WWW Conference, Santa Clara, CA, 1997.

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison–Wesley, Reading, MA, 1974.

[AB+97] D. Anson, C. Barrett, D. Kubicek, M. Marathe, K. Nagel, M. Rickert, and M. Stein, *Engineering the Route Planner for Dallas Case Study*, Technical report, Los Alamos National Laboratory, Los Alamos, NM, 1997.

[AL+91] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, J. Algorithms, 12 (1991), pp. 308–340.

[Bo88] H. L. Bodlaender, *Dynamic programming on graphs of bounded treewidth*, in Proceedings of the 15th International Colloquium on Automata Language and Programming, Lecture Notes in Comput. Sci. 317, Springer-Verlag, New York, 1988, pp. 105–118.

[Bo92] H. L. Bodlaender, *A tourist guide through treewidth*, Technical report RUU-CS-92-12, Utrecht University, The Netherlands, 1992.

[BL+87] M. W. Bern, E. L. Lawler, and A. L. Wong, *Linear-time computation of optimal subgraphs of decomposable graphs*, J. Algorithms, 8 (1987), pp. 216–235.

[BAL97] V. Blue, J. Adler, and G. List, *Real-time multiple objective path search for in-vehicle route guidance systems*, in Proceedings of the 76th Annual Meeting of the Transportation Research Board, Washington, D.C., 1997.

[BKV91] A. L. Buchsbaum, P. C. Kanellakis, and J. S. Vitter, *A data structure for arc insertion and regular path finding*, in Proceedings of the 1st ACM-SIAM Symposium on Discrete Algorithms, San Francisco, CA, SIAM, Philadelphia, PA, 1990, pp. 22–31.

[BK+99] C. Barrett, R. Jacob, G. Konjevod, R. Beckman, K. Bisset, and M. Marathe, *Routing Problems in TRANSIMS: Theory, Practice and Validation*, Technical report, Los Alamos National Laboratory, Los Alamos, NM, 1999.

[CMW87] M. Cruz, A. Mendelzon, and P. Wood, *A graphical query language supporting recursion,* in Proceedings of the 9th ACM SIGMOD Conference on Management of Data, San Francisco, CA, ACM, New York, 1987, pp. 323–330.

[CMW88] I. Cruz, A. Mendelzon, and P. Wood, *G^+: Recursive queries without recursion*, in Proceedings of the 2nd International Conference on Expert Data Base Systems, Tysons Corner, CA, ACM, New York, 1988, pp. 25–27.

[CLR] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, McGraw-Hill, New York, 1990.

[GJ79] M. R. Garey and D. S. Johnson, *Computers and intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[Ha88] F. G. Halasz, *Reflections on notecards: Seven issues for the next generation of hypermedia systems*, Comm. ACM, 31 (1988), pp. 836–852.

[HNR68] P. Hart, N. Nilsson, and B. Raphel, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Trans. System Sci. Cybernetics, 2 (1968), pp. 100–107.

[Ha92] R. Hassin, *Approximation schemes for the restricted shortest path problem*, Math. Oper. Res., 17 (1992), pp. 36–42.

[HM95] Highway Research Board, *Highway Capacity Manual*, Special Report 209, National Research Council, Washington, D.C., 1994.

[HU79] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison–Wesley, Reading, MA, 1979.

[Hu97] U. Huckenbeck, *Extremal paths in graphs. Foundations, search strategies, and related topics*, Math. Top. 10, Akademie Verlag, Berlin, 1997.

[JBM99] R. Jacob, C. Barrett, and M. Marathe, *Models and efficient algorithms for class of routing problems in time dependent and labeled networks*, Technical report, Los Alamos National Laboratory, Los Alamos, NM, 1999.

[JMN98] R. Jacob, M. Marathe, and K. Nagel, *A computational study of routing algorithms for realistic transportation networks*, in Proceedings of the 2nd Workshop on Algorithmic Engineering, Saarbrucken, Germany, 1998; ACM J. Exp. Algorithmics, to appear.

[Ku77] D. E. Knuth, *A generalization of Dijkstra's algorithm*, Inform. Process. Lett., 6 (1977), pp 1–5.

[HRS76] H. B. Hunt III, D. Rosenkrantz, and T. Szymanski, *On the equivalence, containment and covering problems for regular and context free grammars*, J. Comput. System Sci., 12 (1976), pp. 222–268.

[MW95] A. O. Mendelzon and P. T. Wood, *Finding regular simple paths in graph databases*, SIAM J. Comput., 24 (1995), pp. 1235–1258.

[MRS+95] M. V. Marathe, R. Ravi, R. Sundaram, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, *Bicriteria network design problems*, J. Algorithms, 28 (1998), pp. 142–171.

[Pa94] C. Papadimitriou, *Computational Complexity*, Addison–Wesley, Reading, MA, 1994.

[Ro88] J. Roumeuf, *Shortest paths under rational constraints*, Inform. Process. Lett., 28 (1988), pp. 245–248.

[SJB97] K. Scott, G. Pabon-Jimenez, and D. Bernstein, *Finding alternatives to the best path*, in Proceedings of the 76th Annual Meeting of the Transportation Research Board, Washington, D.C., 1997.

[Str94] H. Straubing, *Finite Automata, Formal Logic, and Circuit Complexity*, Progr. Theoret. Comput. Sci., Birkhäuser Boston, Boston, MA, 1994.

[TR+95a] C. Barrett, K. Birkbigler, L. Smith, V. Loose, R. Beckman, J. Davis, D. Roberts, and M. Williams, *An Operational Description of TRANSIMS*, Technical report LA-UR-95-2393, Los Alamos National Laboratory, Los Alamos, NM, 1995.

[TR+95b] L. Smith, R. Beckman, K. Baggerly, D. Anson, and M. Williams, *Overview of TRANSIMS, the TRansportation ANalysis and SIMulation System*, Technical report LA-UR-95-1641, Los Alamos National Laboratory, Los Alamos, NM, 1995.

[Ta81] R. Tarjan, *A unified approach to path problems*, J. ACM, 28-3 (1981), pp. 577–593.

[OR90] A. Orda and R. Rom, *Shortest path and minimium delay algorithms in networks with time dependent edge lengths*, J. ACM, 37 (1990), pp. 607–625.

[RR96] G. Ramalingam and T. Reps, *An incremental algorithm for a generalization of the shortest-path problem*, J. Algorithms, 21 (1996), pp. 267–305.

[Ya90] M. Yannakakis *Graph theoretic methods in database theory*, in Proceedings of the 9th ACM SIGACT-SIGMOD-SIGART Symposium on Database Systems, Nashville, TN, ACM, New York, 1990, pp. 230–242.

[Ya95] M. Yannakakis, *Perspectives in database theory*, in Proceedings of the 36th Annual IEEE Symposium on Foundation of Computer Science, ACM, New York, 1995, pp. 224–246.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 838–846

A FAST GENERAL METHODOLOGY FOR INFORMATION-THEORETICALLY OPTIMAL ENCODINGS OF GRAPHS*

XIN HE†, MING-YANG KAO‡, AND HSUEH-I LU§

Abstract. We propose a fast methodology for encoding graphs with information-theoretically minimum numbers of bits. Specifically, a graph with property π is called a *π-graph*. If π satisfies certain properties, then an n-node m-edge π-graph G can be encoded by a binary string X such that (1) G and X can be obtained from each other in $O(n \log n)$ time, and (2) X has at most $\beta(n)+o(\beta(n))$ bits for any continuous superadditive function $\beta(n)$ so that there are at most $2^{\beta(n)+o(\beta(n))}$ distinct n-node π-graphs. The methodology is applicable to general classes of graphs; this paper focuses on planar graphs. Examples of such π include all conjunctions over the following groups of properties: (1) G is a planar graph or a plane graph; (2) G is directed or undirected; (3) G is triangulated, triconnected, biconnected, merely connected, or not required to be connected; (4) the nodes of G are labeled with labels from $\{1, \ldots, \ell_1\}$ for $\ell_1 \leq n$; (5) the edges of G are labeled with labels from $\{1, \ldots, \ell_2\}$ for $\ell_2 \leq m$; and (6) each node (respectively, edge) of G has at most $\ell_3 = O(1)$ self-loops (respectively, $\ell_4 = O(1)$ multiple edges). Moreover, ℓ_3 and ℓ_4 are not required to be $O(1)$ for the cases of π being a plane triangulation. These examples are novel applications of small cycle separators of planar graphs and are the only nontrivial classes of graphs, other than rooted trees, with known polynomial-time information-theoretically optimal coding schemes.

Key words. data compression, graph encoding, planar graphs, triconnected graphs, biconnected graphs, triangulations, cycle separators

AMS subject classifications. 05C10, 05C30, 05C78, 05C85, 68R10, 65Y25, 94A15

PII. S0097539799359117

1. Introduction.

Let G be a graph with n nodes and m edges. This paper studies the problem of *encoding* G into a binary string X with the requirement that X can be *decoded* to reconstruct G. We propose a fast methodology for designing a coding scheme such that the bit count of X is information-theoretically optimal. Specifically, a function $\beta(n)$ is *superadditive* if $\beta(n_1)+\beta(n_2) \leq \beta(n_1+n_2)$. A function $\beta(n)$ is *continuous* if $\beta(n + o(n)) = \beta(n) + o(\beta(n))$. For example, $\beta(n) = n^c \log^d n$ is continuous and superadditive, for any constants $c \geq 1$ and $d \geq 0$. The continuity and superadditivity are closed under additions. A graph with property π is called a *π-graph*. If π satisfies certain properties, then we can obtain an X such that (1) G and X can be computed from each other in $O(n \log n)$ time, and (2) X has at most $\beta(n)+o(\beta(n))$ bits for any continuous superadditive function $\beta(n)$ so that there are at most $2^{\beta(n)+o(\beta(n))}$ distinct n-node m-edge π-graphs. The methodology is applicable to general classes of graphs; this paper focuses on planar graphs.

*Received by the editors July 22, 1999; accepted for publication (in revised form) January 31, 2000; published electronically August 9, 2000. A preliminary version appeared in *Proceedings of the 7th Annual European Symposium on Algorithms*, Lecture Notes in Comput. Sci. 1643, Springer-Verlag, New York, 1999, pp. 540–549.

http://www.siam.org/journals/sicomp/30-3/35911.html

†Department of Computer Science and Engineering, State University of New York at Buffalo, Buffalo, NY 14260 (xinhe@cse.buffalo.edu).

‡Department of Computer Science, Yale University, New Haven, CT 06250 (kao-ming-yang@cs.yale.edu). This author's research was supported in part by NSF grant CCR-9531028.

§Institute of Information Science, Academia Sinica, Taipei 115, Taiwan, R.O.C. (hil@iis.sinica.edu.tw). Part of this work was performed at the Department of Computer Science and Information Engineering, National Chung-Cheng University, Chia-Yi 621, Taiwan, R.O.C. This author's research was supported in part by NSC grant NSC-89-2213-E-001-034.

A *conjunction* over k groups of properties is a boolean property $\pi_1 \wedge \cdots \wedge \pi_k$, where π_i is a property in the ith group for each $i = 1, \ldots, k$. Examples of suitable π for our methodology include every conjunction over the following groups:

F1. G is a planar graph or a plane graph.
F2. G is directed or undirected.
F3. G is triangulated, triconnected, biconnected, merely connected, or not required to be connected.
F4. The nodes of G are labeled with labels from $\{1, \ldots, \ell_1\}$ for $\ell_1 \leq n$.
F5. The edges of G are labeled with labels from $\{1, \ldots, \ell_2\}$ for $\ell_2 \leq m$.
F6. Each node of G has at most $\ell_3 = O(1)$ self-loops.
F7. Each edge of G has at most $\ell_4 = O(1)$ multiple edges.

Moreover, ℓ_3 and ℓ_4 are not required to be $O(1)$ for the cases of π being a plane triangulation. For instance, π can be the property of being a directed unlabeled biconnected simple plane graph. These examples are novel applications of small cycle separators of planar graphs [12, 11]. Note that the rooted trees are the only other nontrivial class of graphs with a known polynomial-time information-theoretically optimal coding scheme, which encodes a tree as nested parentheses using $2(n-1)$ bits in $O(n)$ time.

Previously, Tutte proved that there are $2^{\beta(m)+o(\beta(m))}$ distinct m-edge plane triangulations where $\beta(m) = (\frac{8}{3} - \log_2 3)m + o(m) \approx 1.08m + o(m)$ [17] and that there are $2^{2m+o(n)}$ distinct m-edge n-node triconnected plane graphs that may be nonsimple [18]. Turán [16] used $4m$ bits to encode a plane graph G that may have self-loops. Keeler and Westbrook [10] improved this bit count to $3.58m$. They also gave coding schemes for several families of plane graphs. In particular, they used $1.53m$ bits for a triangulated simple G, and $3m$ bits for a connected G free of self-loops and degree-1 nodes. For a simple triangulated G, He, Kao, and Lu [5] improved the bit count to $\frac{4}{3}m + O(1)$. For a simple G that is triconnected and thus free of degree-1 nodes, they [5] improved the bit count to at most $2.835m$ bits. This bit count was later reduced to at most $\frac{3\log_2 3}{2}m + O(1) \approx 2.378m + O(1)$ by Chuang et al. [2]. These coding schemes all take linear time for encoding and decoding, but their bit counts are not information-theoretically optimal. For labeled planar graphs, Itai and Rodeh [6] gave an encoding of $\frac{3}{2}n \log n + O(n)$ bits. For unlabeled general graphs, Naor [14] gave an encoding of $\frac{1}{2}n^2 - n \log n + O(n)$ bits.

For applications that require query support, Jacobson [7] gave a $\Theta(n)$-bit encoding for a connected and simple planar graph G that supports traversal in $\Theta(\log n)$ time per node visited. Munro and Raman [13] improved this result and gave schemes to encode binary trees, rooted ordered trees, and planar graphs. For a general planar G, they used $2m + 8n + o(m+n)$ bits while supporting adjacency and degree queries in $O(1)$ time. Chuang et al. [2] reduced this bit count to $2m + (5 + \frac{1}{k})n + o(m+n)$ for any constant $k > 0$ with the same query support. The bit count can be further reduced if only $O(1)$-time adjacency queries are supported, or if G is simple, triconnected, or triangulated [2]. For certain graph families, Kannan, Naor and Rudich [8] gave schemes that encode each node with $O(\log n)$ bits and support $O(\log n)$-time testing of adjacency between two nodes. For dense graphs and complement graphs, Kao, Occhiogrosso, and Teng [9] devised two compressed representations from adjacency lists to speed up basic graph search techniques. Galperin and Wigderson [4] and Papadimitriou and Yannakakis [15] investigated complexity issues arising from encoding a graph by a small circuit that computes its adjacency matrix.

Section 2 discusses the general encoding methodology. Sections 3 and 4 use the

methodology to obtain information-theoretically optimal encodings for various classes of planar graphs. Section 5 concludes the paper with some future research directions.

2. The encoding methodology. Let $|X|$ be the number of bits in a binary string X. Let $|G|$ be the number of nodes in a graph G. Let $|S|$ be the number of elements, counting multiplicity, in a multiset S.

FACT 1 (see [1, 3]). *Let $X_1, X_2, \ldots, X_k$ be $O(1)$ binary strings. Let $n = |X_1| + |X_2| + \cdots + |X_k|$. Then there exists an $O(\log n)$-bit string χ, obtainable in $O(n)$ time, such that given the concatenation of $\chi, X_1, X_2, \ldots, X_k$, the index of the first symbol of each X_i in the concatenation can be computed in $O(1)$ time.*

Let $X_1 + X_2 + \cdots + X_k$ denote the concatenation of $\chi, X_1, X_2, \ldots, X_k$ as in Fact 1. We call χ the *auxiliary binary string* for $X_1 + X_2 + \cdots + X_k$.

A graph with property π is called a *π-graph*. Whether two π-graphs are *distinct* or *indistinct* depends on π. For example, let G_1 and G_2 be two topologically non-isomorphic plane embeddings of the same planar graph. If π is the property of being a planar graph, then G_1 and G_2 are two indistinct π-graphs. If π is the property of being a planar embedding, then G_1 and G_2 are two distinct π-graphs. Let α be the number of distinct n-node π-graphs. Clearly it takes $\lceil \log_2 \alpha \rceil$ bits to differentiate all n-node π-graphs. Let $\text{index}_\pi(G)$ be an $\lceil \log_2 \alpha \rceil$-bit indexing scheme of the α distinct π-graphs.

Let G_0 be an input n_0-node π-graph. Let $\lambda = \log\log\log(n_0)$. The encoding algorithm $\text{encode}_\pi(G_0)$ is merely a function call $\text{code}_\pi(G_0, \lambda)$, where the recursive function $\text{code}_\pi(G, \lambda)$ is defined as follows:

```
function code_π(G, λ)
{
  if |G| = O(1) or |G| ≤ λ then
    return index_π(G)
  else
  {
    compute π-graphs G_1, G_2, and a string X, from which G can be recovered;
    return code_π(G_1, λ) + code_π(G_2, λ) + X;
  }
}
```

Clearly, the code returned by algorithm $\text{encode}_\pi(G_0)$ can be decoded to recover G_0. For notational brevity, if it is clear from the context, the code returned by algorithm $\text{encode}_\pi(G_0)$ (respectively, function $\text{code}_\pi(G, \lambda)$) is also denoted $\text{encode}_\pi(G_0)$ (respectively, $\text{code}_\pi(G, \lambda)$).

Function $\text{code}_\pi(G, \lambda)$ *satisfies the separation property* if there exist two constants c and r, where $0 \le c < 1$ and $r > 1$, such that the following conditions hold:

P1. $\max(|G_1|, |G_2|) \le |G|/r$.
P2. $|G_1| + |G_2| = |G| + O(|G|^c)$.
P3. $|X| = O(|G|^c)$.

Let $f(|G|)$ be the time required to obtain $\text{index}_\pi(G)$ and G from each other. Let $g(|G|)$ be the time required to obtain G_1, G_2, X from G, and vice versa.

THEOREM 2.1. *Assume that function $\text{code}_\pi(G, \lambda)$ satisfies the separation property and that there are at most $2^{\beta(n)+o(\beta(n))}$ distinct n-node π-graphs for some continuous superadditive function $\beta(n)$.*

1. *$|\text{encode}_\pi(G_0)| \le \beta(n_0) + o(\beta(n_0))$ for any n_0-node π-graph G_0.*
2. *If $f(n) = 2^{n^{O(1)}}$ and $g(n) = O(n)$, then G_0 and $\text{encode}_\pi(G_0)$ can be obtained from each other in $O(n_0 \log n_0)$ time.*

Proof. The theorem holds trivially if $n_0 = O(1)$. For the rest of the proof we assume $n_0 = \omega(1)$, and thus $\lambda = \omega(1)$. Many graphs may appear during the execution of $\text{encode}_\pi(G_0)$. These graphs can be organized as nodes of a binary tree T rooted at G_0, where (i) if G_1 and G_2 are obtained from G by calling $\text{code}_\pi(G, \lambda)$, then G_1 and G_2 are the children of G in T, and (ii) if $|G| \le \lambda$, then G has no children in T. Further consider the multiset S consisting of all graphs G that are nodes of T. We partition S into $\ell+1$ multisets $S(0), S(1), S(2), \ldots, S(\ell)$ as follows. $S(0)$ consists of the graphs G with $|G| \le \lambda$. For $i \ge 1$, $S(i)$ consists of the graphs G with $r^{i-1}\lambda < |G| \le r^i\lambda$. Let $G_0 \in S(\ell)$, and thus set $\ell = O(\log \frac{n_0}{\lambda})$.

Define $p = \sum_{H \in S(0)} |H|$. We first show

$$|S(i)| < \frac{p}{r^{i-1}\lambda} \tag{1}$$

for every $i = 1, \ldots, \ell$. Let G be a graph in $S(i)$. Let $S(0, G)$ be the set consisting of the leaf descendants of G in T; for example, $S(0, G_0) = S(0)$. By condition P2, $|G| \le \sum_{H \in S(0,G)} |H|$. By condition P1, no two graphs in $S(i)$ are related in T. Therefore $S(i)$ contains at most one ancestor of H in T for every graph H in $S(0)$. It follows that $\sum_{G \in S(i)} |G| \le \sum_{G \in S(i)} \sum_{H \in S(0,G)} |H| \le p$. Since $|G| > r^{i-1}\lambda$ for every G in $S(i)$, inequality (1) holds.

Statement 1. Suppose that the children of G in T are G_1 and G_2. Let $b(G) = |X| + |\chi|$, where χ is the auxiliary binary string for $\text{code}_\pi(G_1, \lambda) + \text{code}_\pi(G_2, \lambda) + X$. Let $q = \sum_{i \ge 1} \sum_{G \in S(i)} b(G)$. Then $|\text{encode}_\pi(G_0)| = q + \sum_{H \in S(0)} |\text{code}_\pi(H, \lambda)| \le q + \sum_{H \in S(0)} (\beta(|H|) + o(\beta(|H|)))$. By the superadditivity of $\beta(n)$, $|\text{encode}_\pi(G_0)| \le q + \beta(p) + o(\beta(p))$. Since $\beta(n)$ is continuous, Statement 1 can be proved by showing $p = n_0 + o(n_0)$ and $q = o(n_0)$ below.

By condition P3, $|X| = O(|G|^c)$. By Fact 1, $|\chi| = O(\log |G|)$. Thus, $b(G) = O(|G|^c)$, and

$$q = \sum_{i \ge 1} \sum_{G \in S(i)} O(|G|^c). \tag{2}$$

Now we regard the execution of $\text{encode}_\pi(G_0)$ as a process of growing T. Let $a(T) = \sum_{H \text{ is a leaf of } T} |H|$. At the beginning of the function call $\text{encode}_\pi(G_0)$, T has exactly one node G_0, and thus $a(T) = n_0$. At the end of the function call, T is fully expanded, and thus $a(T) = p$. By condition P2, during the execution of $\text{encode}_\pi(G_0)$, every function call $\text{code}_\pi(G, \lambda)$ with $|G| > \lambda$ increases $a(T)$ by $O(|G|^c)$. Hence

$$p = n_0 + \sum_{i \ge 1} \sum_{G \in S(i)} O(|G|^c). \tag{3}$$

Note that

$$\sum_{i \ge 1} \sum_{G \in S(i)} |G|^c \le \sum_{i \ge 1} (r^i\lambda)^c p/(r^{i-1}\lambda) = p\lambda^{c-1} r \sum_{i \ge 1} r^{(c-1)i} = p\lambda^{c-1} O(1) = o(p). \tag{4}$$

By (3) and (4), we have $p = n_0 + o(p)$, and thus $p = O(n_0)$. Therefore $\sum_{i \ge 1} \sum_{G \in S(i)} |G|^c = o(n_0)$. By (2) and (3), $p = n_0 + o(n_0)$ and $q = o(n_0)$, finishing the proof of Statement 1.

Statement 2. By conditions P1 and P2, $|H| = \Omega(\lambda)$ for every $H \in S(0)$. Since $\sum_{H \in S(0)} |H| = p = n_0 + o(n_0)$, $|S(0)| = O(n_0/\lambda)$. Together with (1), we know

$|S(i)| = O(\frac{n_0}{r^i\lambda})$ for every $i = 0, \ldots, \ell$. By the definition of $S(i)$, $|G| \leq r^i\lambda$ for every $i = 0, \ldots, \ell$. Therefore G_0 and $\text{encode}_\pi(G_0)$ can be obtained from each other in time

$$\frac{n_0}{\lambda} O\left(f(\lambda) + \sum_{1 \leq i \leq \ell} r^{-i} g(r^i\lambda)\right).$$

Clearly $f(\lambda) = 2^{\lambda^{O(1)}} = 2^{o(\log\log n_0)} = o(\log n_0)$. Since $\ell = O(\log n_0)$ and $g(n) = O(n)$, $\sum_{1 \leq i \leq \ell} r^{-i} g(r^i\lambda) = \sum_{1 \leq i \leq \ell} \lambda = O(\lambda \log n_0)$, and Statement 2 follows. □

Sections 3 and 4 use Theorem 2.1 to encode various classes of graphs G. Section 3 considers plane triangulations. Section 4 considers planar graphs and plane graphs.

3. Plane triangulations. A *plane triangulation* is a plane graph, each of whose faces has size exactly 3. For any plane triangulation P with n nodes, m edges, and f faces, Euler's formula ensures that $n - m + f = 2$ even if P contains self-loops and multiple edges. One can then obtain $m = 3n - 6$. Therefore every n-node plane triangulation, simple or not, has exactly $3n - 6$ edges.

In this section, let π be an arbitrary conjunction over the following groups of properties of a plane triangulation G: F2, F6, and F7, where ℓ_3 and ℓ_4 are not required to be $O(1)$. Our encoding scheme is based on the next fact.

FACT 2 (see [12]). *Let H be an n-node m-edge undirected plane graph, each of whose faces has size at most d. We can compute a node-simple cycle C of H in $O(n + m)$ time such that*

- *C has at most $2\sqrt{dn}$ nodes; and*
- *the numbers of H's nodes inside and outside C are at most $2n/3$, respectively.*

Let G be a given n-node π-graph. Let G' be obtained from the undirected version of G by deleting the self-loops. Clearly each face of G' has size at most 4. Let C' be a cycle of G' having size at most $4\sqrt{n}$ guaranteed by Fact 2. Let C consist of the edges of G corresponding to the edges of C' in G'. Note that C is not necessarily a directed cycle if G is directed. Since G' does not have self-loops, $2 \leq |C| \leq 4\sqrt{n}$. If $\ell_4 \geq 2$, then $|C|$ can be 2. Let G_{in} (respectively, G_{out}) be the subgraph of G formed by C and the part of G inside (respectively, outside) C. Let x be an arbitrary node on C.

G_1 is obtained by placing a cycle C_1 of three nodes outside G_{in} and then triangulating the face between C_1 and G_{in} such that a particular node y_1 of C_1 has degree strictly lower than the other two. Clearly this is feasible even if $|C| = 2$. The edge directions of $G_1 - G_{\text{in}}$ can be arbitrarily assigned according to π.

G_2 is obtained from G_{out} by (1) placing a cycle C_2 of three nodes outside G_{out} and then triangulating the face between C_2 and G_{out} such that a particular node y_2 of C_2 has degree strictly lower than the other two, and (2) triangulating the face inside C by placing a new node z inside of C and then connecting it to each node of C by an edge. Note that (2) is feasible even if $|C| = 2$. Similarly, the edge directions of $G_2 - G_{\text{out}}$ can be arbitrarily assigned according to π.

Let u be a node of G. Let v be a node on the boundary $B(G)$ of the exterior face of G. Define $\text{dfs}(u, G, v)$ as follows. Let w be the counterclockwise neighbor of v on $B(G)$. We perform a depth-first search of G starting from v such that (1) the neighbors of each node are visited in the counterclockwise order around that node, and (2) w is the second visited node. A numbering is assigned the first time a node is visited. Let $\text{dfs}(u, G, v)$ be the binary number assigned to u in the above depth-first search. Let $X = \text{dfs}(x, G_1, y_1) + \text{dfs}(x, G_2, y_2) + \text{dfs}(z, G_2, y_2)$.

LEMMA 3.1.

1. *G_1 and G_2 are π-graphs.*
2. *There exists a constant $r > 1$ with $\max(|G_1|, |G_2|) \leq n/r$.*
3. *$|G_1| + |G_2| = n + O(\sqrt{n})$.*
4. *$|X| = O(\log n)$.*
5. *G_1, G_2, X can be obtained from G in $O(n)$ time.*
6. *G can be obtained from G_1, G_2, X in $O(n)$ time.*

Proof. Statements 1–5 are straightforward by Fact 2 and the definitions of G_1, G_2, and X. Statement 6 is proved as follows. It takes $O(n)$ time to locate y_1 (respectively, y_2) in G_1 (respectively, G_2) by looking for the node with the lowest degree on $B(G_1)$ (respectively, $B(G_2)$). By Fact 1, it takes $O(1)$ time to obtain $\mathrm{dfs}(y_1, G_1, x)$, $\mathrm{dfs}(y_2, G_2, x)$, and $\mathrm{dfs}(y_2, G_2, z)$ from X. Therefore x and z can be located in G_1 and G_2 in $O(n)$ time by depth-first traversal. Now G_{in} can be obtained from G_1 by removing $B(G_1)$ and its incident edges. The cycle C in G_{in} is simply $B(G_{\mathrm{in}})$. Also, G_{out} can be obtained from G_2 by removing $B(G_2)$, z, and their incident edges. The C in G_{out} is simply the boundary of the face that encloses z and its incident edges in G_2. Since we know the positions of x in G_{in} and G_{out}, G can be obtained from G_{in} and G_{out} by fitting them together along C by aligning x. The overall time complexity is $O(n)$. ∎

THEOREM 3.2. *Let G_0 be an n_0-node π-graph. Then G_0 and $\mathrm{encode}_\pi(G_0)$ can be obtained from each other in $O(n_0 \log n_0)$ time. Moreover, $|\mathrm{encode}_\pi(G_0)| \leq \beta(n_0) + o(\beta(n_0))$ for any continuous superadditive function $\beta(n)$ such that there are at most $2^{\beta(n)+o(\beta(n))}$ distinct n-node π-graphs.*

Proof. Since an n-node π-graph has $O(n)$ edges, there are at most $2^{O(n \log n)}$ distinct n-node π-graphs. Thus, there exists an indexing scheme $\mathrm{index}_\pi(G)$ such that $\mathrm{index}_\pi(G)$ and G can be obtained from each other in $2^{|G|^{O(1)}}$ time. The theorem follows from Theorem 2.1 and Lemma 3.1. ∎

4. Planar graphs and plane graphs. In this section, let π be an arbitrary conjunction over the following groups of properties of G: F1, F2, F3, F6, and F7. Clearly an n-node π-graph has $O(n)$ edges.

Let G be an input n-node π-graph. For the cases of π being a planar graph rather than a plane graph, let G be embedded first. Note that this is only for the encoding process to be able to apply Fact 2. At the base level, we still use the indexing scheme for π-graphs rather than the one for embedded π-graphs. As shown below, the decoding process does not require the π-graphs to be embedded.

Let G' be obtained from the undirected version of G by (1) triangulating each of its faces that has size more than 3 such that no additional multiple edges are introduced, and then (2) deleting its self-loops. Let C' be a cycle of G' guaranteed by Fact 2. Let C consists of (a) the edges of G corresponds to the edges of C' in G', and (b) the edges of C' that are added into G' by the triangulation. (C is not necessarily a directed cycle of a directed G.) Let G_C be the union of G and C. Let G_{in} (respectively, G_{out}) be the subgraph of G_C formed by C and the part of G_C inside (respectively, outside) C. Let $C = x_1 x_2 \cdots x_\ell x_{\ell+1}$, where $x_{\ell+1} = x_1$. By Fact 2, $\ell = O(\sqrt{n})$.

LEMMA 4.1. *Let H be an $O(n)$-node $O(n)$-edge graph. There exists an integer k with $n^{0.6} \leq k \leq n^{0.7}$ such that H does not contain any node of degree k or $k-1$.*

Proof. Assume for a contradiction that such a k does not exist. It follows that the sum of degrees of all nodes in H is at least $(n^{0.6} + n^{0.7})(n^{0.7} - n^{0.6})/4 = \Omega(n^{1.4})$. This contradicts the fact that H has $O(n)$ edges. ∎

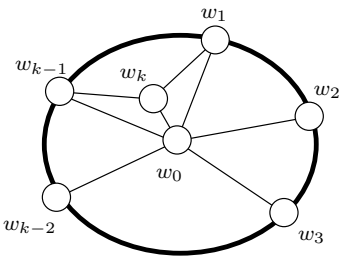


FIG. 1. *A k-wheel graph W_k.*

Let W_k, with $k \geq 3$, be a *k-wheel graph* defined as follows. As shown in Figure 1, W_k consists of $k+1$ nodes $w_0, w_1, w_2, \ldots, w_{k-1}, w_k$, where $w_1, w_2, \ldots, w_k, w_1$ form a cycle. w_0 is a degree-k node incident to each node on the cycle. Finally, w_1 is incident to w_{k-1}. Clearly W_k is triconnected. Also, w_1 and w_k are the only degree-4 neighbors of w_0 in W_k. Let k_1 (respectively, k_2) be an integer k guaranteed by Lemma 4.1 for G_{in} (respectively, G_{out}). Now we define G_1, G_2, and X as follows.

G_1 is obtained from G_{in} and a k_1-wheel graph W_{k_1} by adding an edge (w_i, x_i) for every $i = 1, \ldots, \ell$. Clearly for the case of π being a plane graph, G_1 can be embedded such that W_{k_1} is outside G_{in}, as shown in Figure 2(a). Thus, the original embedding of G_{in} can be obtained from G_1 by removing all nodes of W_{k_1}. The edge directions of $G_1 - G_{\text{in}}$ can be arbitrarily assigned according to π.

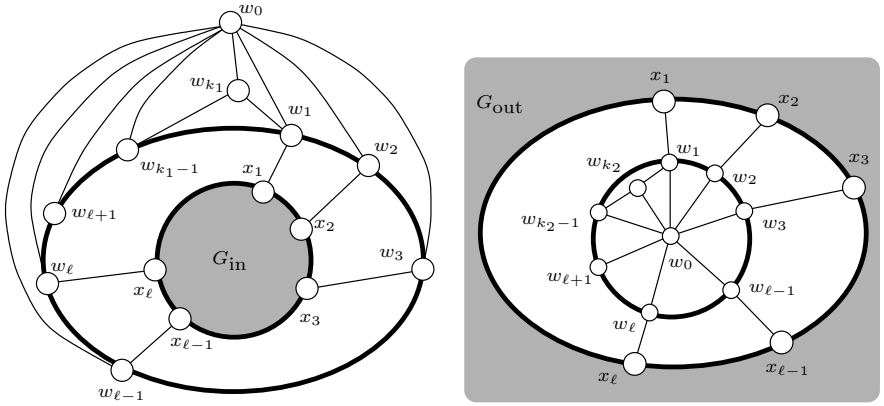
G_2 is obtained from G_{out} and a k_2-wheel graph W_{k_2} by adding an edge (w_i, x_i) for every $i = 1, \ldots, \ell$. Clearly for the case of π being a plane graph, G_2 can be embedded such that W_{k_2} is inside C, as shown in Figure 2(b). Thus, the original embedding of G_{out} can be obtained from G_2 by removing all nodes of W_{k_2}. The edge directions of $G_2 - G_{\text{out}}$ can be arbitrarily assigned according to π.

Let X be an $O(\sqrt{n})$-bit string which encodes k_1, k_2, and whether each edge (x_i, x_{i+1}) is an original edge in G, for $i = 1, \ldots, \ell$.

LEMMA 4.2.

1. *G_1 and G_2 are π-graphs.*
2. *There exists a constant $r > 1$ with $\max(|G_1|, |G_2|) \leq n/r$.*
3. *$|G_1| + |G_2| = n + O(n^{0.7})$.*
4. *$|X| = O(\sqrt{n})$.*
5. *G_1, G_2, X can be obtained from G in $O(n)$ time.*
6. *G can be obtained from G_1, G_2, X in $O(n)$ time.*

Proof. Since W_{k_1} and W_{k_2} are both triconnected, and each node of C has degree at least 3 in G_1 and G_2, statement 1 holds for each case of the connectivity of the input π-graph G. Statements 2–5 are straightforward by Fact 2 and the definitions of G_1, G_2, and X. Statement 6 is proved as follows. First of all, we obtain k_1 from X. Since G_{in} does not contain any node of degree k_1 or $k_1 - 1$, w_0 is the only degree-k_1 node in G_1. Therefore it takes $O(n)$ time to identify w_0 in G_1. w_{k_1} is the only degree-3 neighbor of w_0. Since $k_1 > \ell$, w_1 is the only degree-5 neighbor of w_0. w_2 is the common neighbor of w_0 and w_1 that is not adjacent to w_{k_1}. From now on, w_i, for each $i = 3, 4, \ldots, \ell$, is the common neighbor of w_0 and w_{i-1} other than w_{i-2}. Clearly, $w_1, w_2, \ldots, w_\ell$ and thus $x_1, x_2, \ldots, x_\ell$ can be identified in $O(n)$ time. G_{in} can now be obtained from G_1 by removing W_{k_1}. Similarly, G_{out} can be obtained from G_2 and X by deleting W_{k_1} after identifying $x_1, x_2, \ldots, x_\ell$. Finally, G_C can be recovered by fitting G_{in} and G_{out} together by aligning $x_1, x_2, \ldots, x_\ell$. Based on X, G can then be



(a) G_1 (b) G_2

FIG. 2. *G_1 and G_2. The gray area of G_1 is G_{in}. The gray area of G_2 is G_{out}.*

obtained from G_C by removing the edges of C that are not originally in G. □

Remark. In the proof for statement 6 of Lemma 4.2, identifying the degree-k_1 node (and the k_1-wheel graph W_{k_1}) does not require the embedding for G_1. Therefore the decoding process does not require the π-graphs to be embedded. This is different from the proof of Lemma 3.1.

THEOREM 4.3. *Let G_0 be an n_0-node π-graph. Then G_0 and* $\text{encode}_\pi(G_0)$ *can be obtained from each other in $O(n_0 \log n_0)$ time. Moreover,* $|\text{encode}_\pi(G_0)| \leq \beta(n_0) + o(\beta(n_0))$ *for any continuous superadditive function $\beta(n)$ such that there are at most $2^{\beta(n)+o(\beta(n))}$ distinct n-node π-graphs.*

Proof. Since there are at most $2^{O(n \log n)}$ distinct n-node π-graphs, there exists an indexing scheme $\text{index}_\pi(G)$ such that $\text{index}_\pi(G)$ and G can be obtained from each other in $2^{|G|^{O(1)}}$ time. The theorem follows from Theorem 2.1 and Lemma 4.2. □

5. Concluding remarks. For brevity, we left out F4 and F5 in sections 3 and 4. One can verify that Theorems 3.2 and 4.3 hold even if π is a conjunction over F1 through F7 including F4 and F5.

The coding schemes given in this paper require $O(n \log n)$ time for encoding and decoding. An immediate open question is whether one can encode some graphs other than rooted trees in $O(n)$ time using information-theoretically minimum number of bits. It would be of significance to determine whether the tight bound of the number of distinct π-graphs for each π is indeed continuous superadditive.

REFERENCES

[1] T. C. BELL, J. G. CLEARY, AND I. H. WITTEN, *Text Compression*, Prentice-Hall, Englewood Cliffs, NJ, 1990.

[2] R. C.-N. CHUANG, A. GARG, X. HE, M.-Y. KAO, AND H.-I LU, *Compact encodings of planar graphs via canonical orderings and multiple parentheses*, in Automata, Languages and Programming, 25th Colloquium, K. G. Larsen, S. Skyum, and G. Winskel, eds., Lecture Notes in Comput. Sci. 1443, Springer-Verlag, Aalborg, Denmark, 1998, pp. 118–129.

[3] P. ELIAS, *Universal codeword sets and representations of the integers*, IEEE Trans. Inform. Theory, IT-21 (1975), pp. 194–203.

[4] H. GALPERIN AND A. WIGDERSON, *Succinct representations of graphs*, Inform. and Control, 56 (1983), pp. 183–198.

[5] X. He, M.-Y. Kao, and H.-I Lu, *Linear-time succinct encodings of planar graphs via canonical orderings*, SIAM J. Discrete Math., 12 (1999), pp. 317–325.
[6] A. Itai and M. Rodeh, *Representation of graphs*, Acta Inform., 17 (1982), pp. 215–219.
[7] G. Jacobson, *Space-efficient static trees and graphs*, in Proceedings of the 30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, NC, IEEE Computer Society Press, Los Alamitos, CA, 1989, pp. 549–554.
[8] S. Kannan, M. Naor, and S. Rudich, *Implicit representation of graphs*, SIAM J. Discrete Math., 5 (1992), pp. 596–603.
[9] M. Y. Kao, N. Occhiogrosso, and S. H. Teng, *Simple and efficient compression schemes for dense and complement graphs*, J. Combin. Optim., 2 (1999), pp. 351–359.
[10] K. Keeler and J. Westbrook, *Short encodings of planar graphs and maps*, Discrete Appl. Math., 58 (1995), pp. 239–252.
[11] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.
[12] G. L. Miller, *Finding small simple cycle separators for* 2*-connected planar graphs*, J. Comput. System Sci., 32 (1986), pp. 265–279.
[13] J. I. Munro and V. Raman, *Succinct representation of balanced parentheses, static trees and planar graphs*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 118–126.
[14] M. Naor, *Succinct representation of general unlabeled graphs*, Discrete Appl. Math., 28 (1990), pp. 303–307.
[15] C. H. Papadimitriou and M. Yannakakis, *A note on succinct representations of graphs*, Inform. and Control, 71 (1986), pp. 181–185.
[16] G. Turán, *On the succinct representation of graphs*, Discrete Appl. Math., 8 (1984), pp. 289–294.
[17] W. T. Tutte, *A census of planar triangulations*, Canad. J. Math., 14 (1962), pp. 21–38.
[18] W. T. Tutte, *A census of planar maps*, Canad. J. Math., 15 (1963), pp. 249–271.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 847–871

EFFICIENTLY CONSTRUCTING THE VISIBILITY GRAPH OF A SIMPLE POLYGON WITH OBSTACLES*

SANJIV KAPOOR† AND S. N. MAHESHWARI†

Abstract. This paper describes an output-sensitive scheme to construct the visibility graph of a simple polygon with m obstacles and n vertices in optimal $O(|E| + T + m \log n)$ time where $|E|$ is the size of the visibility graph and T is the time required to triangulate the simple polygon with obstacles. We use a partition of the space into regions called corridors which eases the efforts of the construction. Our algorithms are simple and the data structures used are only linked lists.

Key words. simple polygon, obstacles, visibility graph, triangulation, linked lists, output-sensitive

AMS subject classifications. 68Q25, 68U05

PII. S0097539795253591

1. Introduction.

The visibility graph, $GVIS = (V, E)$, of a simple polygon with m obstacles is a graph where the vertices in set V, $|V| = n$, are the vertices of the simple polygon with obstacles and the edge set, E, is defined as follows: $(u, v) \in E$ iff u and v can be connected to each other by a line segment that does not intersect any edge or vertex of the simple polygon or obstacle. Such an edge is referred to as a *visibility edge*. We refer to the simple polygon with obstacles as a *polygonal structure*. The visibility graph is a fundamental structure in computational geometry. It has a number of applications including being a graph in which the shortest distance between any pair of vertices in the polygonal structure can be found using standard graph algorithms [AAG+86]. While in recent developments algorithms for the shortest path problem use a continuous version of Dijkstra's algorithm, the visibility graph approach still provides for simple and efficient solutions [KM88, KMM97] which are optimal when the number of obstacles is $O(\sqrt{n})$.

Since the visibility graph has size $O(n^2)$, algorithms for this problem will require $O(n^2)$ time in the worst case. In fact, worst case time optimal algorithms are well known for this problem [AAG+86, Wel85, GM91]. Other related results may be found in [Her89, OW88]. In this paper we describe an algorithm which requires time $O(|E| + T + m \log n)$, where E is the edge set of the visibility graph and T is the time required for triangulating the simple polygon with obstacles. An $O(n \log n)$ worst case bound on T is well known [PS85]. However, when the number of obstacles is small improvements are possible. A bound of $O(n + m \log^{1+\epsilon} m)$ [BYC94] is known for the triangulation problem when there are $O(m)$ obstacles.

We note that a scheme requiring $O(|E| + n \log n)$ steps for solving this problem has also been discovered by [GM91]. The algorithm developed here appears simpler and uses less complicated data structures.

Our development is as follows: The algorithm first partitions the space into subregions which we call corridors. We describe this in section 2. The visibility graph of corridors is easy to determine in linear time. We show how to do this in section 3. We

*Received by the editors May 14, 1997; accepted for publication (in revised form) September 21, 1999; published electronically August 9, 2000. A version of these results appeared in [KM88].
http://www.siam.org/journals/sicomp/30-3/25359.html
†Department of Computer Science, Indian Institute of Technology, Hauz Khas, New Delhi 110016, India (skapoor@cse.iitd.ernet.in).

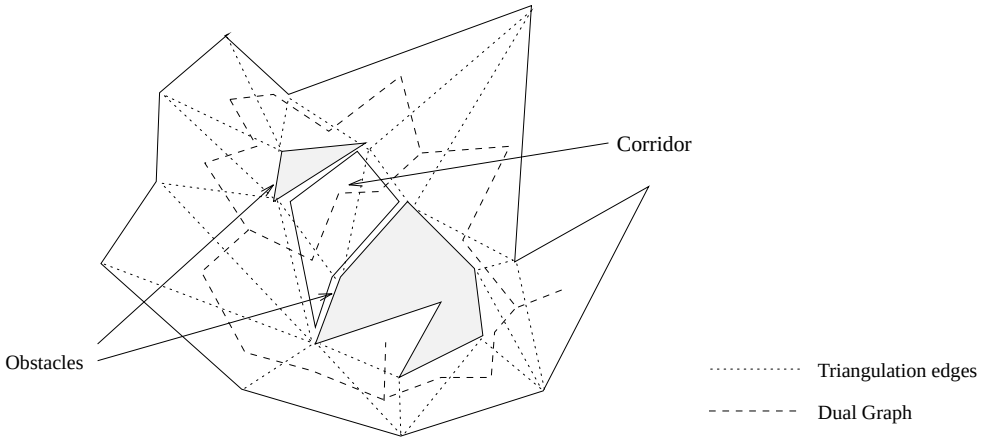


FIG. 1. *Partition.*

then show how to extend this construction to the case of a general polygonal structure in section 4.

2. Corridor structure. In this section we will consider the triangulation of the polygonal structure and extract from it regions which we call corridors. This partition of the space into corridors has been used before in a shortest path algorithm [KM88].

Consider an arbitrary triangulation of the simple polygon with obstacles.

The triangulation partitions the space into triangular regions. We consider the dual graph $G_d = (V_d, E_d)$ obtained by assigning vertices to the regions and edges between a pair of vertices if the two corresponding triangles share an edge. Let $v \in V_d$ be a vertex. The region corresponding to v is denoted by $T(v)$. Furthermore, let $e = (v_1, v_2)$ be an edge in the dual graph. The triangulation edge that separates the triangles, $T(v_1)$ and $T(v_2)$, is defined to be $D(e)$, the edge dual to e. $T(v_1)$ and $T(v_2)$ are termed adjacent triangles (Figure 1).

We next partition the dual graph into paths called C-paths. We define a 2-path in G_d to be a simple path in which all internal vertices, if they exist, are of degree 2. A C-path is a 2-path of maximal length. Each such C-path, P, defines a corridor, $\mathcal{C}$, as follows: $\mathcal{C}(P) = \cup_{a \in P} T(a)$, degree $(a) = 2$ or 1, i.e., the region formed by the union of the sequence of triangles corresponding to the vertices of degree 2 and degree 1, if they exist, of P. A corridor is a simple polygon.

We consider the general case, when the path P starts and ends at vertices of degree 3. The corridor $\mathcal{C}(P)$ has two sides comprising edges which are edges of the polygonal structure. There are also two bounding edges which form the corridor end. These edges are triangulation edges which separate the corridor from the other regions. A corridor may also simply be an edge as we see below.

When an endpoint of the path P has degree 1, the corresponding corridor end has a degenerate bounding edge, (u, v) with $u = v$. The last triangle in the corridor comprises two edges of the polygonal structure. And there is no adjacent corridor at this corridor end.

Corridors are connected to each other by triangles called junctions. These junctions correspond to vertices with degree 3 in the dual graph, G_d. A corridor may be connected either to other corridors through a junction at a corridor end or to none. These corridors are called adjacent corridors. The case when the corridor is connected

to exactly one other corridor through a junction occurs when both the bounding edges of the other corridor form two of the three edges of the junction triangle.

Note that a corridor may be degenerate. Consider when two degree 3 vertices u and v are adjacent. The edge $D(e)$, dual to $e = (u, v)$, is said to form a degenerate corridor. The corridor consists of a single (triangulation) edge alone which serves as the two bounding edges. The two vertices are degenerate sides of the corridor.

Given a triangulation, the partitioning of the polygonal region into corridors and junctions can be achieved in $O(n)$ time. This is done by identifying the degree 3 and degree 1 vertices in the dual graph. Each C-path which connects a degree 3 or degree 1 vertex to another degree 3 or degree 1 vertex defines a corridor.

2.1. Definitions. We need the following definitions.

We define the corridor distance, $dist(u, v)$, between two vertices u and v of the polygonal structure in a corridor, $\mathcal{C}(P)$, to be the minimum number of edges required to be traversed in $P \in G_d$ to reach a triangle containing u from another triangle containing v. More precisely, $dist(u, v)$ is the minimum number of edges required to be traversed in P to reach a vertex $b \in G_d$ from another vertex $a \in G_d$ such that $u \in T(a)$ and $v \in T(b)$.

Given points u, v, and w, we will let $\angle(u, v, w)$ represent the smaller angle (less than 180 degrees) formed at v by the line segments (v, u) and (v, w). An angle $\angle(u, v, w)$ is defined to be inside a corridor if the corridor contains a sector, $S(u, v, w)$, of a circle in the plane defined as follows: $S(u, v, w) = B(v, \epsilon) \cap Cone(u, v, w)$ where ϵ is a sufficiently small constant, $B(v, \epsilon)$ is the circle of radius ϵ centered at v, and $Cone(u, v, w)$ is the cone in the plane with vertex v and angle $\angle(u, v, w)$ with the sides defined by the lines containing the line segments (v, u) and (v, w).

Given a list L and elements (pointers) a and b of (into) L, suppose a precedes b in a list L. This will be indicated by $a \preceq_L b$. And if a succeeds b in the list, L, then we shall represent this by $a \succeq_L b$. Use of L will be eliminated wherever understood.

3. Constructing the visibility diagram of a corridor. In this section we show how to construct the visibility diagram for the set of vertices in a corridor. We consider only nondegenerate corridors.

Let P be a C-path in G_d. Let $\mathcal{C}(P)$ be the corresponding corridor with two bounding edges $U = (a, b)$ and $B = (p, q)$ and two sides, (i) S_1, which connects a to p using edges of the polygonal structure, and (ii) S_2, which connects b to q. We let the sequence of degree 1 and 2 vertices of the path P be $p_1, p_2, \ldots, p_k$ and let the triangles, corresponding to the vertices on the path, be $T_1, T_2, \ldots, T_k$. Note that $T_i = T(p_i)$.

To construct the visibility diagram for the corridor, we traverse the corridor from one boundary, say, $U = (a, b), a \in S_1, b \in S_2$, called the upper boundary, to the other boundary, B, the bottom boundary, i.e., the vertices of the corridor are processed in order of increasing corridor distance from the vertices on the bounding edge, U. The basic attempt is to construct a list for the current vertex, v, that contains vertices visible to v from among the vertices already considered in the corridor. This suffices to construct all the visibility edges since each visibility edge, (u, v), v being at greater corridor distance than u from the boundary U, is constructed once when v is processed. Two lists are constructed for each vertex v, one list for each of the two sides of the corridor. Each such list is called a visibility list. The lists are sorted in the order that the visible vertices occur along the sides of the corridor, i.e., by decreasing value of the distance $dist(u, w)$, $u \in \mathcal{C}, w \in U$, or, equivalently, by increasing value of $dist(v, u), u \in C$.

For a vertex, v, the visibility list comprising vertices visible to v on side S_1 is $LIST_1(v)$ and that comprising vertices visible to v on side S_2 is $LIST_2(v)$. We let $T_1 = (a, b, v_1)$. v_1 is the first vertex to be considered in the corridor. $LIST_1(v_1)$ contains a and $LIST_2(v_1)$ contains b. The algorithm next considers another vertex obtained from the triangle T_2 adjacent to T_1 in the corridor C. T_2 adds one new vertex. This vertex is adjacent to v_1 via a triangulation or a corridor edge. The visibility list of the new vertex is constructed and this process is repeated for successive vertices. We describe the general step next.

Let v_i be the vertex and T_i the triangle to be considered at the ith step. v_i is contained in triangle T_i which also contains v_{i-1}, the last vertex considered. Assume without loss of generality (w.l.o.g.) that v_i is on side S_1. Let $u_1 \dots u_k = v_i$ be the vertices on side S_1 already considered by the algorithm and let $w_1 \dots w_l$ be the vertices on side S_2 already considered up to this step.

First we describe how to build $LIST_1(v_i)$. The first vertex visible to v_i is u_{k-1} on side S_1. From u_{k-1} we will obtain the next visible vertex, say, u_j. Considering vertices along the corridor would involve traversing too many useless (invisible) vertices. We thus use the vertices on the visibility lists of u_{k-1} itself, taking care not to traverse this list more than a constant number of times.

We find u_j by traversing $LIST_1(u_{k-1})$ to find the highest indexed vertex v such that $\angle(v, u_{k-1}, v_i)$ is an angle inside the corridor, if it exists. This vertex is the vertex in $\{u_1 \dots u_{k-2}\}$ closest to v_i on side S_1, which may be visible to v_i. Note that if no such angle is found, then no visible vertex u_j exists. The vertex satisfying the above angle property is a candidate for addition to the visibility list of v_i.

Let this candidate be called v_q. Suppose v_q is visible. The next vertex on $LIST_1(v_i)$ is obtained from v_q. Again we find the highest indexed vertex in $LIST_1(v_q)$, v, such that the $\angle(v, v_q, v_i)$ is an angle inside the corridor and we determine its visibility. This is repeated until we find a candidate which is not visible or the end of the corridor is reached.

To determine if the candidate v_q is visible or not, note that the vertex will be invisible only if (v_i, v_q) intersects an edge on S_2. At least one such edge, say, e_t, will be visible from v_i and in fact the vertex of the intersected edge furthest from v_i, say, v_t, satisfies $dist(v_t, v_i) < dist(v_q, v_i)$. This is because the sequence of triangles in the corridor which starts from a triangle containing v_i and ends at a triangle containing v_q includes the triangle containing e_t. Note that in fact the edge which obstructs the visibility has an endpoint which either is in $LIST_2(v_i)$ or is a candidate for addition to $LIST_2(v_i)$.

In the algorithm below, $LIST_1(v_i)$ and $LIST_2(v_i)$ are built simultaneously. At each step we maintain two candidates to be added, one to each list. When both candidates are present then the candidate closer to u_k is added to the appropriate list. If only one candidate is present, then it is added to the appropriate list provided it is visible. The next candidate for the updated list is computed from the visibility list of the candidate added. This process is repeated either until the end of the corridor is reached or until no candidate is available.

We can now detail our algorithm. It is assumed that v_i is on side S_1.

ALGORITHM CORRVIS(C).

{ Corridor C has a boundary $(a = v_0, b = v_{-1})$ and comprises triangle $T_1 \dots T_k$. }

$i \leftarrow 0$;

Repeat

$i \leftarrow i + 1$;

```
Pick vertex v_i, adjacent to v_{i-1}, in triangle T_i = (v_{i-1}, v_j, v_i), j ≤ i − 2;
If v_{i-1} is on side S_1 then
        begin
        LIST_1(v_i) ← v_{i-1}; can_2 ← v_{i-2}
        end
else
        begin
        LIST_1(v_i) ← v_{i-2}; can_2 ← v_{i-1}
        end;
Repeat
        vis ←last vertex added to visibility lists of v_i;
        If vis is in LIST_1(v_i) then
                can_1 ← CANDIDATE(v_i, vis)
        else can_2 ← CANDIDATE(v_i, vis);
        If Both can_1 and can_2 exist then
                If dist(can_1, v_i) ≤ (can_2, v_i) then
                   Add can_1 to LIST_1(v_i)
                else Add can_2 to LIST_2(v_i)
        else    If can_1 exists and is visible to v_i then
                   Add can_1 to LIST_1(v_i)
                else If can_2 exists and is visible to v_i then
                        Add can_2 to LIST_2(v_i);
        Until (no more candidates are added to the lists);
Until (no more triangles left);
```

Next we describe two procedures used in the above algorithm. The first procedure determines if a candidate *can* is visible to vertex v_i. The second procedure, $CANDIDATE(v_i, vis)$, generates the candidate following vis which may be added to the visibility list of v_i that contains vis. The procedure may not return with a candidate if vis is the last vertex in the visibility list of v_i.

3.1. Determining visibility. Let can_1 and can_2 be the two candidates at time step t. Let $u_1, u_2 \ldots u_l$ be the vertices in $LIST_1(v_i)$ and let $w_1, w_2 \ldots w_r$ be the vertices in $LIST_2(v_i)$. Assume that there is a coordinate frame centered at v_i. Let α_j be the angle that (v_i, u_j) makes with the x-axis, and let β_j be the angle that (v_i, w_j) makes with the x-axis. We assume w.l.o.g. that $LIST_1(v_i)$ is being constructed in a clockwise fashion and $LIST_2(v_i)$ in a counterclockwise fashion. $LIST_1$ and $LIST_2$ are defined to overlap if $\exists p, q, 1 \le p, q \le r$ and $\exists j, k, 1 \le j, k \le l$ such that $[\beta_p, \beta_q] \cap [\alpha_j, \alpha_k] \neq \emptyset$. If such an overlap occurs, then one of the vertices in (u_p, u_q, w_j, w_k), say, u_q, is not visible from v_i since it is blocked by an edge on side S_2.

To determine visibility the following cases arise:

(1) Both can_1 and can_2 exist. Then the closer of the two vertices is visible to v_i.
(2) Suppose can_1 exists but can_2 does not. Let w' be the candidate generated for the visibility list, $LIST_2(v_i)$, when the last vertex w_r was added to it. w' may not have existed. In this case it suffices to check if $LIST_1(v_i)$, with the candidate can_1 added to it, overlaps with $LIST_2(v_i)$. If it does, then can_1 is not visible. If w' existed, then can_1 is not visible and will not exist since w' is not visible (see proof of Lemma 3.1 below).
(3) Suppose can_2 exists and can_1 does not. This case is symmetric to the previous case.

Note that candidates are added to the visibility lists for v_i in order of increas-

ing distance from v_i. We formally prove the correctness of the above procedure for computing the visibility.

LEMMA 3.1. *The candidate visible to v_i, from among can_1 and can_2, is correctly computed.*

Proof. We show the correctness of the procedure in the three cases that arise when determining can_1 and can_2.

Consider the first case. In this case since both can_1 and can_2 exist the candidate closer to v_i, say, can_1, is visible. The proof is by contradiction: Suppose not. Then can_1 is obstructed due to the overlap of $LIST_1(v_i)$ appended with can_1 and $LIST_2(v_i)$. But then no candidate at greater corridor distance than can_1 will be visible from the last vertex, t, in $LIST_2(v_i)$. Since can_2 is generated from $LIST_2(t)$ and must be visible from t, this is a contradiction.

In the second case, if w' does not exist, then there is no vertex, w', on side S_2 such that w' is visible to w_r and $\angle(v_i, w_r, w')$ is an angle less than 180 degrees inside the corridor. Thus only vertices in $LIST_2(v_i)$ can obstruct the visibility of the candidate, can_1, from v_i. This is determined correctly.

Alternatively suppose w' exists. Then since it is not a candidate currently, it has already been determined that w' is not visible due to an obstruction by an obstacle edge on side S_1. This implies that any vertex, v, such that $dist(v, v_i) > dist(w', v_i)$ and $\angle(v, u_j, v_i)$ is an angle inside the corridor, u_j being the last vertex in the current list $LIST_1(v_i)$, is not visible to v_i and to u_j. As candidates are considered in order of increasing distance from v_i, $dist(can_1, v_i) > dist(w', v_i)$ and thus can_1 is not visible to v_i and cannot exist.

The third case is symmetric to the first. □

3.2. Next candidate computation. To find the next vertex which is a candidate for the visibility list of v after a vertex vis has been added, we employ a scheme which generates the next candidate by a simple scan of the visibility list at vis. To do so we associate a constant number of pointers with each of the visibility lists of every vertex whose visibility lists have been built up. We next show that the lists will be scanned once only by a pointer, throughout the construction, thus ensuring efficient generation of candidates.

We need the following property and definitions: We let $U = (u_1, u_2)$ be the upper bounding edge of a corridor C.

DEFINITION. *A vertex a* occurs before *a vertex b on side, S, of corridor, C, with respect to bounding edge U, if $dist(a, u_1) < dist(b, u_1)$.*

Henceforth, we assume that the scan direction and starting boundary are fixed. Note that the visibility lists at a vertex, v, are ordered so that a vertex y follows x in the list if $dist(x, v) \leq dist(y, v)$.

LEMMA 3.2 (property monotonicity). *Let a and b be vertices of corridor C such that a occurs before b. Let u be a vertex visible to both a and b such that $dist(u, u_1) < dist(a, u_1) \leq dist(b, u_1)$. Let P_a and P_b be the pointer positions in $LIST_1(u)$ (or in $LIST_2(u)$ as appropriate) which give the next candidates for addition to the visibility list $LIST_1$ (or $LIST_2$) of a and b. If a and b are on side $S_1(S_2)$, then*

- *$P_a \succeq P_b$ ($P_a \preceq P_b$) in $LIST_1(u)$ if u is on S_1, and*
- *$P_a \preceq P_b$ ($P_a \succeq P_b$) in $LIST_2(u)$ if u is on S_2.*

Proof. We assume that a, b, and u are on S_1. The next candidate vertex for addition to $LIST_1$ of a or b is obtained by the first vertex, v_a or v_b, visible to u such that the angle $\angle(v_a, u, a)$ or $\angle(v_b, u, b)$ is an angle inside the corridor. Consider the straight line rays (a, u) and (b, u) with origin a and b, respectively. Let the rays strike

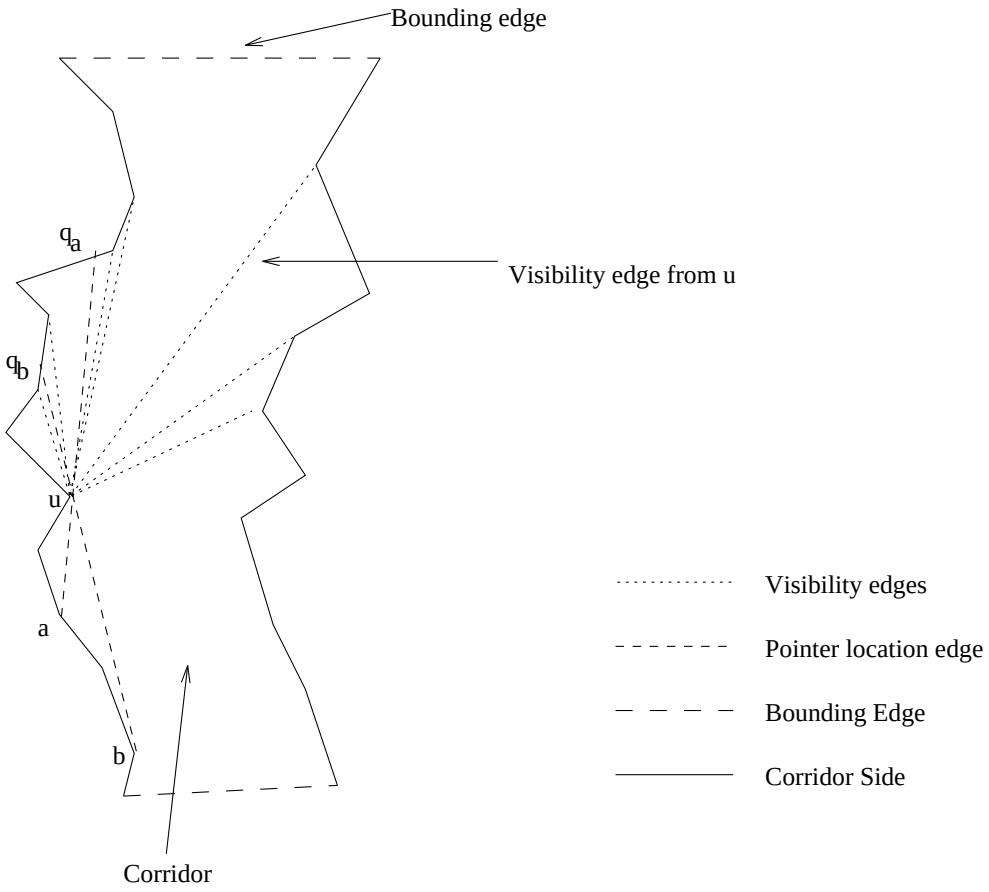


FIG. 2. *Order in a corridor.*

the side S_1 at q_a and q_b. The next candidate, $v_a(v_b)$, for $a(b)$ is the first vertex in $LIST_1(u)$ after the point $q_a(q_b)$ on S_1. Since q_b is closer to u than q_a the property follows. A symmetric proof applies in the other cases. (See Figure 2.) □

We use this property as follows: There are two pointers associated with a list, say, $LIST_1(u)$, at a vertex u on side S_1 of a corridor C. The first pointer, p_1, generates candidates for vertices on side S_1 whereas the second one, p_2, generates candidates for vertices on side S_2. Initially, the pointer p_1, associated with a list of u, is at one end of the list. Note again that the list is ordered by increasing distance of the visible vertices from u. The pointer, p_1, associated with the list at vertex u points to the candidate which is generated for $LIST_1(v)$, using the list $LIST_1(u)$, when u is added to the visibility list of v. The next time u is added to the visibility list of some other vertex, say, w, further along the corridor and on the same side, S_1, the pointer moves from its current position in $LIST_1(u)$ to generate the candidate for w. Since vertices are considered in order of their increasing distance from the bounding edge, for construction of the visibility lists, the monotonicity property of the position of the pointers (Lemma 3.2) ensures that the direction of movement of the pointer, p_1, is monotone and thus the list is scanned once only by the pointer, p_1, to generate candidates for all the vertices on side S_1 in the corridor. The same property is true

for p_2 when generating candidates for vertices in S_2 using $LIST_1(u)$, except that the list is traversed by this pointer in a direction opposite to that of p_1. Thus we have the following.

LEMMA 3.3. *Let $v \in C$. The visibility lists $LIST_1(v)$ and $LIST_2(v)$ are traversed only once to generate candidates for vertices in C.*

3.3. Correctness and time complexity. We show the following lemma.

LEMMA 3.4. *The visibility graph, $GVIS(V, E)$, where V is the set of vertices of a triangulated corridor, is correctly generated in $O(|E|)$ time.*

Proof. We first show correctness of the algorithm, CorrVis(C). It suffices to show that for a given vertex v_i, the visibility lists are correctly computed given the visibility lists of vertices which occur before v_i in the corridor. We show that $LIST_1(v_i)$ is correctly computed. The construction of $LIST_2(v_i)$ is symmetric and the correctness follows by a similar argument.

Assume that v_i is not on the boundary of the corridor. If it is, then the visibility list is the null list which is trivially constructed. To prove that $LIST_1(v_i)$ is correctly computed, we use induction on the distance $dist(v_i, vis)$ where $vis \in LIST_1(v_i)$ or $LIST_2(v_i)$. For the base case note that the vertex adjacent to v_i at distance 1 is trivially in $LIST_1(v_i)$. Suppose all visible vertices at distance $\leq k$ have been added to $LIST_1(v_i) \cup LIST_2(v_i)$. Consider the closest vertex, v, visible to v_i at distance greater than k from v_i. Assume w.l.o.g. that $v \in LIST_1(v_i)$. The vertex, v, is in the visibility list of vertex v' such that $dist(v_i, v') \leq k$ and v' is the vertex prior to v in $LIST_1(v_i)$. To prove this, consider the triangle $T = (v, v_i, v')$. If v is not visible to v', then (v', v) is intersected by an edge e of side S_1 but no edge of S_1 intersects (v_i, v) and (v_i, v'). Thus $\exists$ a vertex v'' on side S_1 such that $dist(v_i, v'') > dist(v_i, v')$, which is inside T and visible to v_i. This contradicts the choice of v'. Thus v will be generated as a candidate when v' is added to $LIST_1(v_i)$. Moreover, the visibility of the candidate is correctly determined as Lemma 3.1 proves. This completes the induction step.

To prove the time complexity note that, by Lemma 3.3, each visibility list is traversed once only while generating candidates for visibility lists of other vertices. Moreover, visibility of a candidate is checked in constant time since the last vertices in the two lists together with the candidate suffice to determine when an overlap occurs. Visibility list construction thus requires time $O(|\cup_{v \in C}(LIST_1(v) \cup LIST_2(v))|) = O(|E|)$. □

4. Visibility in a simple polygon with obstacles. To construct the visibility diagram of a simple polygon among obstacles we investigate the properties of the visibility lists of a point v in the polygonal structure. We need the following definitions.

Let $SC = C_1, C_2 \ldots C_k$ be a sequence of corridors such that (s.t.) C_{i+1} is adjacent to C_i via junction J_i, i.e., both corridors have a bounding edge in the triangle corresponding to the junction J_i, $\forall i, 1 \leq i \leq k-1$. Let $V_{SC} = \{v | v \in G_d \ \& \ T(v) \in [\cup_i C_i] \cup \mathcal{J}\}$ where $\mathcal{J} = \{J_1, J_2, \ldots, J_{k-1}\}$ and $E_{SC} \subset E$ is the set of edges induced by V_{SC}. The vertices and edges in V_{SC} and E_{SC} form a path $P_{SC} \in G_d$.

Similar to the previous section the corridor distance between two vertices v_1 and v_2, $dist(v_1, v_2)$, along a sequence of corridors SC is the minimum number of edges required to be traversed on the path, P_{SC}, to reach a triangle containing v_1 from a triangle containing v_2.

We assume a rectangular coordinate system $\mathcal{L}$ on the plane.

In the visibility diagram of a simple polygon with obstacles since the visible vertices may belong to different corridors there is no induced ordering on the vertices

themselves. However, the set of vertices visible to v can be partitioned such that with each partition is associated a sequence of corridors. In this section we show that the visible vertices can be arranged in a tree, called the visibility tree. We then show how to construct these trees for all the vertices in the polygonal structure in time proportional to the number of visibility edges.

4.1. Visibility trees. Let v be a vertex in a corridor C. Let $U = (a, b)$ and $B = (p, q)$ be the two bounding edges of the corridor. And let $VIS_B(v)$ be the set of vertices visible to v through edge B, where a vertex is defined to be visible through edge B if the visibility edge $(v, v'), v' \in VIS_B(v)$, intersects B. Suppose each visibility edge is labeled by the sequence of corridors which it traverses to reach vertex v. This labeling partitions the set of visibility edges in $VIS_B(v)$, with edges that traverse the same sequence of corridors being in one partition. Note that all the visibility edges in a partition are in the same corridor. The edges in a partition can be ordered and will be shown in fact to be constructed by a procedure similar to that in the previous section. We represent the partition of the set of visibility edges in $VIS_B(v)$ by a tree called a visibility tree and termed $TVIS_B(v)$. We will describe the construction of this tree below. All vertices visible to v can be divided into two sets, $VIS_U(v)$ and $VIS_B(v)$, and for each set a visibility tree will be constructed.

We now show how to construct the tree, $TVIS_B(v)$. At the root, r, of $TVIS_B(v)$ are two lists $LIST_1$ and $LIST_2$ which store, in order, vertices u of C on the two sides S_1 and S_2 that are visible to v and with corridor distance to v, $dist(u, v)$, less than $dist(p, v)$ and $dist(q, v)$, respectively. These vertices are termed to lie in between v and B. The root has at most two sons. With each son is associated a corridor which contains vertices visible to v through B. The two sons are ordered from left to right in increasing order of the angle of visibility edges from v to visible vertices in each of the two corridors. The angle is measured in, say, the counterclockwise direction in a coordinate system parallel to $\mathcal{L}$ and centered at vertex v. Assume w.l.o.g. that the angle between (p, v) and (q, v) is less than π in the counterclockwise direction. The angle of a visibility edge (v, p') is measured relative to (v, p), i.e., by $\angle(p, v, p')$. The ordering obtained is called an angle ordering. A similar ordering is defined for $TVIS_U(v)$. In this tree the sons are ordered again by increasing angle order. The angles are measured in the counterclockwise direction in a coordinate system parallel to $\mathcal{L}$ and fixed at vertex v. The angle of a visibility edge (v, b') is measured relative to (v, b), i.e., by $\angle(b', v, b)$.

W.l.o.g. we restrict our attention to $TVIS_B(v)$. The corridors at the sons of the root of $TVIS_B(v)$ are chosen so as to contain the closest vertex visible to v through B. Note that these corridors are not necessarily adjacent to C via the junction triangle, T, containing edge B. Since a vertex may belong to one or two corridors, the node has at most two sons. Let C_1 and C_2 be the two corridors. Each of the sons has two lists which contain, in order, the vertices on each of the two sides of the corridor, say, C_1, that are visible to v such that the corresponding visibility edges cross triangles in C_1 and intersect B. The visibility edges to the vertices in the corridor at the left son precede the visibility edges to the vertices in the corridor at the right son. Note that the corridors at the two sons could be the same. However, the lists of visible vertices stored at the left son are disjoint from the lists at the right son. This relationship of sons of a node in the tree to their parent node, as illustrated for the root node, is repeated recursively.

In general, a node p in the tree $TVIS_B(v)$ is associated with a corridor C_p, a bounding edge $B(p)$ of C_p, and at most two lists, $LIST_1(C_p)$ and $LIST_2(C_p)$. Note

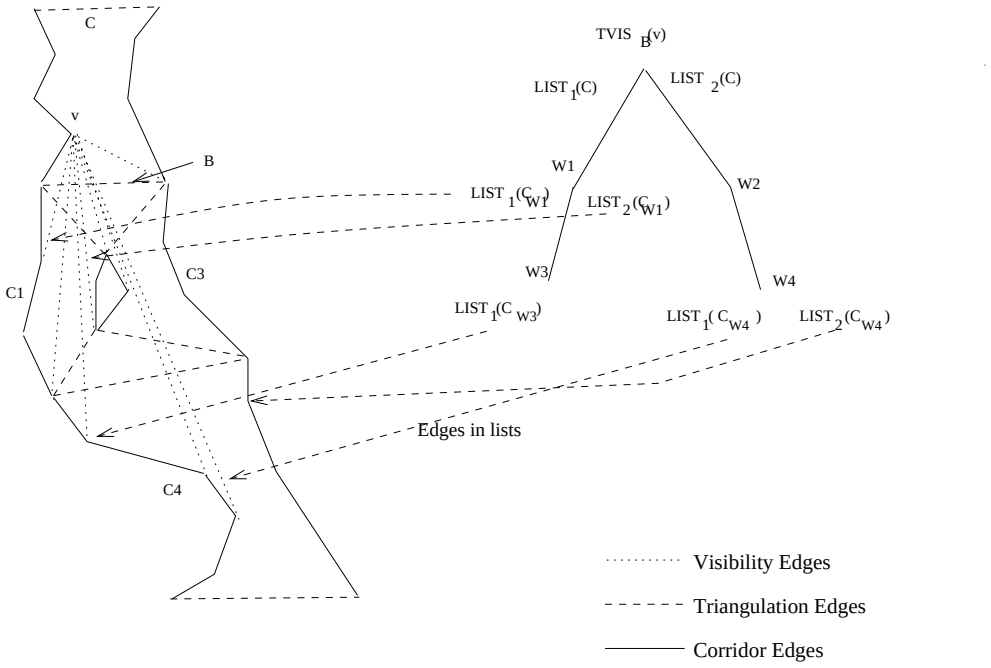


FIG. 3. *Visibility tree.*

that C_p may not be adjacent to the corridor, C', associated with the parent of p. In this case there is a unique sequence of corridors which connects C' to C_p. This is the sequence of corridors traversed by the visibility edge to the vertex closest to v in C_p. We term the sequence which connects C' with C_p as $AC(p)$. We let $SC(path(p))$ be the sequence of corridors, not including C_p, which are associated with or connect corridors corresponding to nodes on $path(p)$ from p to the root node of $TVIS_B(v)$, i.e., $SC(path(p)) = AC(p)C_{p_1}AC(p_1)C_{p_2}AC(p_2)\ldots AC(p_k)$, where p_k is the root of the tree $TVIS_B(v)$, and $p_1, p_2 \ldots$ are the nodes on the path from p to the root.

We consider vertices, w, in C_p that are visible to v such that the visibility edge, $(w, v), w \in C_p$, intersects $B(p)$ and the triangles in the sequence of corridors $SC(path(p))$. C_p has two sides. Assume w.l.o.g. that visible vertices on side S_1 occur before than visible vertices on side S_2, the vertices being ordered by angle ordering. $LIST_1(C_p)$ is the list of vertices on side S_1 of C_p, visible to v, such that the visibility edge, $(v_i, v), v_i \in LIST_1(C_p)$, intersects $B(p)$ and the triangles in the sequence of corridors $SC(path(p))$. Similarly, $LIST_2(C_p)$ is the list of nodes on side S_2 of the corridor C_p visible to v through $B(p)$ and the same sequence of corridors, $SC(path(p))$ (Figure 3). As in the previous section the vertices in the lists are ordered according to increasing corridor distance from v.

Moreover, let $son(p)$ be either one of the two possible sons of p. Associated with $son(p)$ is a corridor C_{son} and a bounding edge $B(son)$ which contains the closest visible vertex to v through $B(son)$ and the sequence $SC(path(son(p)))$. $son(p)$ contains $LIST_1(C_{son})$ and $LIST_2(C_{son})$.

The following lemma shows the usefulness of this tree.

LEMMA 4.1. *Let v be in corridor C. Let B be a bounding edge of C. At the root of $TVIS_B(v)$ are stored all vertices visible to v in corridor C and lying in between*

v and B. Every other vertex visible to v is stored at a node in $TVIS_B(v)$ s.t., at node $w \in TVIS_B(v)$, are stored all vertices visible to v having the property that the corresponding visibility edge intersects B and $B(w)$ and traverses the corridor sequence $SC(path(w))$.

Proof. The first part of the lemma follows from the definition of $TVIS_B(v)$. To prove the second part first note that a vertex stored in a list at a node in $TVIS_B(v)$ satisfies the required property by construction.

Second, we show that every visible vertex is stored at the correct node w in $TVIS_B(v)$. Consider a vertex, v_w, in a corridor C_w, visible to v through B.

We prove by induction on the distance of v_w from v that v_w is present in either $LIST_1$ or $LIST_2$ at a node w, with an associated bounding edge $B(w)$ and a corridor C_w, in $TVIS_B(v)$ s.t. this vertex is visible through a corridor sequence $CS = SC(path(w))$ and boundary $B(w)$. For the base case, the construction procedure adds the closest vertex from the same corridor, C. Assume that the hypothesis is true for vertices at distance at most k from v. Consider the vertex v_w in C_w which is at distance greater than k from v. Suppose $CS = C_1 \dots C_q$ is the sequence of corridors intersected by the edge (v, v_w) where C_1 is the first corridor traversed after the boundary B. W.l.o.g. assume that C_q, the last corridor in the sequence CS, has a vertex visible to v. By the induction hypothesis, a node q in $TVIS_B(v)$ corresponding to C_q exists.

First consider the case when v_w is the first vertex in corridor C_w that is visible to v through the sequence CS and $B(w)$. Since v_w is the closest vertex visible to v through the sequence CS and since q exists, the definition of $TVIS_B(v)$ ensures that a node corresponding to C_w will be present in the tree as a son of q and v_w will be contained in a list at that node. Since $CS = SC(path(w))$ the edge (v, v_w) has the required property that it intersects $B, B(w)$, and $SC(path(w))$. Note that if v_w is a vertex in a junction triangle, then it is part of two corridors and q will have two sons.

Alternatively, vertices from C_w already exist at the node w and v_w is added to a list at C_w in order of its distance from v according to the description of the visibility tree.

This completes the induction step. □

We can list all the vertices visible to v through edge B by enumerating the lists at the nodes of $TVIS_B(v)$. We describe an algorithm *L-Traversal* for doing so below. In the algorithm, r is the node in the tree being currently visited and r_l and r_r are the left and right sons, respectively.

ALGORITHM L-TRAVERSAL (r).

begin

Output $LIST_1(C_r)$;
L-Traversal(r_l);
L-Traversal(r_r);
Output $LIST_2(C_r)$ in reverse order;

end.

Here C_r is the corridor associated with node r. The procedure is called with the root of $TVIS_B(v)$ as parameter.

Note that if w is a node in $TVIS_B(v)$, then all vertices in the lists associated with nodes in the left subtree at w are output before the vertices in the lists associated with nodes in the right subtree at w. It is easy to see the following claim which shows an ordering of the visibility edges by angle.

LEMMA 4.2. *An L-traversal of $TVIS_B(v)$ lists, in increasing angle order, all the*

vertices visible to v through edge B of corridor C.

We note the following fact about $TVIS_B(v)$. A corridor may be associated with more than one node in the tree since sections of the same corridor may be visible to v but through different sequences of corridors (see Figure 3 where $C_{w3} = C_{w4} = C_4$) or through different boundary edges. However, each vertex appears only once in the tree.

To summarize, the sets of all visible vertices are stored in trees. For each vertex v, in corridor C with bounding edges U and B, are constructed two trees $TVIS_U(v)$ and $TVIS_B(v)$ which store vertices visible to v through U and B, respectively.

4.2. Constructing $TVIS_B(v)$. Next we describe how to construct $TVIS_B(v)$. We start with building the visibility lists at the root of $TVIS_B(v)$. Let C be the corridor containing v. After building the visibility list for C, one or two sons of the root node are created, each corresponding to a corridor containing the closest visible vertex to v through the bounding edge, B. Visibility lists are built for these corridors and the process is repeated with the addition of sons associated with available corridors.

In the next section we detail the construction of the visibility list for a corridor C_w at a node w in $TVIS_B(v)$ and subsequent generation of children nodes and the first vertex of the visibility lists at each child node. We outline the methodology: At each step we maintain at most two candidates from C_w, one each for addition to the two visibility lists, $LIST_1(C_w)$ and $LIST_2(C_w)$. A vertex p is added to $LIST_1(C_w)$ or to $LIST_2(C_w)$, as applicable, at the tree node w in $TVIS_B(v)$ as follows:

1. If both candidates exist, then p is the closest vertex, visible to v, among the two candidates from C_w.
2. If only one candidate exists, then p is that candidate provided it is visible.

As in the previous section, visibility is determined from the candidates and the current visibility lists, $LIST_1(C')$ and $LIST_2(C')$ where C' is either C_w or a corridor at the parent of w in $TVIS_B(v)$.

4.3. Obtaining candidates and constructing visibility edges. We detail the maintenance of two candidates for addition to the lists at the node w, corresponding to C_w, in $TVIS_B(v)$. The initial candidates at the root of $TVIS_B(v)$ are obtained from the vertices adjacent to v in the corridor, C. Suppose a vertex q is added to $LIST_1(C_w)$ from among the candidates. A new candidate needs to be obtained. This candidate is obtained from the visibility tree of q in corridor C_w. Let X be the bounding edge of the present corridor, C_w, containing q through which the visibility has to be further extended. Using a scan of $LIST_1(C_r)$, where $r = w$ is the root node of $TVIS_X(q)$, the first vertex l, referred to by can^*, which is a candidate for addition to $LIST_1$ as a vertex visible to v, is determined. Note that edges are ordered by their corridor distance. There are two cases: If l exists and is in the same corridor, then it forms the next candidate for $LIST_1(C_w)$. Alternatively, l may not exist in the same corridor.

Suppose the candidate does not exist in the same corridor. Then a new candidate is to be obtained from another corridor. However, the construction of the other list $LIST_2(C_w)$ may not have finished. A candidate is obtained for the construction of this list and added to $LIST_2(C_w)$ so that the resultant list does not overlap with $LIST_1(C_w)$. We assume that given a vertex q in corridor C, the two lists of vertices in C visible to q are present. This construction is performed as preprocessing for all the corridors using the algorithm in the previous section. When construction of both $LIST_1(C_w)$ and $LIST_2(C_w)$ finishes, new candidates from other corridors are to be obtained.

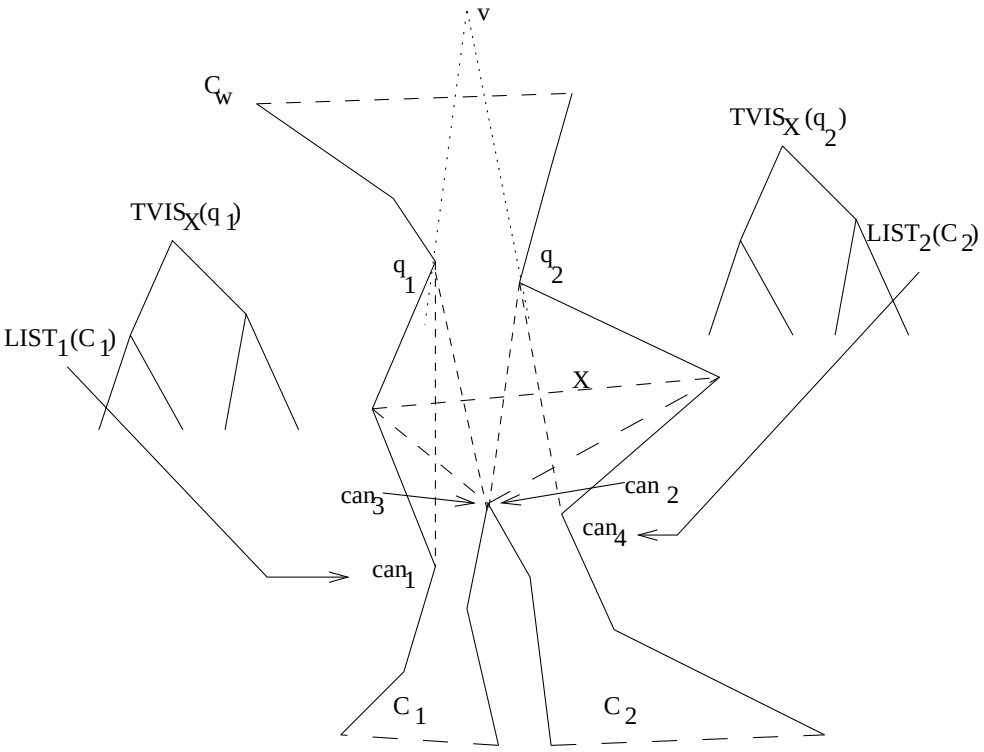


FIG. 4. *Candidate generation.*

Let q_1 and q_2 be the two vertices last added to $LIST_1(C_w)$ and $LIST_2(C_w)$, respectively. It may be that one of these lists, say, $LIST_2(C_w)$, is empty. In this case q_2 is chosen to be the vertex last added to a $LIST_2$ at the first ancestor node of w in the tree $TVIS_B(v)$, which contains a vertex in the list $LIST_2$ at that node. We let $C(q_1)$ and $C(q_2)$ be the corridors containing q_1 and q_2, respectively.

The following scheme, termed *Find-next*, is adopted to find candidates and extend the visibility tree in the adjacent corridors.

4.3.1. Description of procedure *Find-next*. Let t_1 be the node of $TVIS_{X1}(q_1)$ such that vertices visible through bounding edge X_1 are to be found from the corridors associated with the sons of that node. Initially, t_1 is the root node. And let t_2 be a node of $TVIS_{X2}(q_2)$ such that vertices visible through bounding edge X_2 are to be found from the corridors associated with the sons of that node. Again, initially t_2 is the root node. We assume w.l.o.g. that both the left and right sons of t_1 and t_2 exist. We generate two candidates can_1 and can_2 from $LIST_1(C_{left(t_1)})$ and $LIST_1(C_{right(t_1)})$. And two other candidates can_3 and can_4 come from $LIST_2(C_{left(t_2)})$ and $LIST_2(C_{right(t_2)})$. These candidates are obtained by finding the first vertex in the corresponding list that may be visible to v (Figure 4). The candidates may or may not exist. A candidate is visible if there exists a visibility edge which traverses, unobstructed, the sequence $SC(Path(w))$ as well as the corridor C_w. The visibility of can_1 and can_2 is determined by checking that each is not obstructed by the last candidate entered into $LIST_2(C(q_2))$ and the visibility of can_3 and can_4 is determined by checking that each is not obstructed by the last

candidate in $LIST_1(C(q_1))$. Such a visible candidate is termed r-visible. This check for visibility ignores that a candidate may be obstructed by some other candidate.

The following property is easy to see.

LEMMA 4.3. *The closest r-visible candidate from among can_1, can_2, can_3, and can_4 is visible from v.*

We next describe how to determine the closest r-visible candidate and extend the construction of the visibility tree. We need the following definition.

DEFINITION. $dist(C, v)$, C a corridor and v a vertex, is the corridor distance of the vertex in C closest to v.

Let c_m be the candidate closest to v among the visible candidates can_1, can_2, can_3, and can_4. c_m will be used as can^* to further construct the tree $TVIS_B(v)$. The following cases arise.

Case 1. c_m is can_1 in corridor C'. In this case add a tree node, w', corresponding to corridor C'. Next, candidates can be generated for $LIST_1(C')$ and $LIST_2(C')$. The first candidate for $LIST_2$ may be found from C'. If no candidate is available from C', then the first candidate for $LIST_2$ will be found from a descendant corridor. Note that a sibling of w' in the tree cannot exist. This follows by contradiction. Suppose it did. Then the vertex at the junction of the corridors, corresponding to the left and right sons, would be visible and thus can_1 would not be the closest candidate, c_m.

Case 2. c_m is can_2 in corridor C''. If this candidate vertex is a junction vertex, then two tree nodes at the two sons of the current tree node are constructed. Each node corresponds to a corridor containing can_2. Alternatively, one son of the current tree node is constructed. Note that in the second case can_1 does not exist. In the first case, let C' and C'' be the two corridors obtained. Candidates for $LIST_1$ and $LIST_2$ in both corridors are generated. This is done by using the left son of t_1 to obtain the first candidate available for $LIST_1(C')$ and by using the left son of t_2 to obtain the first candidate for $LIST_2(C')$. Similarly the lists at the right sons of t_1 and t_2 are used to obtain the first candidates for $LIST_1(C'')$ and $LIST_2(C'')$, respectively. These candidates are then used to generate other candidates at greater corridor distance. In the second case, the lists in the corridor containing can_2, say, C'', are generated similarly as above.

Case 3. The candidate is can_3. Again if the vertex is a junction vertex, then two sons are constructed. Otherwise, one son is constructed. Candidates for $LIST_1$ and $LIST_2$ in the corridors constructed are obtained as in the previous case.

Case 4. The candidate c_m is can_4. This case is symmetric to the first case and one tree node is added as a son of the current tree node.

Finally, consider the case when candidates can_1 and can_2 either do not exist or are not r-visible. Then we show how to determine a node in $TVIS_{X1}(q_1)$ such that visible vertices are to be found from the corridors associated with the sons of that node. This node is used to update t_1.

We have three cases (Figure 5).

Case (i). If can_1 does not exist and can_2 is not r-visible due to overlap with $LIST_2(C(q_2))$, then t_1 is updated to be the left son of t_1 in $TVIS_{X1}(q_1)$ or its descendants. This is because no vertex can be visible through the corridor associated with the right son since can_2 itself is obstructed by a vertex in $LIST_2(C(q_2))$.

Case (ii). If can_2 also does not exist, then t_1 is updated to become the right son since no vertex can become a candidate from the corridor associated with the left son of t_1 or its descendants.

Case (iii). If can_1 exists and is not visible again due to overlap with $LIST_2(C(q_2))$,

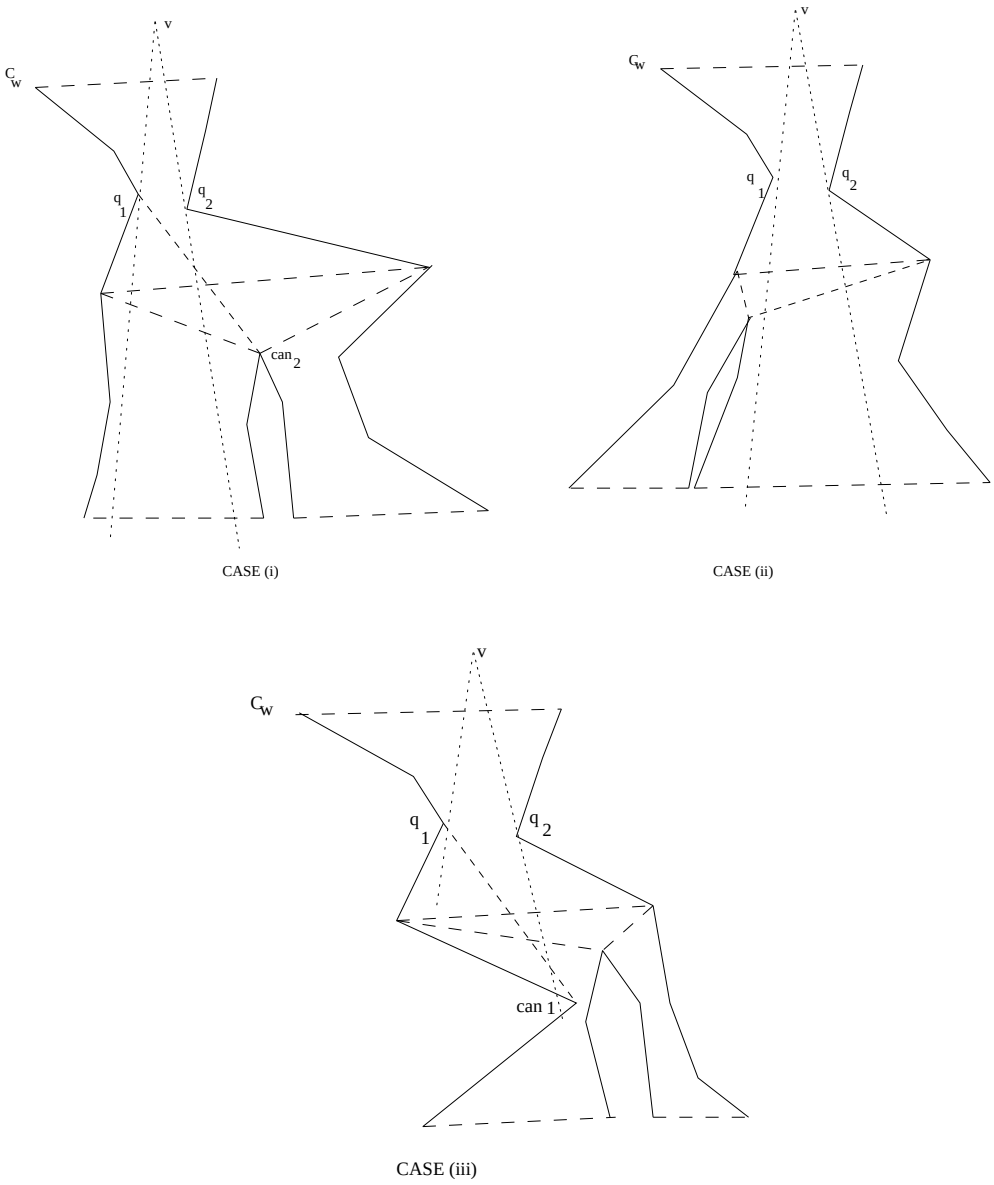


FIG. 5.

then processing need not continue.

A similar procedure is used to update t_2 if both can_3 and can_4 either do not exist or are not r-visible. We call the entire procedure described above *Find-new*.

We obtain the following lemma.

LEMMA 4.4. *Given a pair of nodes, t_1 and t_2, in two visibility trees, $TVIS_{X1}(q_1)$ and $TVIS_{X2}(q_2)$, respectively, the procedure Find-new correctly does one of the following:*

(1) *Finds the closest visible candidate, can^*, from the corridors at the sons of t_1 and t_2 and generates sons at node w of $TVIS_B(v)$;*

(2) *updates t_1 and t_2 in the two trees from among the sons of t_1 and t_2 such that the visible vertex closest to v is in a corridor at a descendant node of the updated t_1 and t_2;*

(3) *determines that no visible vertex exists at a descendant node of t_1 or t_2.*

Find-new is applied repeatedly to obtain candidates or to determine if the processing stops.

This describes procedure *Find-next*. Thus procedure *Find-next* finds the closest visible candidate from the corridors at descendant nodes of t_1 and t_2 and generates sons of w in $TVIS_B(v)$ or determines that no visible vertex exists at a descendant node of t_1 or t_2.

Furthermore, note the following. Suppose that can_1 exists. Then can_1 as well as can_2, if can_2 exists, can be found in time proportional to the time required to find can_1 in $LIST_1(left(t_1))$ or at descendant nodes. This is so because in this case can_2 will be a vertex of a junction triangle. Similarly, if can_4 exists, the time required to find can_3 and can_4 is proportional to the time required to find can_4.

For an efficient algorithm we will not implement procedure *Find-next* as above. We will detail an efficient procedure to determine the closest visible candidates later in the section. The algorithm will use linked lists and pointers to detect the closest visible vertex.

Finally, note that at a vertex the algorithm may not be able to obtain a candidate for its visibility list since the candidates may not be currently present. The availability of the visibility lists depends on the sequence in which vertices are considered by the overall algorithm for building visibility lists. However, if all the candidates required were available, the following lemma shows that the lists are correctly built.

LEMMA 4.5. *Suppose vertices in visibility lists are available when required for computation of candidates. Then $TVIS_B(v)$ is correctly computed.*

Proof. The proof is by induction on the size of $TVIS_B(v)$. The base case, when $TVIS_B(v)$ has one node, is trivially true since the vertices adjacent to v in C, of which there is at least one, are in the visibility list at the root of $TVIS_B(v)$. Assume that the tree is correctly built when there are k vertices in the visibility tree of v. Let w be the tree node at which the vertex prior to the $(k+1)$st vertex, in L-Traversal order, exists. Consider additions to the visibility lists, $LIST_1(C_w)$ and $LIST_2(C_w)$. If the $(k+1)$st candidate is in the same corridor, then by a proof similar to the proof of Lemma 3.1 addition to the visibility lists is correctly done.

Thus consider the case when there are no candidates in corridor C_w. Note that $LIST_1(C_w)$ or $LIST_2(C_w)$ may be empty. Let v_1 be the last vertex added to $LIST_1$ at w or an ancestor node, if $LIST_1(C_w)$ is empty. And let v_2 be the last vertex added to $LIST_2$ at w or at an ancestor node. Candidates are to be obtained from the corridors at the descendants of the root node in $TVIS_{B'}(v_1)$ and $TVIS_{B''}(v_2)$, B' and B'' being bounding edges. We let t_1 and t_2 be the root nodes of $TVIS_{B'}(v_1)$ and $TVIS_{B''}(v_2)$, respectively. Let v_c be the closest vertex visible to v through C_w. We prove by induction on the corridor distance between v_c and C_w that this candidate and the corridors at the sons of the node corresponding to w will be correctly added. For the base case, consider the case when the vertex v_c is at distance 1 from C_w, i.e., $dist(C_w, v_c) = 1$. The vertex v_c must be either in $LIST_1(y)$, where y is one of $left(t_1)$ or $right(t_1)$, or in $LIST_2(z)$, where z is one of $left(t_2)$ or $right(t_2)$. Let C_{lw} and C_{rw} be the corridors adjacent to C_w. If the closest vertex is in the junction between the three corridors, then two sons are added to the node w, one corresponding to C_{lw} and the other to C_{rw}. If the closest vertex is not the vertex in the junction, then one node

is added since only one adjacent corridor may have visible vertices. If both do, then the vertex at the junction will also be visible. This is determined in the procedure above as shown in Lemma 4.4

For the induction step, suppose that $dist(C_w, v_c) = j$ and further assume that if $dist(C_w, v_c) < j$, then the vertex and the corridors at the sons of w have been correctly determined and v_c obtained. We consider the corridors C_{lw} and C_{rw} adjacent to C_w above. Suppose v_c is in one of these corridors or in the closest corridor from among the set $\mathcal{C} = \{C_{left(t_1)}, C_{right(t_1)}, C_{left(t_2)}, C_{right(t_2)}\}$. Then, by Lemma 4.4, we are done since the son corresponding to the corridor will be added when $c_m = v_c$, the closest visible vertex in the corridors at the sons of t_1 and t_2, is determined. Otherwise, we need to determine the corridor through which the closest vertex is visible. This is achieved in the above procedure, *Find-next*, by determining a corridor, say, $C' \in \mathcal{C}$, which does not contain v_c but through which vertices are visible. The correctness of the determination of C' follows from Lemma 4.4. Candidates are generated from the sons of the tree node which is in $TVIS_{B'}(v_1)$ and $TVIS_{B''}(v_2)$ and corresponds to the corridor C'. Since $dist(C', v_c) < j$ we can, by induction, conclude that the procedure *Find-next*, when applied recursively, correctly determines the vertex, v_c, and the corridor or corridors which are to be associated with the son or sons, respectively, of the tree node $w \in TVIS_B(v)$.

This also completes the induction step to show that $TVIS_B(v)$ is correctly generated. □

Each vertex added to $TVIS_B(v)$ uses the visibility tree at v, $TVIS_U(v)$, where U is the other bounding edge of the corridor containing v, to further construct its visibility tree. The vertices from $TVIS_B(v)$ are considered in order of their occurrence in the L-Traversal of $TVIS_B(v)$ for construction of visibility edges using v. This is done to ensure efficiency.

We next consider the possibility that candidates are not available. Let us consider the case when a candidate for the visibility tree of a vertex v is to be obtained from $LIST_1(C)$ at a node associated with corridor C in $TVIS_X(q)$, the visibility tree of a vertex q. If the candidate is not present in the list, then v is put in a list associated with the vertex q, $WAITLIST_1(q)$. This list contains the vertices whose candidates have to be obtained from $LIST_1(C)$ for some C, in $TVIS_X(q)$.

When vertices are added to the visibility list of q, a candidate for each vertex in $WAITLIST_1(q)$ is generated if possible.

A similar situation arises when a candidate for v is to be obtained from $LIST_2(C)$ for some C in $TVIS_X(q)$. $WAITLIST_2(q)$ is a list which contains v when the candidate for v is not present.

We note the following property of the visibility lists.

LEMMA 4.6 (property sort). *Let v_1 and v_2 be two vertices added to $LIST_1(C_1)(LIST_2(C_1))$ for a corridor C_1 in tree $TVIS_B(v)$. Then v_1 and v_2 are added to the visibility list in the same order as they occur in the visibility list, $LIST_1(C_1)(LIST_2(C_1))$, i.e., in order of increasing corridor distance from v.*

Proof. We use induction on the maximum corridor distance between v and the vertices to be added. The basis of induction is easily established since when the corridor distance is 1 then there is only one vertex to be considered. Assume that v_1 is closer to v than v_2 is. And let the corridor distance between v and v_2 be k or greater. Next suppose that the claim is true for distances less than or equal to $k-1$. We show that v_1 and v_2 become candidates for the visibility list of v in that order.

Suppose v_2 occurs immediately after v_1 in the visibility list of v. Then v_2 will

be generated as a candidate *after* v_1 has been added to the visibility list of v since procedure *Find-next* determines the closest candidate for a list at every step. Alternatively, suppose there is some other vertex, say, $w \in C_1$, s.t. $dist(v_1, v) < dist(w, v) < dist(v_2, v)$. If more than one vertex satisfies this property then let w be the vertex closest to v_2. By induction, w occurs as a candidate for the visibility list of v after v_1 and since v_2 will be obtained from w, v_2 becomes a candidate after v_1. □

Using this property we next show that the visibility trees can be built in time proportional to the sizes of all the trees. There are two kinds of pointers associated with each vertex q. One kind is used to construct the tree and the other kind is used to generate candidates.

The movements of the first kind of pointer require $O(E)$ time, given the candidates, since by **Property Sort**, vertices are added in order for each corridor. We next consider the complexity of generating candidates.

4.4. Generating r-visible candidates. There are two cases in the generation of candidates. In one case, the candidates are generated from already constructed lists. In the second case, candidates are generated for vertices which are in some $WAITLIST$. As the vertices are added to the visibility list of a vertex, say, q, with respect to one direction along the corridor, C, containing q, candidates are generated for vertices visible to q in the other direction and occurring in $WAITLIST_1(q)$ and $WAITLIST_2(q)$. The ordering in which vertices are added to $WAITLIST$s will be the ordering in an L-Traversal of $TVIS_U(q)$ when candidates are to be generated from $TVIS_B(q)$ where U and B are the bounding edges of the corridor C. We assume that the visibility edges are visited in sorted angle order.

We show below that, as in the case of a simple corridor, the generation of candidates requires linear time. In order to generate can_1, can_2, can_3, and can_4 we will generate two kinds of candidates. From the first kind, can_h and can_i, we will obtain can_1 and can_2. From the second kind, can_j and can_k, we will obtain can_3 and can_4, respectively. Consider a sequence of vertices for which a candidate is generated from the visibility list in $TVIS_B(q)$, B a bounding edge. We show below that as in the case of simple corridors the direction of the pointer movement required to generate candidates is monotone.

Let q be a vertex with visibility trees $TVIS_B(q)$ and $TVIS_U(q)$, B and U being bounding edges. Let a and b be vertices visible to q through edge U. Let $q \in LIST_1(C_w)$ or $LIST_2(C_w)$, where C_w is a corridor at node $w \in TVIS_X(a)$, and let $q \in LIST_1(C_y)$ or $LIST_2(C_y)$, where C_y is a corridor at node $y \in TVIS_Y(b)$ for some bounding edges X and Y. Note that C_w and C_y are the same corridor.

Let $P1_a$ be the position of the candidate c_a in $TVIS_B(q)$, which gives can_h, the first kind of candidate, and which is the first candidate which follows q in some $LIST_1$ in $TVIS_X(a)$. The candidate c_a may be required to be added, as can_h, in an L-Traversal to $LIST_1$ of corridor C_w or of a corridor at the left or only son of w in $TVIS_X(a)$. Let $P1_b$ be the position of the candidate c_b in $TVIS_B(q)$ which is the first candidate following q, in some $LIST_1$ in $TVIS_Y(B)$. This candidate may be required for addition, as can_h, to $LIST_1$ of C_y or of a corridor at the left or only son of y in $TVIS_Y(b)$. Similarly, let $Q1_a$ and $Q1_b$ be the positions of d_a and d_b, which give can_i, the first kind of candidate if one exists, in $TVIS_B(q)$. The candidates d_a and d_b may be required to be added, as can_i, to $TVIS_X(a)$ and $TVIS_Y(b)$, respectively. In fact d_a and d_b are the first candidates following q in a L-Traversal which may be required to be added to $LIST_1$ at the right sons of $TVIS_X(a)$ and $TVIS_Y(b)$, respectively.

The following property, the proof of which is similar to Lemma 3.2 and hence

omitted, is stated next.

LEMMA 4.7. *If a follows $b, a \neq b$ in an L-Traversal of $TVIS_U(q)$, then*

- *$P1_a \succeq_{LIST_1} P1_b (Q1_a \succeq_{LIST_1} Q1_b)$ when $c_a(d_a)$ and $c_b(d_b)$ are in a $LIST_1$ at a node $v \in TVIS_B(q)$;*
- *alternatively, c_b (d_b) may be found at a node which is a predecessor of c_a (d_a) in an L-Traversal of $TVIS_B(q)$.*

The candidates found above are related to can_1 and can_2 and are required to be obtained using procedure *Find-next* as follows. Consider the case that the candidate is not in the current corridor C_w or C_y. Suppose can_1 exists at the left son of the current node w or y. Then can_h is can_1 and if can_i exists it is can_2. Note that in this case can_i is a vertex at a junction.

Alternatively, suppose can_1 does not exist at the left son of the current node w or y. In the first case, if no vertex is visible at the left son or a descendant of the left son of the current node in $TVIS_B(q)$, can_h, if it exists, is can_2. Otherwise, if a vertex is visible at the left son or a descendant of the left son of the current node, then, if can_i exists, can_i is can_2.

A similar property is true for pointer positions of the candidates for $LIST_2$ in $TVIS_X(a)$ and $TVIS_Y(b)$. In this case can_j and can_k are the two candidates to be found and are analogous to can_h and can_i. Let $P2_a$ be the position of the candidate e_a in $TVIS_B(q)$, which is the first candidate of the second kind, called can_k, following q to be added to a $LIST_2$ in $TVIS_X(a)$. This candidate may be required to be added in an L-Traversal to $LIST_2$ of C_w or a corridor at the right or only son of w in $TVIS_X(a)$. Similarly let $P2_b$ be the position in the tree of the candidate e_b in $TVIS_B(q)$ which is the first candidate following q, for addition to $LIST_2$ of $TVIS_Y(b)$. The candidate is added to C_y or a corridor at the right or only son of y in $TVIS_Y(b)$. Similarly, let $Q2_a$ and $Q2_b$ be the positions of another second kind of candidate can_j, if it exists, in $TVIS_B(q)$, termed f_a and f_b, respectively, which are the first candidates following q which may be added in an L-Traversal to $LIST_2$ at the left sons of $TVIS_X(a)$ and $TVIS_Y(b)$, respectively.

LEMMA 4.8. *If a follows $b, a \neq b$ in an L-Traversal of $TVIS_U(q)$, then*

- *$P2_b \succeq_{LIST_2} P2_a (Q2_b \succeq_{LIST_2} Q2_a)$ when $e_a(f_a)$ and $e_b(f_b)$ are in a $LIST_2$ at a node $v \in TVIS_B(q)$;*
- *alternatively, e_b (f_b) may be found at a node which is a predecessor of e_a (f_a) in an L-Traversal of $TVIS_B(q)$.*

We next show how the above properties ensure that a linear scan of lists at nodes in $TVIS_B(q)$ suffices to generate candidates for all points having q in their visibility list. Suppose we have just constructed a visibility edge from a vertex v to a vertex q in a corridor C. Assume that $q \in LIST_1(C)$. To extend the visibility tree of v, we need to obtain either one or two candidates for $LIST_1$ from the visibility tree $TVIS_B(q)$ for some B. To obtain the candidates from q we use the pointer positions when the candidates were last generated for a vertex v' which was added to the visibility list of q immediately preceding the addition of v. The following procedure is used to generate candidates can_h and can_i.

Let P_1 be the position of the pointer used to generate can_h for v' at a node x in the tree $TVIS_B(q)$. Let C_x be the corridor corresponding to node x. We now show how to find a candidate can^* from which the two candidates can_h and can_i will be obtained. The search for can^* starts with the current position of P_1, say, in $LIST_1(C_x)$, and proceeds linearly along the list, $LIST_1$, at node x and other nodes as would an L-Traversal of the tree $TVIS_B(q)$. The search stops having found the first

candidate vertex in a $LIST_1$ in the L-Traversal. There are two cases. Either can^* is found or not.

- *can^* exists.* Assume that $v \neq v'$. Further suppose that can^* is found at node y in the tree. We let can_h be can^*. Next we need to find can_i. There are two subcases. Either the node at which the candidate can_i for v' was found, if at all, is on the path, $path(y, root)$, from y to the root, $root$, of $TVIS_B(q)$, or it is not on $path(y, root)$.

  In the first subcase let w be the node. The search for can_i proceeds along the path from w to the root, i.e., along $path(w, root)$. This search for can_i requires considering the vertices at the junction triangles connecting adjacent corridors along the tree path. Let x' be a node on the path. Then a junction triangle connects the three corridors associated with x', $left(x')$, and $right(x')$. The vertex common to $C(left(x'))$ and $C(right(x'))$, called $vcr(x')$, is checked to be a candidate for can_i provided $right(x') \notin path(w, y)$. The search stops at the last node z on the path, $path(w, root)$ s.t. $vcr(z)$ is a candidate. (See Figure 6.)

  Alternatively, when can_i for v' was not on $path(y, root)$, the search for can_i proceeds as above along the path from y to the root. Note that in this case the previous node at which the candidate can_i for v' was found occurred prior to nodes on the path in an inorder traversal of the tree $TVIS_B(q)$.

  We now consider the case when $v = v'$. In this case, the candidate can_h previously found for v' has not been used. If can_i, found previously, also has not been used, then no new candidate need be discovered. If it has been used, then can_i is updated. Let $y \in TVIS_B(q)$ be the node at which can_h for v' was discovered. And let $w \in TVIS_B(q)$ be the node at which can_i for v' was discovered. Let z be the first descendant node of w on $path(w, y)$ with a right son not on $path(w, y)$. Then can_i is the vertex common to $C(left(z))$ and $C(right(z))$.
- *can^* does not exist.* Suppose that can^* was not found. Then neither can_h nor can_i can exist since a scan of $LIST_1$ at nodes obtained by an L-Traversal of the tree would have found can_h or can_i.

A symmetric procedure is also applied for the search for can_j and can_k. We repeat the details for completion. Let P_2 be the position of the pointer used to generate can_j for v at a node w in the tree $TVIS_B(q)$. We now show how to find a candidate can^* from which the two candidates can_j and can_k will be obtained. The search for can^* starts with the current position of P_2, say, in $LIST_2(C_x)$, and proceeds linearly along the list, $LIST_2$, at node x and other nodes as would an L-Traversal of the tree $TVIS_B(q)$. The search stops having found the first vertex which is a candidate in a $LIST_2$ of $TVIS_B(q)$. There are two cases. Either can^* is found or not.

- *can^* exists.* Assume that $v \neq v'$. Further suppose that can^* is found at node y in the tree. We let can_j be can^*. Next we need to find can_k. There are two subcases. Either the node at which the candidate can_k for v' was found, if it exists at all, is on the path from y to the root, $path(y, root)$, or not.

  In the first subcase let w be the node. The search for can_k proceeds along the path from w to y, i.e., along $path(w, y)$. This search for can_k requires considering the vertices at the junction triangles connecting adjacent corridors along the tree path. Let x' be a node on the path. Then the junction triangle connects the three corridors associated with x', $left(x')$, and $right(x')$. The vertex common to $C(left(x'))$ and $C(right(x'))$, called $vcl(x')$, is checked to

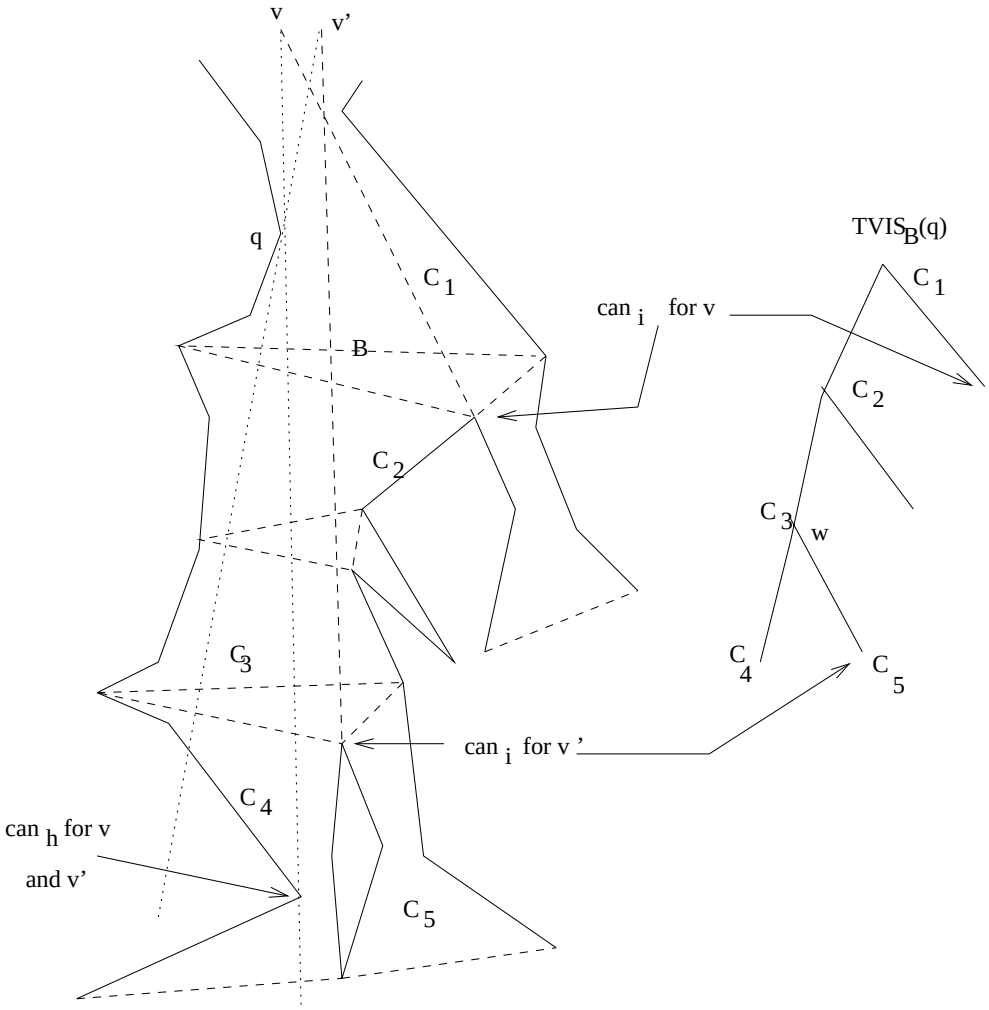


FIG. 6. *Searching for candidates can_h and can_i.*

be a candidate for can_k provided $left(x') \notin path(w, y)$. The search stops at the last node z on the path, $path(w, root)$, s.t. $vcl(z)$ is a candidate.

Alternatively, when can_k for v' was not on $path(y, root)$, the search for can_k proceeds as above along the path from y to the root. Note that in this case the previous node at which the candidate can_k for v' was found occurred prior to nodes on the path in an in-order traversal of the tree $TVIS_B(q)$.

Next suppose that $v = v'$. In this case, the candidate can_j previously found for v' has not been used. If can_k, found previously, also has not been used, then no new candidate need be discovered. If it has been used, then can_k is updated. Let $y \in TVIS_B(q)$ be the node at which can_j for v' is discovered. And let $w \in TVIS_B(q)$ be the node at which can_k for v' was discovered. Let z be the first descendant node of w on $path(w, y)$ with a left son not on $path(w, y)$. Then can_k is the vertex common to $C(left(z))$ and $C(right(z))$.

- *can^* does not exist.* Suppose that can^* was not found. Then neither can_j nor can_k can exist since a scan of $LIST_2$ at nodes obtained by an L-Traversal of the tree would have found can_j or can_k.

The closest candidate from among can_h, can_i, can_j, and can_k will give the candidate to extend the visibility tree of v as described in procedure *Find-next*.

LEMMA 4.9. *The candidates can be correctly generated in time linear in the size of the visibility trees.*

Proof. Consider a vertex q in corridor C with bounding edges U and B. We only consider generation of candidates which lead to the addition of new nodes in the visibility trees. Candidates along a corridor are easily determined as shown in the previous section. We prove the claim for the generation of the candidates can_h and can_i for vertices in $TVIS_U(q)$ from the lists $LIST_1$ at nodes in $TVIS_B(q)$. We will assume that candidates are generated for the vertices in $TVIS_U(q)$ in the same order as they occur in an L-Traversal of $TVIS_U(q)$. A similar analysis holds when the vertices are considered in reverse order of the L-Traversal. The proof for the generation of the candidates can_j and can_k from lists $LIST_2$ is similar.

The candidates can_h and can_i are generated using a pointer. Initially the pointer is at the beginning of the list of vertices obtained from $LIST_1(C = C_w), w \in TVIS_B(q)$, when the visible vertices are scanned in an L-Traversal of the tree. At the generic step there are two cases. Either the vertex, v, for which a candidate is to be found is different from the vertex, say, v', for which a candidate is found at the previous step or it is the same.

First, consider the case when the vertex is different. The pointer P_1 which generates can_h may move from its current position. By Lemma 4.7 this movement is monotone. We next show that can_i is correctly determined. If can_h and can_i exist, then can_i is a junction vertex since then both the visibility lists $LIST_1$ and $LIST_2$ are nonempty at the left and right sons, respectively, of the current node, called *curr*, in $TVIS_{B'}(v)$, where B' is a bounding edge. The tree node at which can_i exists is to be determined. First note that all junction vertices, which are candidates, occur consecutively along the path. Also, by Lemma 4.7, the position of the node at which can_i is obtained comes after its previous position in an L-Traversal of $TVIS_B(q)$. Thus it suffices to consider the path $path(w, root)$ or $path(y, root)$ where y is the node at which can_h is found and w is the node at which the can_i candidate for the vertex v' was found.

Next, consider the case when the vertices are the same, i.e., $v' = v$. In this case the position of can_h does not change. Let this position be at node y. can_i may have been added to a visibility list and used. Let *vcan* be the node at which can_i is found. Since can_i is added to a visibility list at the right son of the current node, *curr*, in $TVIS_{B'}(v)$ where B' is a bounding edge, candidates for addition to lists at the descendants of the left son are required. can_h is one candidate which may be added to a visibility list at the left son or a descendant node of the left son. To determine can_i it suffices to consider junction vertices at nodes which are proper descendants of the node *vcan* with their right son not on the path to y. This is because the junction vertices at these nodes are candidates and the junction vertex at the first such descendant is can_i. This path, P_s, is traversed only once for successive determination of candidates can_i for vertex v when can_i is used but can_h is not used as a candidate. Eventually the corridor containing can_h will be reached and either can_h will be used or it will be determined that the visibility subtree of v cannot be extended further. The time for traversing this path, P_s, is charged to the initial traversal which determined the

position of can_i when a candidate for v was computed.

Thus the lists in a tree, $TVIS_B(q)$, can be traversed in linear time to determine candidates for vertices visible to q through the other bounding edge of the corridor C. The lemma follows. □

Finally note that the determination of a candidate for a vertex u at q occurs only after candidates are determined for other vertices before u in an L-Traversal of the lists in the visibility tree $TVIS_B(q)$, B a bounding edge. Since vertices have to wait for candidates in $WAITLIST$s, there is a possibility of a deadlock. To avoid deadlock we adopt the following simple strategy.

4.5. Avoiding deadlock. We consider two directions, UP and $DOWN$. A visibility edge (u, v) corresponding to vertex v in the visibility list at u is assumed to be directed from u to v. The same edge is assumed to be directed from v to u when u occurs in the visibility list of v. A directed edge (u, v) is in the direction UP if $y(v) \geq y(u)$ and in the direction $DOWN$ if $y(u) > y(v)$. We partition the edges in the visibility tree $TVIS_B(u)$ into two sets. One set is the set of edges in the direction $DOWN$, termed $TDVIS_B(u)$, and the other in the direction UP, termed $TUVIS_B(u)$. A candidate for a vertex in $TUVIS_U(u)$ is obtained from $TDVIS_B(u)$ and $TUVIS_B(u)$ where U and B are the bounding edges of corridor C containing u. A candidate for a vertex in $TDVIS_U(u)$ is similarly obtained. A symmetric procedure applies when candidates are to be found for vertices in $TVIS_B(u)$.

The lists $TUVIS_B(u)$ and $TDVIS_B(u)$ are constructed by the procedure described above in the previous sections. We require candidates to start the construction. The first candidate for $TUVIS_B(u), u \in C$, through a bounding edge B, is obtained by finding the edge which a horizontal ray crossing B, in the direction of increasing x, strikes the polygonal structure. The endpoint of this edge above the ray gives the required first candidate. This first candidate can be found for all vertices simultaneously by a horizontal sweep through the corridors. Given the list of horizontal rays emanating through a bounding edge, each ray starting at a vertex in the corresponding corridor, the rays which traverse without obstruction a given corridor, C', which is adjacent to the current one via a junction, can be computed in $O(\log n)$ steps. The rays emanating from vertices in C' can be appended to the rays which traverse C' in time linear in the size of C'. In general, given a set of rays entering a corridor, C', through a bounding edge, U, the rays that emanate from the other bounding edge, B, can be obtained by using binary search to eliminate rays that strike the convex hull of any of the two sides of the corridor. Moreover, the rays that originate from a side, say, S_1, of C' can be obtained by a linear scan, which starts at the bounding edge U and eliminates rays obstructed by the edges on the corridor sides. The corridor processing requires $O(\log n + C'_r)$ where C'_r is the number of rays that originate from C'. Given that there are $O(m)$ corridors the sweep requires $O(m \log n + n)$ steps. The sweep details are similar to the sweep in [KM] which sweeps using the corridor structure to construct a restricted set of visibility edges. A similar sweep gives the first candidates in $TDVIS_B(u), u \in C$, and B a bounding edge, for all vertices u in the polygonal structure.

4.6. Correctness. The correctness of the scheme is obtained from the following properties.

LEMMA 4.10. (1) *The visibility lists $LIST_1(C)$ and $LIST_2(C)$ are correctly built in sorted order at a tree node.*

(2) *The nodes of $TVIS_B(v)$ are correctly generated.*

Proof. First, assume that all the required visibility lists are present. The proof of (1) is similar to the proof of Lemma 3.4.

To prove (2) we show that the visibility tree is correctly built. If candidates are available, then the correctness follows from Lemma 4.5.

We thus remove the assumption that the visibility lists containing the required candidates are present. We need to show that the algorithm is not deadlocked. For this purpose we consider the following rank function on visibility edges. Let u and v be vertices in corridors with bounding edges $B1, U1$ and $B2, U2$, respectively. Let v be a visible vertex in $TUVIS_{B1}(u)$ and u be a visible vertex in $TDVIS_{B2}(v)$. Let $prev(v)$ be the vertex prior to v in $TUVIS_{B1}(u)$ and let $prev(u)$ be the vertex prior to u in $TDVIS_{B2}(v)$. Note that the vertices are ordered by an L-Traversal of the visibility tree.

We let $v' = prev(v)$. We define $\text{rank}(u, v) = \max(r(u, v), r(v, u))$, where

- $r(u, v) = 1 + \text{rank}(u, v')$ when v is not visible to v';
- $r(u, v) = \text{rank}(u, v') + \text{rank}(v', v)$ when v is visible to v';
- $\text{rank}(u, v) = 1$ when v is the first vertex in the L-Traversal of the visibility tree of u.

The *rank* function is well defined since the rank $\text{rank}(u, v)$, $v \in TUVIS_{B1}(u)$, depends only on the rank of a vertex w above u, i.e., with $y(w) > y(u)$.

We prove by induction on the rank of edge (u, v) that when at time t a candidate is to be generated for v from $TVIS_{U1}(u)$, then $TVIS_{B1}(u)$ contains v as well as all vertices in the tree prior to v. Also at this step $TVIS_{B2}(v)$ contains u and all vertices prior to u.

Basis step. When the rank of an edge (u, v) is 1 the claim is trivially true since both vertices u and v are added as the first vertex in each other's visibility lists by the initial construction.

Induction step. Suppose that the induction hypothesis is true for all edges with rank $< k$. Consider an edge (u, v) with rank k. Let v' be $prev(v)$ in $TDVIS_{B1}(u)$, $v \in TDVIS_{B1}(u)$, and let u' be $prev(u)$ in $TUVIS_{B2}(v)$, $u \in TUVIS_{B2}(v)$. Since (u, v') has rank less than k, when a candidate for u is to be generated u has already been added to the visibility list of $TUVIS_{B'}(v')$, B' a bounding edge, and all vertices prior to u have been added to $TUVIS_{B'}(v')$. Similarly v is in a visibility list at $TDVIS_{B''}(u')$, B'' a bounding edge, and all vertices prior to v have also been added. When v is considered as a candidate for the visibility list of u after v', v' and all vertices prior to v' in $TDVIS_{B1}(u)$ have been added. And similarly u' and all vertices prior to u' also are present in $TUVIS_{B2}(v)$. The algorithm will thus generate the candidate v for addition to the visibility list of u and the candidate u for addition to the visibility list of v using vertices already in the visibility lists at u' and v', respectively. The conditions for the generation of candidates are satisfied as shown above. Thus v will be added to $TDVIS_{B1}(u)$ and u will be added to $TUVIS_{B2}(v)$. And all vertices prior to v and u have been added to $TDVIS_{B1}(u)$ and $TUVIS_{B2}(v)$, respectively. This completes the induction step.

The algorithm is thus never deadlocked.

The lemma follows. □

We have thus shown that the algorithm correctly builds the visibility lists.

Since the data structures used in the algorithm are lists which store the visibility edges at each node of the visibility trees, the space requirement is linear in the number of visibility edges and vertices in the polygonal structure. Together with the analysis of the time complexity in the previous section we obtain the last theorem.

THEOREM 4.11. *GVIS(V,E), the visibility graph of V, the set of vertices of a simple n-vertex polygon with m obstacles, can be correctly constructed in time $O(\mid E \mid + T + m \log n)$ and $O(E)$ space, where $\mid E \mid$ is the number of edges in the visibility graph and T is the time required to triangulate the polygon with obstacles.*

5. Conclusions. We may note that the algorithm presented above does not use sorting but only a triangulation of the polygonal structure.

The corridor structure used in this algorithm for partitioning the polygonal structure has been used for another algorithm to construct a restricted visibility graph, $G_s = (V, E_s)$, which contains shortest path information [KM]. In this graph common tangents between the corridor chains, where a corridor chain is a convex chain enclosing a side of a corridor, are required. The common tangents from a corridor chain, CH_C, in C through a bounding edge B to other corridor chains, can be stored in a tree structure, called $TANVIS_B(CH_C)$, similar to $TVIS_B(v), v \in C$. With each node of the tree are associated a corridor chain, say, $CH_{C'}$, and a tangent common to both corridor chains, CH_C and $CH_{C'}$, if there is a visible common tangent. The tangents are constructed using the trees for other corridor chains and the sides of the corridors as in the algorithm presented above. The crucial property is a monotonicity property of the position of the tangents in the corridors as well as at the nodes of the tree $TANVIS_B(CH_C)$. This holds when the corridor chains, for which tangents are to be constructed using CH_C, are considered in an in-order traversal of $TANVIS_U(CH_C)$, where U is the other bounding edge of C. This monotonicity property ensures that each tree will be scanned once during the construction of all the tangents in the restricted graph. Also each vertex in the corridor is scanned once to construct the tangents. The above outlines an extension of the scheme, described above in the paper, for constructing the restricted visibility graph in $O(|E_s| + T + m \log n)$ time.

Finally, we would like to thank the referees for their helpful comments on the manuscript.

REFERENCES

[AAG+86] T. ASANO, T ASANO, L. GUIBAS, J. HERSHBERGER, AND H. IMAI, *Visibility of disjoint polygons*, Algorithmica, 1 (1986), pp. 49–63.

[BYC94] R. BAR-YEHUDA AND B. CHAZELLE, *Triangulating disjoint jordan chains*, Internat. J. Comput. Geom. Appl., 4 (1994), pp. 475–481.

[GM91] S.K. GHOSH AND D.M. MOUNT, *An output-sensitive algorithm for computing visibility graphs*, SIAM J. Comput., 20 (1991), pp. 888–910.

[Her89] J. HERSHBERGER, *Finding the visibility graph algorithm of a simple polygon in time proportional to its size*, Algorithmica, 4 (1989), pp. 141–155.

[KM88] S. KAPOOR AND S.N. MAHESHWARI, *Efficient algorithms for Euclidean shortest paths and visibility graphs amongst polygonal obstacles*, in proceedings of the Fourth ACM Symposium on Computational Geometry, Urbana-Champaign, IL, 1988, pp. 172–182.

[KMM97] S. KAPOOR, S.N. MAHESHWARI, AND J.S.B. MITCHELL, *Efficient algorithms for Euclidean shortest paths amongst polygonal obstacles*, Discrete Comput. Geom., 18 (1997), pp. 377–383.

[OW88] M.H. OVERMARS AND E. WELZL, *New methods for computing visibility graphs*, in Proceedings of the Fourth ACM Symposium on Computational Geometry, Urbana-Champaign, IL, 1988, pp. 164–171.

[PS85] F.P. PREPARATA AND M.I. SHAMOS, *Computational Geometry—An Introduction*, Springer-Verlag, Berlin, New York, Heidelberg, 1985.

[Wel85] E. WELZL, *Constructing the visibility graph for n line segments in $o(n^2)$ time*, Inform. Process. Lett., 20 (1985), pp. 167–171.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 872–905

OPTIMAL WORST CASE FORMULAS COMPARING CACHE MEMORY ASSOCIATIVITY*

HÅKAN LENNERSTAD† AND LARS LUNDBERG‡

Abstract. In this paper we derive a worst case formula comparing the number of cache hits for two different cache memories. From this various other bounds for cache memory performance may be derived.

Consider an arbitrary program P which is to be executed on a computer with two alternative cache memories. The first cache is set-associative or direct-mapped. It has k sets and u blocks in each set; this is called a (k, u)-cache. The other is a fully associative cache with q blocks—a $(1, q)$-cache.

We derive an explicit formula for the ratio of the number of cache hits $h(P, k, u)$ for a (k, u)-cache compared to a $(1, q)$-cache for a worst case program P. We assume that the mappings of the program variables to the cache blocks are optimal.

The formula quantifies the ratio

$$\inf_P \frac{h(P, k, u)}{h(P, 1, q)},$$

where the infimum is taken over all programs P with n variables. The formula is a function of the parameters n, k, u, and q only. Note that the quantity $h(P, k, u)$ is NP-hard.

We assume the commonly used LRU (least recently used) replacement policy, that each variable can be stored in one memory block, and that each variable is free to be mapped to any set.

Since the bound is decreasing in the parameter n, it is an optimal bound for all programs with at most n variables. The formula for cache hits allows us to derive optimal bounds comparing the access times for cache memories. The formula also gives bounds (these are not optimal, however) for any other replacement policy, for direct-mapped versus set-associative caches, and for programs with variables larger than the cache memory blocks.

Key words. cache memory, fully associative cache, set-associative cache, direct-mapped cache, 0, 1-matrices, performance bound, extremal matrices

AMS subject classifications. 68R05, 68M07, 90C27, 05A05, 05D99

PII. S0097539798349164

1. Introduction.

Cache memories reduce the memory access time in computer systems; an excellent survey can be found in [20]. The hardware budget is, however, limited in terms of gates, routing, etc., when designing cache memories. This hardware budget can be used in different ways, e.g., one can include different amounts of associative memories thus yielding a fully associative, set-associative, or direct-mapped cache. Due to the limited hardware budget, using associative memories reduces the number of storage elements in the cache. Moreover, the access time usually increases with increasing associativity. Consequently, trade-offs between the access time, the degree of associativity, and the number of storage elements are of interest. Such design trade-offs are difficult. One aspect here is the implementation cost of associative memories as compared to the storage elements. This ratio depends on the implementation technique [17].

*Received by the editors December 14, 1998; accepted for publication (in revised form) November 9, 1999; published electronically August 9, 2000. This work was partially supported by the Blekinge Research Foundation.

http://www.siam.org/journals/sicomp/30-3/34916.html

†Department of Mathematics, University of Karlskrona/Ronneby, S-371 79 Karlskrona, Sweden (hakan@itm.hk-r.se).

‡Department of Computer Science, University of Karlskrona/Ronneby, S-372 25 Ronneby, Sweden (lars.lundberg@ipd.hk-r.se).

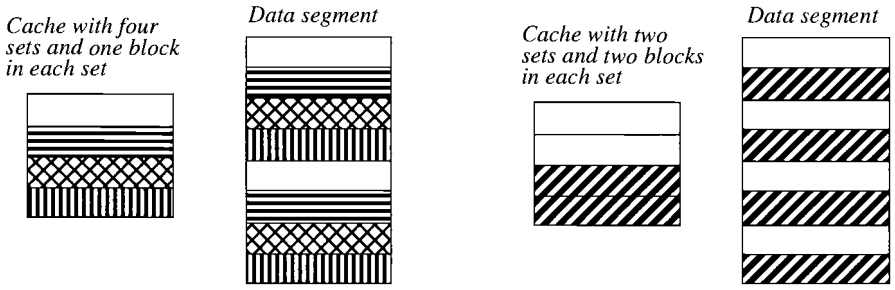


FIG. 1. *A direct-mapped and a set-associative cache with two sets for a data segment of size* 8.

Another aspect is the maximum difference in hit ratio between small fully associative caches and larger set-associative or direct-mapped caches. Calculating the maximum hit ratio decrease when reducing the number of storage elements is not difficult, i.e., in the worst case scenario the hit ratio may drop to zero. This can best be understood by considering a loop with a large number of iterations. (The definition of a loop can be obtained from any textbook on introductory computer programming.) In each loop iteration we access the same n variables. If we use the LRU or FIFO replacement policies [21] and reduce the number of cache blocks from n to $n-1$, the cache hit ratio will drop from 100 percent to zero. Estimating the maximum hit ratio increase when increasing the degree of associativity and decreasing the total number of storage elements is, however, nontrivial. Figure 1 illustrates two cases of different associativity.

A problem with direct-mapped caches and caches with a limited degree of associativity, i.e., a limited set size, is that the performance for certain access patterns may be seriously impaired due to collisions. Consider a direct-mapped cache and a loop with a large number of iterations. Two different variables are accessed in the loop. If these two variables are mapped to the same set, there will be collisions, and each variable reference will result in a cache miss. However, if we consider a system with a degree of associativity larger than or equal to two, no collisions will occur; with the exception of the initial references, all references to the two variables result in cache hits.

If the variables can be mapped independently of each other, the compiler or programmer may detect the collisions and try to map the variables to different sets. However, in some cases the number of collisions will increase when the set size decreases, even when using the best possible mapping of variables to sets. This can be understood with the aid of the example below.

Consider a program with three loops, each containing a large number of iterations. In loop one, variables A and B are accessed repeatedly. In loop two, variables B and C are accessed, and in loop three, variables A and C are accessed. Only one variable can be allocated in a block. Consider also a small cache system with two blocks. This system can be arranged in one of two ways: direct-mapped or fully associative. In the direct-mapped case, at least two of the variables A, B, and C are mapped to the same set. As a result, at least one of the three loops results in a large number of collisions, even when using the best possible mapping of variables to sets. In the fully associative case, there will be no collisions.

In this paper we establish several comparisons of cache memory performance, all derived from one basic optimal formula. The main part of the report is focused on this

particular formula. It is an optimal upper bound on the maximum relative increase of cache hits when using a fully associative cache as compared to a set-associative or direct-mapped cache. In section 9 we show that the result can also be used for comparing set-associative caches with direct-mapped caches. We assume that the compiler, or programmer, is free to map variables to any arbitrary cache set. In some cases, e.g., for different elements in large variables such as arrays and matrices, this is obviously not true. In section 9, however, we will demonstrate that the results are indeed valid for a large number of real cases, including many programs which access large variables. From an applied point of view, section 3, which presents the formula, and section 9, which presents various applications and extensions of the main formula, are the most important sections.

The present paper is part of a series of reports on optimal worst case bounds in computer science contexts. We have also derived optimal bounds for static versus dynamic allocation of parallel programs [7, 9], for cluster versus dynamic allocation [8], for all programs with a specified parallelism [10], for the worst case performance drop at memory reorganization [11], and for the cluster execution time for a program P only knowing one or more execution times for P, executed with dynamic allocation and any schedule [12, 13]. We have also obtained optimal performance bounds on the gain of permitting dynamic allocation of communication channels in distributed processing [14]. All these reports provide optimal bounds for NP-hard quantities [2]. In the present report we consider the NP-hard problem of finding an optimal mapping for a certain program P with a (k, u)-cache, i.e., a mapping maximizing the number of cache hits. We provide a formula for this number for worst case programs.

The method in all the reports consists of two steps:

1. Successive elimination of unnecessary programs, leading to a subset of all programs. This set contains at least one extremal (worst case) program and allows matrix representation of the remaining programs.
2. In the remaining set of programs—sometimes regarded as matrices—those where all possible rows occur equally frequently are proven to be extremal. Here we have obtained a formulation which is sufficiently simple to allow the derivation of explicit formulas.

The first step is, of course, application-dependent to a higher degree than the second. In the present report, the first step is taken care of in sections 2–6, while sections 7 and 8 deal with the second step. Section 9 gives different applications and generalizations of the results. Section 10 contains a short discussion of the results in relation to existing research. Section 11 displays plots of the formula as well as an application which takes memory access times into consideration.

2. Problem definition and notation. A data cache is defined by five parameters: the number of sets, the number of blocks in each set, the block size, the replacement policy within a set, and the update policy to main memory. We focus here on two parameters: the number of cache sets and the number of blocks in each set. A cache with k sets containing u blocks each is denoted as (k, u)-cache. In some reports, this cache configuration is referred to as a u-way set-associative cache with k sets. A *mapping* maps each variable in a program onto a specific memory address and thus to a specific cache set. During the complete execution each variable will be stored on this address. Thus a mapping can be thought of as a partition of the variables into k sets.

If u equals one, the cache is direct-mapped. If k equals one, the cache is fully associative. It is well known [5] that with a fixed number of blocks, the hit ratio is

higher, or at least equal, with a fully associative cache than with a direct-mapped cache. The left part of Figure 1 shows a $(4, 1)$-cache, and the right part shows a $(2, 2)$-cache for a data segment of size 8.

In this study, the results are valid for any block size (for further details see the discussion in section 9.4). However, all blocks are of the same size. We are only interested in the hit ratio, i.e., the percentage of all memory references that can be handled by the cache. The hit ratio is unaffected by the update policy, e.g., write-through or copy-back. The update policy is thus of no importance in this study.

In cache memory systems, code references are much easier to handle than data references. In this report, we only consider data references. When the code references are ignored, the execution of the program P can be regarded as a sequence of $m(P)$ references to n different program variables. All variables are stored in the data segment. We assume that the data segment is infinitely large, and that the compiler or programmer is free to map any variable onto any block. This assumption is discussed in section 9. We also assume that all memory words in a variable are stored in the same block. We only consider variables which are referenced at least once, and hence $n \leq m(P)$.

The only limit on the size of the program variables is that no variable should exceed one cache memory block. This restriction is discussed and to a large extent removed in section 9. Several small variables may be mapped to the same cache block. If a variable in one block is transferred to, or from, the cache, all other variables mapped to the same block are transferred simultaneously. In Theorem 2.2, we show that we may in the remainder of the report consider only the case of having exactly one variable mapped to each block.

No initial storing is assumed; the first referenced block will thus be stored in the cache without replacing any other variable. We assume that the LRU [16] replacement policy is used within each set. This means that if a cache set is full and a variable mapped to this set is referenced which is not in the cache, then the variable—together with other variables mapped to the same block—replaces the least recently referenced block in the cache set. This is a reasonable replacement policy which has been used in some machines [6] as well as in other studies (e.g., see [5] and [19]). We discuss other replacement policies in section 9.

If the number of sets exceeds one, different mappings of variables to memory addresses may result in different hit ratios. Consider, for example, a direct-mapped cache and a program which repeatedly accesses two variables in a loop. If these two variables are mapped to the same set, each access will result in a cache miss. If, however, the two variables are mapped to different sets, there would be no cache misses, apart from the initial references. A mapping which for a particular program P results in a maximal number of hits is referred to as an *optimal mapping* for P. The number of hits for a program P, using an optimal mapping and a (k, u)-cache, is denoted $h(P, k, u)$. The number of hits for a program P with an arbitrary mapping A is denoted by $h(A, P, k, u)$. Hence, if A_1 is an optimal mapping for the program P, $h(A_1, P, k, u) = h(P, k, u)$.

Some of the memory references that can be handled by the $(1, q)$-cache may lead to misses using the (k, u)-cache, at least for some values on n, k, q, and u. References which are hits using the $(1, q)$-cache but misses using the (k, u)-cache are referred to as *extra misses*. We now consider a mapping which, using a (k, u)-cache, minimizes the number of extra misses. Using this mapping, the number of extra misses for a certain program P is denoted $em(P, k, u, q)$. The number of extra misses with a specific

mapping A is denoted by $em(A, P, k, u, q)$. In mathematical language, all possible mappings are all possible partitions of the n variables in k sets. We summarize as follows.

DEFINITION 2.1. *Given a program P, we denote the maximal number of hits and the minimal number of extra misses as follows:*

$$h(P, k, u) = \max_A h(A, P, k, u),$$

and

$$em(P, k, u, q) = \min_A em(A, P, k, u, q).$$

Note that $h(P, k, u)$ and $em(P, k, u, q)$ may be realized with different mappings. The problem of finding an optimal mapping using a (k, u)-cache is NP-hard (see [2, problems P01 and P02], for example).

We next prove that in the interest of worst case estimates, we need only to consider programs where each variable is the size of one block. Note that in this case $h(A_1, P, 1, q) = h(A_2, P, 1, q)$ for any mappings A_1 and A_2. As a result, we need no notation of mappings in conjunction with the quantity $h(P, 1, q)$. In the following theorem, n-programs denote programs with n variables.

THEOREM 2.2.

$$\inf_{n\text{-programs } P} \frac{h(P, k, u)}{h(P, 1, q)} = \inf_{n\text{-programs } P \text{ with blocksized variables only}} \frac{h(P, k, u)}{h(P, 1, q)}.$$

Proof. Consider any program P with n variables in which some variables are mapped to the same block when using an optimal mapping for a $(1, q)$-cache. Suppose that the variables $i_1, i_2, \ldots, i_j$ are mapped to the same block. Then there is another program P', which has none of the variables $i_1, i_2, \ldots, i_j$, but a new variable i_0. The size of i_0 equals the sum of the sizes of the variables $i_1, i_2, \ldots, i_j$, and each reference to $i_1, i_2, \ldots, i_j$ in P is replaced by a reference to i_0 in P'. Since all variables mapped to the same block are transferred to and from the cache simultaneously, we have $h(P, 1, q) = h(P', 1, q)$.

Next consider a (k, u)-cache. By the construction the mappings of the variables of P' can be regarded as those mappings of the variables of P where the variables $i_1, i_2, \ldots, i_j$ are mapped to the same block. In addition, for each mapping A' of the variables of P', there is a mapping A of the variables of P so that $h(A', P', k, u) = h(A, P, k, u)$. Hence

$$h(P', k, u) = \max_{A'} h(A', P', k, u) \leq \max_A h(A, P, k, u) = h(P, k, u).$$

Thus

$$\frac{h(P', k, u)}{h(P', 1, q)} \leq \frac{h(P, k, u)}{h(P, 1, q)}.$$

By repeating this argument we obtain a program P'' where no variables are mapped to the same block, and where

$$\frac{h(P'', k, u)}{h(P'', 1, q)} \leq \frac{h(P, k, u)}{h(P, 1, q)}.$$

We also increase the size of each variable in P'' to equal the size of one block. Clearly this also preserves the inequality.

Denote the number of variables of the program P'' by y; obviously, $y < n$. We next construct a program P''' with n variables by adding one reference each to $n - y$ new variables as the $n-y$ last references in the program P''. Each new variable is the same size as one block. The $n - y$ last references in the program P''' are all misses in any cache and with any mapping since the first reference to a block is always a miss. Hence we can also obtain an optimal mapping from an optimal mapping of the program P'' by adding the new variables to any block. Hence $h(P'', 1, q) = h(P''', 1, q)$, and $h(P'', k, u) = h(P''', k, u)$.

Given any program P with n variables, we can thus construct a program P''' with n variables, where each variable is the size of one block, and where

$$\frac{h(P''', k, u)}{h(P''', 1, q)} \leq \frac{h(P, k, u)}{h(P, 1, q)}. \qquad \square$$

We also need the quantity $eh(A, P, k, u, q)$, the extra hits. This is the number of memory references for P which are misses with a $(1, q)$-cache but hits using a (k, u)-cache, when using a mapping A. We clearly have the following relation.

LEMMA 2.3. $h(A, P, k, u) - h(P, 1, q) = eh(A, P, k, u, q) - em(A, P, k, u, q)$.

The significance of the results is based on the calculation of an explicit formula for the following function.

DEFINITION 2.4.

$$r(n, k, u, q) = \max_P \frac{em(P, k, u, q)}{h(P, 1, q)},$$

where the maximum is taken over all programs P with n variables.

The formula for $r(n, k, u, q)$ is presented in Theorem 3.2.

3. Main results. Our first result states that the formula for

$$\max_P em(P, k, u, q)/h(P, 1, q)$$

can be used to obtain optimal worst case estimates for the hit ratio comparing the two caches. Here we thus obtain an optimal bound on the NP-hard quantity $h(P, k, u)$.

THEOREM 3.1.

(i) *The worst case ratio of cache hits comparing two caches is given by*

$$\inf_P \frac{h(P, k, u)}{h(P, 1, q)} = 1 - r(n, k, u, q),$$

where the infimum is taken over all programs P with n variables.

(ii) *Consider a program P with n variables, and suppose that the hit ratio $h(P, 1, q)/m(P)$ for P with a $(1, q)$-cache is known. In this case, an optimal estimate of the hit ratio for the program P using a (k, u)-cache is given by*

$$\inf_P \frac{h(P, k, u)}{m(P)} = (1 - r(n, k, u, q)) \frac{h(P, 1, q)}{m(P)}.$$

The proof of Theorem 3.1 is given at the end of this section. The function $1 - r(n, k, 1, k)$, which compares a direct-mapped cache to a fully associative cache, is plotted in section 11.2.

The formula for $r(n,k,u,q)$ is given in the following theorem.

THEOREM 3.2. *Suppose that n,k,u and q are positive integers, and denote $I = \lfloor \frac{n-i}{k-1} \rfloor$ and $J = n - i - (k-1)\lfloor \frac{n-i}{k-1} \rfloor = (n-i) mod (k-1)$.*

Then, if $n \le ku$ or $u \ge q$, $r(n,k,u,q) = 0$. If $ku < q$ and $ku < n$, then $r(n,k,u,q) = 1$. In the remaining cases we may compute the value of $r(n,k,u,q)$ as

$$r(n,k,u,q) = \frac{1}{q\binom{n}{q}} \min_{i=\lceil n/k \rceil, \lceil nu/q \rceil, \lceil nu/q \rceil + 1, \ldots, n} \left(\sum_{j=u+1}^{\min(q,i)} j \binom{i}{j} \binom{n-i}{q-j} + J \sum_{j=u+1}^{\min(q,I+1)} j \binom{I+1}{j} \binom{n-I-1}{q-j} + (k-1-J) \sum_{j=u+1}^{\min(q,I)} j \binom{I}{j} \binom{n-I}{q-j} \right),$$

or as

$$r(n,k,u,q) = 1 - \frac{1}{q\binom{n}{q}} \max_{i=\lceil n/k \rceil, \lceil nu/q \rceil, \lceil nu/q \rceil + 1, \ldots, n} \left(\sum_{j=1}^{\min(u,i)} j \binom{i}{j} \binom{n-i}{q-j} + J \sum_{j=1}^{\min(u,I+1)} j \binom{I+1}{j} \binom{n-I-1}{q-j} + (k-1-J) \sum_{j=1}^{\min(u,I)} j \binom{I}{j} \binom{n-I}{q-j} \right).$$

By counting terms in the sums it is easily seen that the first formula has fewer terms if and only if

$$\min(q,i) - \min(u,i) + J(\min(q,I+1) - \min(u,I+1)) + (k-1-J)(\min(q,I) - \min(u,I)) \ge ku.$$

That is, for $q \ge 2u$, the first formula is computationally most efficient, and if u is close to 1, the second is favorable.

This theorem is proved in section 8.

We also prove that the function r is increasing.

LEMMA 3.3. *For any $n > 0$, $r(n,k,u,q) \le r(n+1,k,u,q)$.*

By this lemma, which is proved in section 8, a bound for a large n is valid as a bound for all programs with at most n variables. We thus have the following more n-independent version of Theorem 3.1(i).

THEOREM 3.4. *The worst case ratio of cache hits comparing two caches is given by*

$$\inf_P \frac{h(P,k,u)}{h(P,1,q)} = 1 - r(n,k,u,q),$$

where the infimum is taken over all programs P with at most n variables.

We will next describe the results for specific programs (Theorem 3.9) from which the general results of Theorems 3.1 to 3.4 follow. These also have some interest in

their own right. Below, we describe the complete programs: these are extremal for the ratio $\max_P em(P,k,u,q)/h(P,1,q)$ and essential in the calculation of the formula.

As discussed previously, a program can be represented by a sequence of m integers, each between 1 and n, where each integer occurs at least once since each variable is referenced at least once. In this sequence, each entry represents one memory reference, and its value denotes the variable which is referenced.

When executed using a $(1,q)$-cache, the program induces a sequence of states for the cache. Our results are based on an analysis of this sequence of states. We will next describe its representation briefly—the full description is given in section 6.

For any given program P we initially add n references, one reference to each of the n variables, in the same order as they are initially referenced in the program P. This gives the program P' with $m(P') = m(P) + n$. Note that if $n > q$, we add n misses using a $(1,q)$-cache.

For any reference in the program P' after the initial n references, the state of the cache is described by a so-called (q,n)-hit vector. This vector contains one entry for each variable; the content describes whether the variable is in the cache or not. There are $n-q$ 0's for variables not in the cache. Variables in the cache are denoted by "1," except the least recently referenced variable in the cache, which is represented by "q" at the corresponding position.

The program P is represented by a matrix $M(P)$ having as rows all induced (q,n)-hit vectors of P' which represent hits using a $(1,q)$-cache. If $n > q$, P' has at least $n+1$ initial misses, i.e., the number of rows of the matrix $M(P)$ is less than $m(P)$. Obviously there exist $q\binom{n}{q}$ different (q,n)-hit vectors.

The raison d'etre for the (q,n)-hit vectors is that we represent enough information for our purpose, and that any collection of (q,n)-hit vectors represents a program (Theorem 6.12). Therefore, results about programs can be derived by studying collections of (q,n)-hit vectors. The main part of section 6 is devoted to establishing the fact that the problem can be studied using the (q,n)-hit vector representation.

DEFINITION 3.5. *A program P is complete if all $q\binom{n}{q}$ possible (q,n)-hit vectors are equally frequent in the matrix $M(P)$.*

THEOREM 3.6. *All complete programs are extremal for the ratio $em(P,k,u,q)/h(P,1,q)$, i.e., for each complete program P we have*

$$em(P,k,u,q)/h(P,1,q) = \max_{P'} em(P',k,u,q)/h(P',1,q).$$

Here the maximum is taken over all programs P' with the same number of variables as P.

The main part of the report is devoted to proving the correctness of this theorem. In parallel with this we prove the usefulness of the (q,n)-hit vector description.

We next consider optimal mappings of the n variables onto the k sets. A mapping where each set contains either $\lfloor n/k \rfloor$ or $\lceil n/k \rceil$ variables is called a uniform mapping. A mapping where one set has i variables, where $i > \lceil n/k \rceil$, and the other $k-1$ sets have $\lfloor (n-i)/(k-1) \rfloor$ or $\lceil (n-i)/(k-1) \rceil$ variables is called a quasi-uniform mapping.

We prove the following theorem, which is essential in order to compute the Theorem 3.2 formula.

THEOREM 3.7. *For any complete program P there is an optimal mapping which is either uniform or quasi-uniform.*

In section 7 we prove that, practically speaking, complete programs may have any hit ratio. We in fact prove that the set of possible hit ratios for complete programs is dense in the interval $(0,1)$.

LEMMA 3.8. *For any number $H, 0 \le H \le 1$, and for any $\epsilon > 0$, there is a complete program P such that*

$$\left| \frac{h(P,1,q)}{m(P)} - H \right| < \epsilon.$$

We conclude this section by showing how Theorem 3.1 follows once Theorem 3.6 and Lemma 3.8 have been established.

THEOREM 3.9.

(i) *For any program P with n variables, we have*

$$h(P,1,q)(1 - r(n,k,u,q)) \le h(P,k,u).$$

(ii) *For any complete program P with n variables, we have*

$$h(P,1,q)(1 - r(n,k,u,q)) \le h(P,k,u) < m(P) - h(P,1,q)r(n,k,u,q).$$

Proof. Given a program P, use A_0 and A_1 to denote optimal mappings for the number of cache hits and for the number of extra misses, respectively. From Lemma 2.3 we then obtain

$$\begin{aligned} h(P,k,u) = h(A_0,P,k,u) &\ge h(A_1,P,k,u) \\ &\ge h(P,1,q) - em(A_1,P,k,u,q) = h(P,1,q) - em(P,k,u,q). \end{aligned}$$

Hence

$$h(P,k,u) \ge h(P,1,q) - em(P,k,u,q).$$

By invoking $r(n,k,u,q) \ge em(P,k,u,q)/h(P,1,q)$ the inequality in (i) follows.

The right inequality for complete programs follows from the relation

$$h(A,P,k,u) + em(A,P,k,u,q) < m(P).$$

To the left we here have all hits and a subset of the misses when using a (k,u)-cache, and to the right we have all memory references. Only the misses which are also misses with a $(1,q)$-cache are not represented. Since the initial reference is a miss for both caches, a strict inequality is valid. By taking optimal mappings A_0 and A_1 as before we get

$$\begin{aligned} h(A_0,P,k,u) < m(P) - em(A_0,P,k,u,q) &\le m(P) - em(A_1,P,k,u,q) \\ &= m(P) - em(P,k,u,q). \end{aligned}$$

Now, by invoking Theorem 3.6, for complete programs P we have $r(n,k,u,q) = em(P,k,u,q)/h(P,1,q)$; the right inequality follows. □

Proof of Theorem 3.1(i). By taking $h(P,1,q)/m(P)$ arbitrarily close to 1, Theorem 3.1(i) follows immediately from Lemma 3.8, Theorem 3.6, and Theorem 3.9.

4. Method overview. The results in section 3 are established when we have proved the following facts for complete programs.

1. Complete programs are extremal: Theorem 3.6.
2. The formula in Theorem 3.2 is valid for complete programs. Part of this is to establish that a complete program has an optimal mapping which is uniform or quasi-uniform (Theorem 3.7).

3. There are complete programs with any hit ratio: Lemma 3.8.

In order to derive a mathematical formula for $r(n, k, u, q)$ for all positive integers n, k, q, and u, we start by taking care of trivial cases of integers n, k, q, and u (section 5). For the nontrivial cases, we successively take away uninteresting sets of programs with the argument that if a program which is removed is extremal, then there is another program which is also extremal and which is not removed. Thus we always keep at least one program which is extremal for the problem.

We next represent the sequence of states for a $(1, q)$-cache which is induced by the program P with a sequence of so-called (q, n)-state vectors. This vector describes the state of the cache and thus contains enough information about the history of P to decide whether the next reference is a hit or a miss.

We have seen that the minimum of the ratio $h(P, k, u)/h(P, 1, q)$ can be obtained by calculating the maximum of $em(P, k, u, q)/h(P, 1, q)$. For this ratio, the $(1, q)$-cache misses are uninteresting and they are not represented in the sequence of (q, n)-state vectors. By frequent use of this flexibility, i.e., by adding $(1, q)$-cache misses in certain ways, it is possible to prove that any arbitrary sequence of (q, n)-state vectors represents a valid program. Theorem 6.9 thus states that, starting with any (q, n)-state vector, one can add $(1, q-1)$-cache misses so that the next hit occurs at any other state for the $(1, q-1)$-cache.

With the aid of Theorem 6.11 we can always add cache misses so that the next hit always references the variable which in this time cycle is the least recently referenced one. This allows the representation to be further simplified into so-called (q, n)-hit vectors. The aim of section 6 is Theorem 6.12. Using this theorem, it becomes valid to consider arbitrary sequences of (q, n)-hit vectors.

In section 7, complete programs are defined and studied. In this section, Theorems 3.6 and 3.7 and Lemma 3.8 are proved. The proof of Theorem 3.6 consists of the so-called duplication argument, which is a cornerstone in [7, 8, 9, 10, 11, 12, 13, 14].

The calculation of the explicit formula, Theorem 3.2, is made in section 8. In section 9 the applicability of the results is discussed, and section 10 concludes the paper. Section 11 displays some graphics related to the results.

Figures 2 and 3 present the logical structure of the theorems and lemmas needed to establish Theorems 3.1 and 3.2.

5. Trivial cases.

THEOREM 5.1. *If $n \leq ku$, then $r(n, k, u, q) = 0$.*

Proof. If $n \leq ku$, then all variables can be contained within a cache with k sets each containing u blocks. As a result, there will be no extra misses for the (k, u)-cache. □

THEOREM 5.2. *If $ku < q$ and $ku < n$, then $r(n, k, u, q) = 1$.*

Proof. From the definition of $r(n, k, u, q)$ it is obvious that $r(n, k, u, q) \leq 1$ for any values of n, k, q, and u.

If $ku < q$, it is possible to write a program P which repeatedly accesses a set of variables requiring q blocks of memory space. Since we use the LRU replacement policy, we will obtain a cache-miss for every reference using the (k, u)-cache. For the $(1, q)$-cache we will, however, obtain no misses after the first iteration. By increasing the number of such iterations, it is possible to make the ratio $em(P, k, u, q)/h(P, 1, q) > 1 - \epsilon$ for any $\epsilon > 0$. Thus $\sup em(P, k, u, q)/h(P, 1, q) = 1$. □

THEOREM 5.3. *If $u \geq q$, then $r(n, k, u, q) = 0$.*

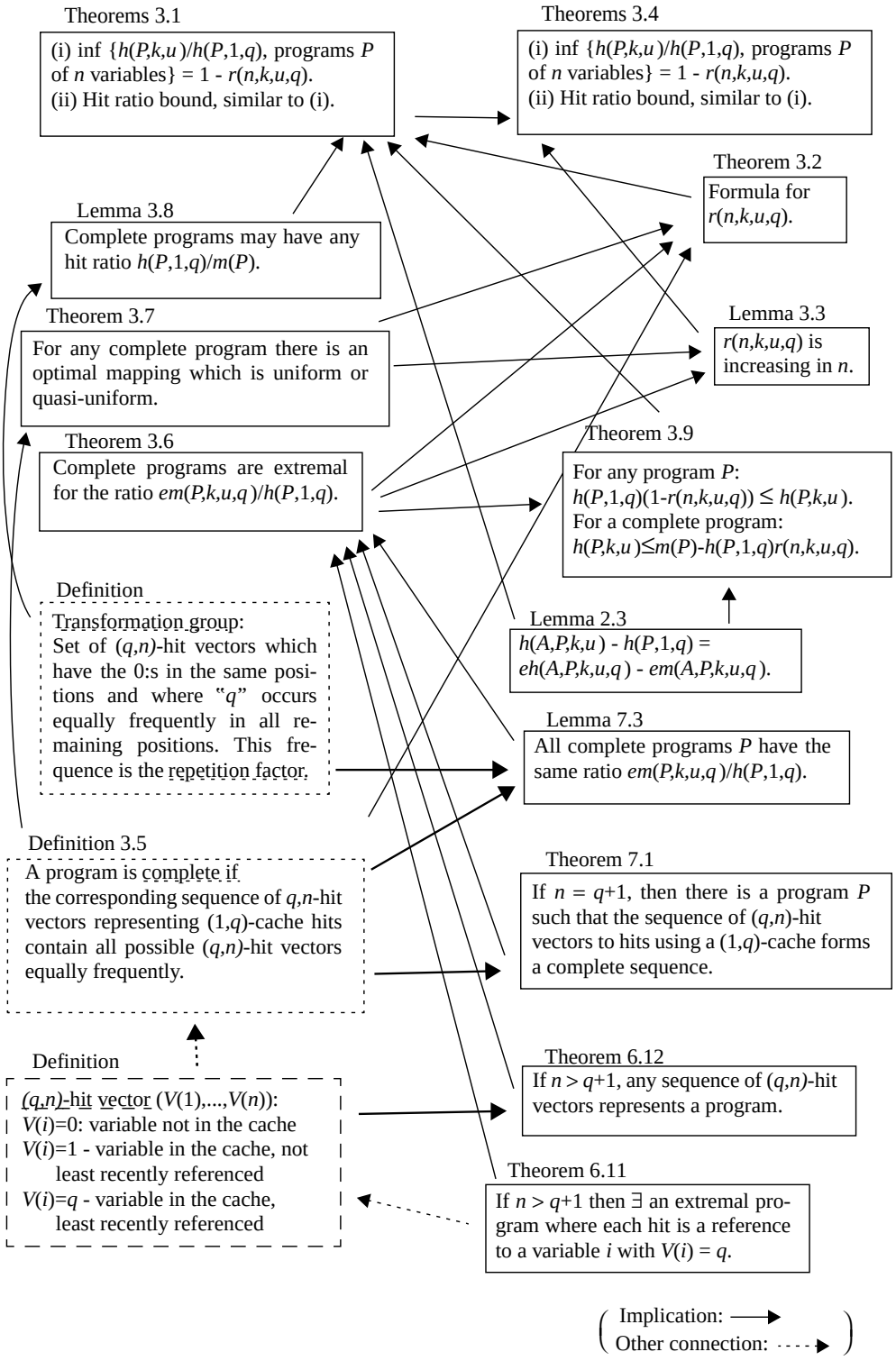


FIG. 2. *Logical structure outside section* 6 *(excluding the proofs in section* 9*).*

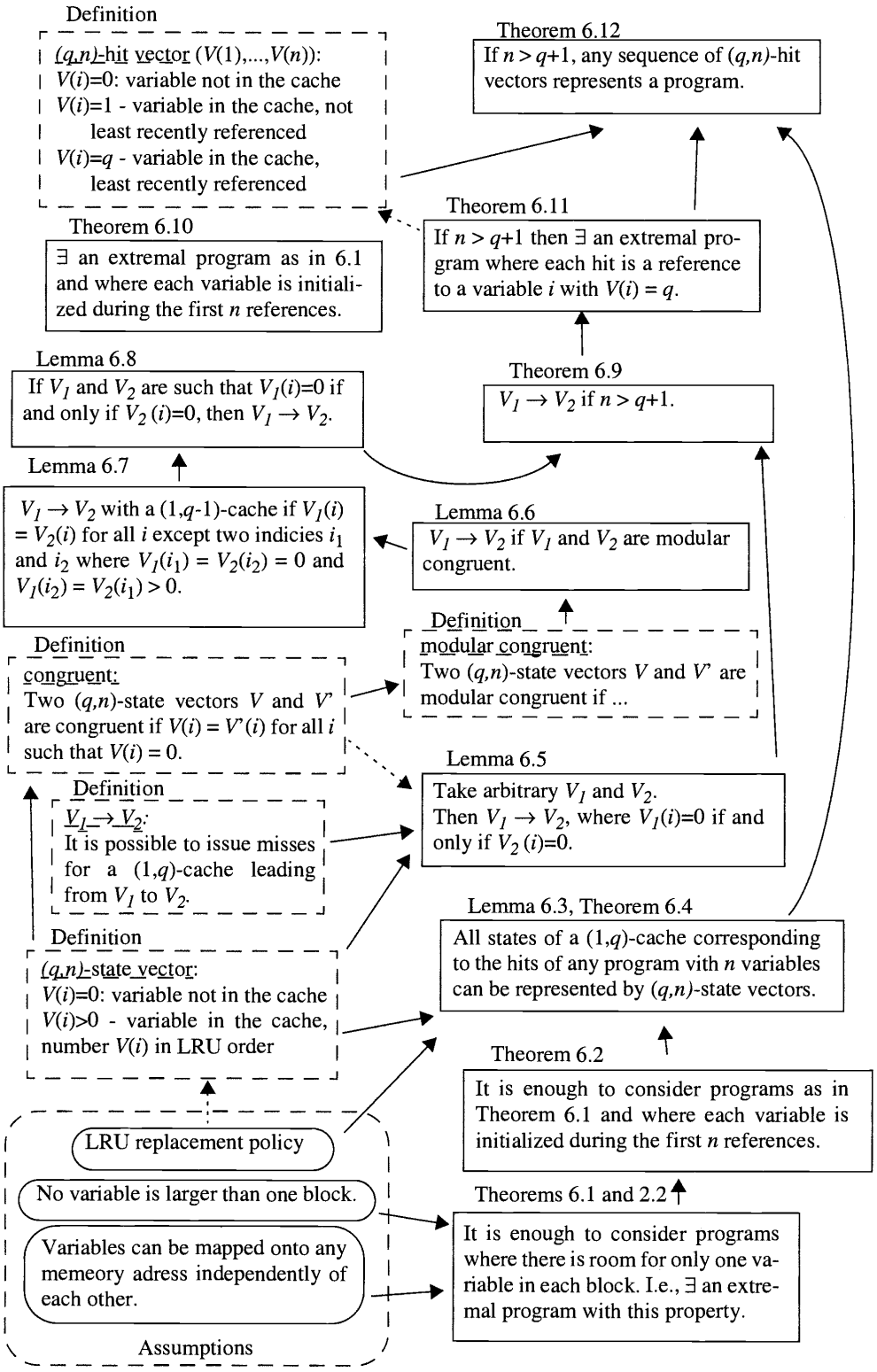


FIG. 3. *Logical structure of section* 6.

Proof. If $u \geq q$, then each set in the (k, u)-cache is larger than the set in the $(1, q)$-cache. Consequently, using the LRU replacement policy, no extra misses can occur. ∎

Having isolated the trivial cases above, we assume that $n > ku \geq q > u$ in the rest of this paper, unless otherwise stated. In addition, "extremal programs" will, unless otherwise stated, denote programs which maximize the quantity $em(P, k, u, q)/h(P, 1, q)$.

6. From programs to (q, n)-hit vectors.

THEOREM 6.1. *For any values of n, k, u, and q, there is at least one extremal program such that each variable occupies exactly one block.*

Proof. The only difference in the above statement compared to that in Theorem 2.2 is that here we wish to maximize $em(P, k, u, q)/h(P, 1, q)$, whereas in that proof we want to minimize $h(P, k, u)/h(P, 1, q)$. It is easily seen that the same proof also works for this theorem. ∎

There are thus extremal programs which have room for exactly one variable in each block. From now on we consider only such programs.

Since there are n variables in the program, we know that there will be at least n cache misses because each variable will be referenced at least once. The first reference to a variable, which always results in a miss, is referred to as the initial reference.

THEOREM 6.2. *Consider an arbitrary extremal program P and denote the first referenced variable by i_1, the second referenced variable by i_2, and so on. Then consider a program P' which is identical to P, with the exception that a sequence of n memory references to variables $i_1, i_2, \ldots, i_n$ in this order has been added at the start of the program. Then P' is also an extremal program.*

Proof. Since we are considering the case when $n > q$, the number of hits using a $(1, q)$-cache is the same for P and P'. That is, memory reference number r in P results in a miss using a $(1, q)$-cache if and only if memory reference number $r + n$ in P' results in a miss using a $(1, q)$-cache. Recall that we use the LRU replacement policy. Hence $h(P, 1, q) = h(P', 1, q)$.

Consider memory reference number r_2 in program P and denote the referenced variable by i. If r_2 results in a hit using a $(1, q)$-cache, then there is another reference r_1 which also accesses variable i, such that $r_1 < r_2$ and there are no references to variable i between r_1 and r_2. The sequence of memory references between r_1 and r_2 completely determines if r_2 will result in a miss using a (k, u)-cache, i.e., this sequence determines if an extra miss will occur. Obviously this sequence corresponds to the sequence $r_1 + n$ to $r_2 + n$ in P'. As a result, the number of extra misses is the same for P and P', i.e., $em(P, k, u, q)/h(P, 1, q) = em(P', k, u, q)/h(P', 1, q)$. Consequently, if P is an extremal program, P' is also an extremal program. ∎

Hence there are extremal programs which have no hits during the n first memory references using a $(1, q)$-cache. From now on we consider only such programs.

A (q, n)-state vector V defines the variables currently stored in a $(1, q)$-cache and the order in which they have been referenced. V has one entry for each variable. If variable i is not in the cache, then $V(i) = 0$. Otherwise $V(i)$ denotes the number of variables referenced since variable i was referenced last, including variable i. Hence if variable i is the most recently accessed variable, $V(i) = 1$. If it is the second most recently accessed variable, $V(i) = 2$, and so on (see Figure 4). Hence after the first n memory references, the (q, n)-state vector consists of the q first positive integers and $n - q$ zeros. As discussed before, the LRU replacement policy is used.

Clearly, a (q, n)-state vector has n entries and contains each of the first q positive integers exactly once, and $n - q$ zeros. If a reference to variable j is a cache miss, and

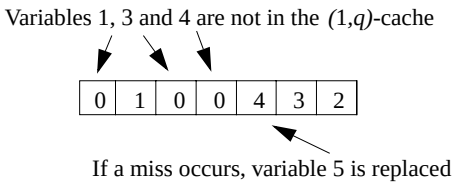


FIG. 4. *A (q,n)-state vector where $q=4$ and $n=7$.*

$V(i)=q$, then variable i is replaced in the cache. We then set $V(i)=0$, increment all positive integers, and set $V(j)=1$. If the reference to variable j is a hit, we increment all entries which are smaller than $V(j)$ and then set $V(j)=1$. Figure 4 shows a (q,n)-state vector for the case when $q=4$ and $n=7$.

We are interested in the ratio $em(P,k,u,q)/h(P,1,q)$ for extremal programs. When using a $(1,q)$-cache there are no hits during the first n memory references. Therefore, we do not consider the n first memory references, i.e., we assume that there are always q nonzero entries in a (q,n)-state vector. The numbering of the (q,n)-state vectors is chosen so that the (q,n)-state vector number r describes the state of the cache immediately before reference r in the program.

Now consider a mapping A of the n variables to k sets. Each mapping can be regarded as a partition of the first n positive integers in k sets A_j; mapping A is thus the sets $A_1,\ldots,A_k$.

LEMMA 6.3. *Consider a (k,u)-cache, a mapping A, and a (q,n)-state vector V corresponding to reference r in a program P. Assume that $i_0 \in A_\alpha$.*

Then the variable i_0 is not in the (k,u)-cache at reference r in program P if and only if there are j variables i_j, which are also mapped to A_α, such that $0<V(i_j)<V(i_0)$ and $j \geq u$.

Proof. This is a direct consequence of the definition of the (q,n)-state vector, Theorem 6.2, and of the fact that the LRU replacement policy is used. □

The next theorem states that a (q,n)-state vector contains enough information about the history of the program for our purposes.

THEOREM 6.4. *If the (q,n)-state vector after r references and the mapping of variables to sets are known, then it is possible to determine from this information whether reference r is an extra miss or not.*

Proof. Consider a (q,n)-state vector V, a mapping A, and a reference to a variable i_0. If $V(i_0)>0$, no miss will occur using a $(1,q)$-cache. This is a direct consequence of the definition of the (q,n)-state vector.

Assume that $i_0 \in A_\alpha$ using the (k,u)-cache and there are j variables i_j, which are also mapped onto A_α. In this case, the previous lemma guarantees that if $0<V(i_j)<V(i_a)$ and $j \geq u$, a cache miss will occur using the (k,u)-cache and mapping A.

Consequently, if the (q,n)-state vector and the mapping of variables to memory blocks are known, it is possible to determine if an extra miss will occur. □

Our next aim is to prove that any sequence of (q,n)-state vectors represents the sequence of $(1,q)$-hits for some program (Theorem 6.9). This allows us to consider arbitrary sequences of (q,n)-state vectors.

Two (q,n)-state vectors V and V' are said to be *congruent* if $V(i)=V'(i)$ for all i such that $V(i)=0$.

LEMMA 6.5. *For two arbitrary (q,n)-state vectors V_1 and V_2, it is possible to*

generate a sequence of memory references such that, starting from V_1, we reach a state vector which is congruent with V_2 without producing any hits using a $(1, q)$-cache.

Proof. Generate a reference to the first variable i (the variables are enumerated in some fixed arbitrary order) for which $V_1(i) = 0$ and $V_2(i) \neq 0$, thereby producing a new (q, n)-state vector V'. If V' and V_2 are not congruent, we again issue a reference to a variable i, where $V'(i) = 0$ and $V_2(i) \neq 0$. This procedure can be repeated until we reach a vector V' which is congruent with V_2.

For the sequence of memory references generated in this way there are no hits using a $(1, q)$-cache. □

Consider two congruent (q, n)-state vectors V and V'. The q indexes i_j for which $V(i_j) \neq 0$ are enumerated in such an order that $V(i_1) = 1, V(i_2) = 2, \ldots, V(i_q) = q$. V and V' are *modular congruent* if there is an integer x: $V(i_j) = ((V'(i_j)+x) \bmod q)+1$ for all $j : 1 \leq j \leq q$.

LEMMA 6.6. *For two modular congruent (q, n)-state vectors V_1 and V_2, it is possible to generate a sequence of memory references such that, starting from V_1, we reach V_2 without producing any hits using a $(1, q-1)$-cache.*

Proof. If we issue a reference to the variable $i_q : V_1(i_q) = q$, we will obtain a new state vector V_1' which is modular congruent to V_1. This reference will also clearly be a miss when using a $(1, q-1)$-cache. By repeating this kind of reference, we can generate any vector V which is modular congruent to V_1 without producing any hits when using a $(1, q-1)$-cache. □

Two (q, n)-state vectors V_1 and V_2 are *quasi-identical* if $V_1(i) = V_2(i)$ for all i except two indexes i_a and i_b so that $V_2(i_a) = 0, V_2(i_b) > 0, V_1(i_a) = V_2(i_b)$, and $V_1(i_b) = V_2(i_a)$.

LEMMA 6.7. *For two quasi-identical (q, n)-state vectors V_1 and V_2, it is possible to generate a sequence of memory references, such that, starting from V_1, we reach V_2 without producing any hits using a $(1, q-1)$-cache.*

Proof. Consider two state vectors V_1' and V_2' which are modular congruent with V_1 and V_2, respectively. V_1' and V_2' have been chosen so that $V_1'(i_b) = q$ and so that V_1' and V_2' are also quasi-identical. By issuing one reference to the variable i_a with $V_2'(i_a) = q$ we obtain V_2'' which is modular congruent to V_2'.

From Lemma 6.6 we know that it is possible to generate a sequence of memory references, such that, starting from V_1, it is possible to reach V_1' without producing any hits using a $(1, q-1)$-cache. From V_1' we obtain V_2'' by issuing a reference to variable i_a. Since $V_1'(i_a) = 0$, this reference leads to a miss using a $(1, q-1)$-cache. Again, Lemma 6.6 guarantees that, starting from V_2'', it is possible to generate a sequence of memory references such that we reach V_2 without producing any hits using a $(1, q-1)$-cache. □

LEMMA 6.8. *For two congruent (q, n)-state vectors V_1 and V_2, it is possible to generate a sequence of memory references so that, starting from V_1, we reach V_2 without producing any hits using a $(1, q-1)$-cache.*

Proof. Assume that $V_1(z) = V_2(z) = 0$, and that $V_2(i_j) = j$ for all $j : 1 \leq j \leq q$. In this case, the algorithm below reaches $V_1' = V_2$, starting from V_1, without generating any hit using a $(1, q-1)$-cache.

1. Let $j = q$ and let V_j' be modular congruent to V_1 such that $V_j'(i_q) = q$.
2. If $j = 1$, the algorithm terminates; else let $j = j - 1$.
3. If $V_{j+1}'(i_j) = j$, then $V_j' = V_{j+1}'$ and go to 2.
4. Go to state X_j, which is quasi-identical with V_{j+1}', i.e., $X_j(i) = V_{j+1}'(i)$ for all $i : 1 \leq i \leq n$ except that $X_j(i_j) = V_{j+1}'(z) = 0$ and $X_j(z) = V_{j+1}'(i_j) = t$

$(t < j)$.

5. Go to state X'_j, which is quasi-identical with X_j, i.e., $X'_j(i) = X_j(i)$ for all $i : 1 \leq i \leq n$ except that $X'_j(i_y) = Xj(i_j) = 0$ and $X'_j(i_j) = X_j(i_y) = j$, i.e., i_y is the index for which $X_j(i_y) = j$.
6. Go to state V'_j, which is quasi-identical with X'_j, i.e., $V'_j(i) = X'_j(i)$ for all $i : 1 \leq i \leq n$ except that $V'_j(z) = X'_j(i_y) = 0$ and $V'_j(i_y) = X'_j(z) = t$, n.b. V'_j is congruent with V_1 and V_2.
7. Go to 2.

The above algorithm guarantees that $V'_1(i) = V_2(i)$ for all $i : j \leq i \leq n$. Moreover, Lemma 6.7 guarantees that there will be no hits using a $(1, q-1)$-cache during steps 4, 5, and 6 in the above algorithm. □

THEOREM 6.9. *If $n > q+1$, it is possible to go from any (q,n)-state vector V_1 to any other (q,n)-state vector V_2 without producing any hits in a $(1,q)$-cache.*

Proof. This follows immediately from Lemmas 6.5 and 6.8. □

For the moment, we are considering only cases in which $n > q+1$. The case when $n = q+1$ will be handled later. As discussed previously, a program can be represented by the sequence of memory references, with associated (q,n)-state vectors, which corresponds to hits using a $(1,q)$-cache. From Theorem 6.9 we know that if $n > q+1$, each (q,n)-state vector in this sequence is completely independent of any other vector in the sequence.

THEOREM 6.10. *Consider a mapping A and two memory references r_1 and r_2, both referring to the same variable x. Denote the (q,n)-state vectors corresponding to r_1 and r_2 by V_1 and V_2, respectively. Assume that these vectors are such that $0 < V_1(x) = p < q, V_2(x) = q$, and $V_2(i) = V_1(i)$, when $V_1(i) < p$; otherwise $V_2(i) = V_1(i) - 1$ for all $i \neq x$. Assume also that reference r_1 is a miss using a (k,u)-cache and a mapping A. Then reference r_2 is also a miss using a (k,u)-cache and the mapping A.*

Proof. From Lemma 6.3 we know that if reference r_1 leads to a miss using a (k,u)-cache and mapping A, then there are i variables z_i, which are mapped to the same set as x, such that $0 < V(z_i) < V(x) = p$ and $i \geq u$. We also know that $V_2(i) = V_1(i)$, when $V_1(i) < p$. Consequently, at reference r_1 the same z_i variables are mapped to the same set as x. Under these conditions, Lemma 6.3 tells us that reference r_2 will result in a cache miss. □

THEOREM 6.11. *If $n > q+1$, there exists an extremal program such that each hit using a $(1,q)$-cache is a reference to the variable for which the corresponding entry in the (q,n)-state vector equals q.*

Proof. Consider a program P and its induced sequence of (q,n)-state vectors. If a reference r occurs to a variable i with $V(i) < q$, it is possible to use Theorem 6.9 to add memory references immediately before reference r so that $V'(i) = q$, where V' is the new (q,n)-state vector preceeding reference r. If this is done for all i such that $V(i) < q$, we obtain a new program P' which has the same number of hits using a $(1,q)$-cache: $h(P,1,q) = h(P',1,q)$. It remains to check that the number of extra misses does not decrease. Using Theorem 6.9 it is possible to add memory references in such a way that V' fulfills all conditions in Theorem 6.10, with no new hits. It then follows from Theorem 6.10 that all extra misses in P are also extra misses in P'. If sequences of memory references are added in this way at each spot where a reference r occurs to a variable i with $V(i) < q$, then from the program P we obtain a program P' where each hit is a reference to a variable i with $V(i) = q$, and $em(P,k,u,q)/h(P,1,q) = em(P',k,u,q)/h(P',1,q)$. □

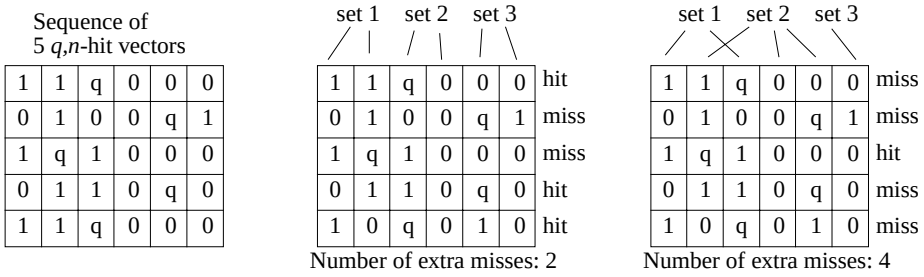


FIG. 5. *Two mappings of the same program with a* $(3,1)$*-cache, compared to a* $(1,3)$*-cache.*

From now on we will consider only programs for which each hit using a $(1,q)$-cache is a reference to the variable for which the corresponding entry in the current (q,n)-state vector equals q. There is at least one extremal program within this set. The ratio $em(P,k,u,q)/h(P,1,q)$ is, however, unchanged if all (q,n)-state vectors corresponding to misses using a $(1,q)$-cache are deleted—both quantities in the ratio deal with $(1,q)$-cache hits. It is clear that each arbitrary sequence of (q,n)-state vectors represents a program of this kind, with all misses removed. Hence we may look for extremal programs among the set of arbitrary sequences of (q,n)-state vectors.

Now consider a fixed mapping A, i.e., a partition of the variables in k sets consisting of u variables in each set. Our aim is to find how many of the (q,n)-state vectors result in a miss for a (k,u)-cache; this is the number of extra misses. Each reference to a variable i corresponds to an entry equaling q in the current (q,n)-state vector, $V(i)=q$. We know that the variable i is discarded using a (k,u)-cache if the number of variables $j : V(j) > 0$ belonging to this partition set is larger than u. This is a necessary and sufficient condition for an extra miss. Since we are thus only interested in whether a certain entry in a (q,n)-state vector is $0, q$, or some value between 0 and q, we can further simplify our notation. In a (q,n)-state vector "0," "1," and "q" are left unchanged, but all other integers are replaced by "1." As a result, each vector has one "q," $q-1$ "1," and $n-q$ "0." Such a vector is referred to as a (q,n)-hit vector. This vector also contains enough information to be able to decide whether an extra miss occurs or not.

In the following discussion we will often represent a program as a matrix which has (q,n)-hit vectors as rows. Again, only the hits are represented—all (q,n)-hit vectors representing misses with a $(1,q)$-cache are deleted.

Since a program has by definition $h(P,1,q)$ hits using a $(1,q)$-cache, we obtain an $h(P,1,q) \times n$ matrix. Figure 5 shows a program for which there are five cache hits using a $(1,3)$-cache, i.e., $h(P,1,q)=5$. The program has six variables. The first cache hit that occurs is a reference to variable number three. When this hit occurs, variables number one and two are also in the cache.

The example in Figure 5 shows that if the mapping of variables to sets is known, it is possible to determine if the memory reference corresponding to a certain (q,n)-hit vector will result in a miss using a (k,u)-cache or not. Remember, we are interested in the mapping of variables to sets resulting in a minimum number of misses using a (k,u)-cache. The figure also shows that the number of extra misses depends on the mapping.

The problem of calculating $r(n,k,u,q) = \max_P em(P,k,u,q)/h(P,1,q)$ for all programs P with n variables can now be formulated as the problem of finding a sequence of (q,n)-hit vectors for which the ratio obtained when dividing the minimum

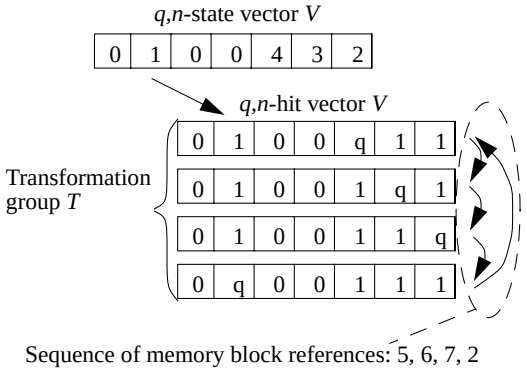


FIG. 6. *A transformation group T.*

number of misses for the sequence, when using a (k, u)-cache, with the length of the sequence is maximal.

The following theorem follows from Theorems 6.11 and 6.4. It summarizes the aim of the present section of reformulating the problem into arbitrary sequences of (q, n)-hit vectors.

THEOREM 6.12. *If $n > q + 1$, any sequence of (q, n)-hit vectors represents a program.*

It is obvious that any program with n variables corresponds to a sequence of (q, n)-hit vectors, as described before Definition 3.5.

7. Complete programs. There are $q\binom{n}{q}$ distinct (q, n)-hit vectors. A program for which the corresponding sequence of (q, n)-hit vectors contains exactly x copies of each distinct vector is referred to as a complete program; x is referred to as the *repetition factor*. Such a sequence of (q, n)-hit vectors is referred to as a complete sequence.

A set of q (q, n)-hit vectors forms a *transformation group* if all zeros are positioned at the same entries in all vectors and if the "q" is positioned differently in all vectors. Consequently, there are $x\binom{n}{q}$ transformation groups in a complete program.

Figure 6 shows a (q, n)-state vector V and the corresponding (q, n)-hit vector V. The figure also shows the transformation group T, in which V is a member. The arrows on the right side of the figure correspond to a sequence of four memory references, where the first reference is to a variable mapped to block 5, the second is to a variable mapped to block 6, the third to block 7, and the fourth to block 2. These four memory references go through all (q, n)-hit vectors in T exactly once, reaching (q, n)-hit vector V again after the fourth memory reference. With a $(1, 4)$-cache, none of the four memory references would result in a cache miss.

So far, we have considered only the case when $n > q + 1$. This is because if $n = q+1$, we cannot guarantee that there is a program corresponding to any arbitrary sequence of (q, n)-hit vectors (see Theorem 6.12). In the next theorem, however, we will show that there are complete programs even when $n = q + 1$.

THEOREM 7.1. *If $n = q + 1$, then there is a program P such that the sequence of (q, n)-hit vectors corresponding to hits using a $(1, q)$-cache forms a complete sequence.*

Proof. We are going to describe a complete program P with $n = q + 1$ variables.

The program P starts by generating the $n - 1$ memory references $2, 3, \ldots, q + 1$. This gives cache misses only, and the (q, n)-hit vector $V = (0, q, 1, \ldots, 1, 1)$. The

following q references are references to the variables $2, 3, \ldots, q+1$ in this order. By the definition of transformation groups we then clearly issue cache hits such that we go through all (q,n)-hit vectors in the transformation group T of V exactly once, returning finally to V.

Having done this we generate a memory reference to variable 1. This issue gives a miss and a new (q,n)-hit vector V' belonging to a different transformation group T', where $V'(1) = 0$. Again, we generate a sequence of q memory references, such that we go through all (q,n)-hit vectors in T' exactly once, finally returning to V' with no misses when using a $(1,q)$-cache. This procedure is repeated until we have produced all the $q+1$ different transformation groups.

Clearly the cache hits in the program P correspond to all possible (q,n)-hit vectors, and each (q,n)-hit vector occurs once. Consequently, P is a complete program for the case where x is one. Note that by repeating the $(q+1)^2$ memory references it is possible to obtain a complete program for any positive integer value of x. □

Proof of Lemma 3.8. Consider a complete program P for which the hits occur in such order that (q,n)-hit vectors belonging to the same transformation group are listed consecutively. If $h(P,1,q)/m(P) > H$, we add y cache misses at the end of the program, thus obtaining a new program P' which is also complete since only cache hits are involved in the definition of complete programs.

$$\frac{h(P',1,q)}{m(P')} = \frac{h(P,1,q)}{m(P)+y}.$$

If $h(P,1,q)/m(P) < H$, we consider another complete program P' where we have increased the repetition factor from x to x', still with (q,n)-hit vectors belonging to the same transformation group listed consecutively. Since only hits occur within a transformation group, we introduce no new misses. Let $\alpha = \frac{x'}{x}$. It is clear that by denoting the misses using a $(1,q)$-cache, $b(P) = m(P) - h(P,1,q)$, and $\alpha = \frac{x'}{x}$ we get

$$\frac{h(P',1,q)}{m(P')} = \frac{\alpha h(P,1,q)}{\alpha h(P,1,q) + b(P)}.$$

Hence we can increase the ratio so it is arbitrarily close to 1. By repeating the two techniques it is possible to find a complete program which has a hit ratio arbitrarily close to H. □

LEMMA 7.2. *Consider a complete program P. The variables in P include i_1 and i_2. If $A(i_1) = S_1$ (variable i_1 is mapped onto set S_1) and $A(i_2) = S_2$, and there is another mapping A' which is identical to A except $A'(i_1) = S_2$ and $A'(i_2) = S_1$, then the number of extra misses is the same for mappings A and A'.*

Proof. Obviously, the number of misses using a (k,u)-cache for vectors V where $V(i_1) = V(i_2)$ is the same for mappings A and A'. The symmetry of complete sequences guarantees that all (q,n)-hit vectors V in P, for which $V(i_2) \neq V(i_1)$, can be grouped into pairs (V,V'), such that $V(i) = V'(i)$, with the exception of $V(i_1) = V'(i_2)$ and $V(i_2) = V'(i_1)$. If and only if V results in a miss for mapping A using a (k,u)-cache, then V' results in a miss for mapping A' using a (k,u)-cache. Similarly, if and only if V results in a miss for mapping A' using a (k,u)-cache, then V' results in a miss for mapping A using a (k,u)-cache. Consequently, the number of extra misses is the same for A and A'. □

LEMMA 7.3. *All complete programs P with n variables have the same ratio $em(P,k,u,q)/h(P,1,q)$.*

Proof. Obviously, the ratio $em(P,k,u,q)/h(P,1,q)$ is not affected by the order in which the (q,n)-hit vectors are listed in the sequence corresponding to the complete program. The only potentially significant difference between two complete programs is thus the repetition factor x. Creating x copies of each row increases both the number of extra misses $em(P,k,u,q)$ and the number of hits using a $(1,q)$-cache $h(P,1,q)$ with the factor x. As a result, the ratio $em(P,k,u,q)/h(P,1,q)$ is not affected by x, i.e., all complete programs P with n variables have the same ratio $em(P,k,u,q)/h(P,1,q)$. □

Proof of Theorem 3.7. Lemma 7.3 and Theorem 3.6 allow us to consider a complete program of $q\binom{n}{q}$ rows. All references in one transformation group corresponding to a certain set will generate extra misses if the number of nonzeros in that set is larger than u; we otherwise obtain no extra misses. One transformation group will therefore contribute with i extra misses for each partition set where the number of nonzeros is $i > u$; the contribution is otherwise zero.

Consider two sets A_1 and A_2 in a mapping A, applied to a complete program P. The program may be represented by a matrix with $q\binom{n}{q}$ rows; the mapping may then also be referred to as a partition. The sets A_1 and A_2 are size n_1 and n_2, respectively, where $n_1 - n_2 \geq 2$. We will next move one variable from the larger set A_1 to the smaller set A_2, thereby producing the partition A'. If we denote the transferred column by c, we can write $A_1' = A_1 \backslash \{c\}$ and $A_2' = A_2 \cup \{c\}$. A' and A have all other partition sets in common.

We will next study under what conditions we have

$$em(A',P,k,u,q) \leq em(A,P,k,u,q).$$

If we replace A by A', the number of extra misses will not change at rows where the cth column contains a zero. However, at each row of P where we have 1 at column c, and we also have exactly u nonzeros in A_2 and do not have u nonzeros in the rest of A_1, the number of extra misses will increase by $u+1$. Here we have added the extra misses for all permutations in the same transformation group. Similarly, in a row where we have exactly u nonzeros in A_1' and do not have u nonzeros in A_2, the number of extra misses will decrease by $u+1$. Since P contains each permutation exactly once, with the aid of elementary combinatorics we may count these changes and we obtain

$$\begin{aligned} em(A',P,k,u,q) = {} & em(A,P,k,u,q) \\ & + (u+1)\left(\binom{n_2}{u}\binom{n-n_2-1}{q-u-1} - \binom{n_1-1}{u}\binom{n-n_1}{q-u-1}\right). \end{aligned}$$

Let $f(l) = \binom{l}{u}\binom{n-l-1}{q-u-1}$. So the contribution to the number of extra misses if one column is moved from A_1 to A_2, where $|A_1| = n_1$ and $|A_2| = n_2$, is

$$em(A',P,k,u,q) - em(A,P,k,u,q) = (u+1)(f(n_2) - f(n_1-1)).$$

We may now easily calculate for which l the function $f(l)$ is increasing or decreasing. We get

$$\begin{aligned} f(l) - f(l-1) &= \binom{l}{u}\binom{n-l-1}{q-u-1} + \binom{l-1}{u}\binom{n-l}{q-u-1} \\ &= \left(\frac{l}{l-u} - \frac{n-l}{n-l-q+u+1}\right)\binom{l-1}{u}\binom{n-l-1}{q-u-1} \end{aligned}$$

$$= \frac{l(n-l-q+u+1)-(l-u)(n-l)}{(l-u)(n-l-q+u+1)} \binom{l-1}{u} \binom{n-l-1}{q-u-1}$$
$$= \frac{-lq+l+un}{(l+1-u)(n-l-q+u)} \binom{l-1}{u} \binom{n-l-1}{q-u-1}.$$

We can conclude that $f(l)-f(l-1) \geq 0$ if and only if $l \leq \frac{un}{q-1}$. Since $q \leq ku$, we have $\frac{un}{q-1} \geq \frac{n}{k}\frac{q}{q-1} \geq \frac{n}{k}$. We therefore have a critical size $\frac{un}{q-1}$ for l which is higher than n/k. Of course, in a uniform partition, the partition sets have size $\lfloor n/k \rfloor$ and $\lceil n/k \rceil$.

Hence, if $n_2 < n_1 - 1 \leq \frac{un}{q-1}$, the number of extra misses will decrease if we move a column from a larger to a smaller set.

On the other hand, if $n_2 > n_1 - 1 \geq \frac{un}{q-1}$, the number of extra misses will decrease if, conversely, we move a column from a smaller to a larger set.

We will next repeat the argument by successively decreasing the number of extra misses by transferring columns between partition sets. We then finally obtain a uniform, or a quasi-uniform partition, which we may see as follows.

If all blocks in the starting partition are smaller than the critical size $\frac{un}{q-1}$, we apply the argument repeatedly to move columns from larger to smaller sets, thereby finally obtaining a uniform partition.

If one block is larger than the critical size, we apply the argument to pairs of the other blocks. We again move columns from larger to smaller sets, giving a quasi-uniform partition.

If the starting partition has two or more sets which are larger than the critical size, we apply the argument to a pair where both blocks are larger than the critical size. We then instead move columns from the smallest set to the largest. This argument is repeated for the same pair, until the smallest set is smaller than the critical size.

If there are still two or more blocks larger than the critical value, we apply this procedure to another such pair. Ultimately, we will have only one set which is larger than the critical size. This is handled as described above.

Consequently, our only candidates for optimal partitions are the uniform partitions, which have partition set sizes

$$\left(\left\lceil \frac{n}{k} \right\rceil, \dots, \left\lceil \frac{n}{k} \right\rceil, \left\lfloor \frac{n}{k} \right\rfloor, \dots, \left\lfloor \frac{n}{k} \right\rfloor \right),$$

together with the quasi-uniform partitions, defined by partition set size sequence

$$\left(i, \left\lceil \frac{n-i}{k-1} \right\rceil, \dots, \left\lceil \frac{n-i}{k-1} \right\rceil, \left\lfloor \frac{n-i}{k-1} \right\rfloor, \dots, \left\lfloor \frac{n-i}{k-1} \right\rfloor \right),$$

where $i = \max(\lceil n/k \rceil + 1, \lceil \frac{un}{q-1} \rceil), \dots, n$.

The quasi-uniform partitions have the size of one set larger or equal to the critical value $\frac{un}{q-1}$ but larger than $\lceil n/k \rceil$; while the other $k-1$ sets are as equally sized as possible. □

Numerical calculation shows that in most cases a uniform partition is optimal if the parameters are moderately large. For example, if all four parameters are at most 10, there are only three cases where a uniform partition is not optimal: $(n,k,u,q) = (9,7,1,7), (10,7,1,7)$, and $(10,8,1,8)$.

Proof of Theorem 3.6. Consider a sequence of (q,n)-hit vectors corresponding to an arbitrary program P_0 (see Figure 7), and take a mapping A_0 which is optimal for

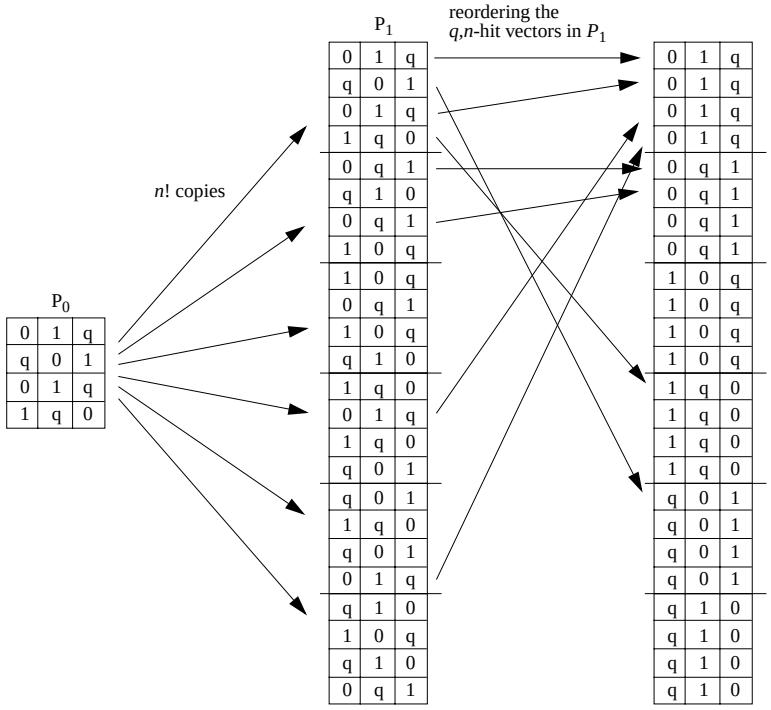


FIG. 7. *Creating a complete program P_1 from an arbitrary program P_0.*

the program P_0. Let A_1 be an optimal mapping for a complete program defined by n and q. We then clearly have

$$\frac{em(P_0,k,u,q)}{h(P_0,1,q)} = \frac{em(A_0,P_0,k,u,q)}{h(P_0,1,q)} \leq \frac{em(A_1,P_0,k,u,q)}{h(P_0,1,q)}.$$

As discussed previously, the (q,n)-hit vectors form a $h(P_0,1,q) \times n$ matrix; in the example in Figure 7 we have $h(P_0,1,q) = 4$ and $n = 3$.

We next duplicate the program P_0 in $n!$ copies, where each copy corresponds to one of the $n!$ permutations of the n columns in the $h(P_0,1,q) \times n$ matrix. The copies are concatenated, resulting in a sequence of (q,n)-hit vectors of the length $n!h(P_0,1,q)$. This sequence corresponds to a program P_1. It is clear that P_1 is a complete program (see the version of P_1 furthest to the right in Figure 7).

The mapping A_1 on each of the $n!$ permutations of the columns of P_0 can now be alternatively regarded as mappings $A_{1,i}, i = 1,\ldots,n!$, on the program P_0. Now fix the mapping A_1 to be such that $em(A_1,P_0,k,u,q) \leq em(A_{1,i},P_0,k,u,q)$ for all $i = 1,\ldots,n!$.

Then $em(A_1,P,k,u,q)$ is also a lower bound of the mean value of the quantities $em(A_{1,i},P_0,k,u,q)$. Since the sum of these is $em(A_1,P_1,k,u,q)$, we obtain

$$\begin{aligned}\frac{em(A_1,P_0,k,u,q)}{h(P_0,1,q)} &\leq \frac{1}{n!}\sum_{i=1}^{n!}\frac{em(A_{1,i},P_0,k,u,q)}{h(P_0,1,q)} \\ &= \frac{em(A_1,P_1,k,u,q)}{n!h(P_0,1,q)} = \frac{em(A_1,P_1,k,u,q)}{h(P_1,1,q)}.\end{aligned}$$

In addition, we know from Lemma 7.3 that all complete programs P with n variables have the same ratio $em(P,k,u,q)/h(P,1,q)$. From this we know that for an arbitrary program P_0,

$$\frac{em(P_0,k,u,q)}{h(P_0,1,q)} \le \frac{em(A_1,P_1,k,u,q)}{h(P_1,1,q)} = \frac{em(P_1,k,u,q)}{h(P_1,1,q)},$$

where P_1 is complete. Hence complete programs are extremal. □

8. Calculating $r(n,k,u,q)$.

Proof of Theorem 3.2. By Theorems 5.1, 5.2, and 5.3 we know that if $n \le ku$ or $u \ge q$, then $r(n,k,u,q) = 0$; and if $ku < q$ and $ku < n$, then $r(n,k,u,q) = 1$. In the remaining part of the proof we disregard such cases.

Consider a complete program P. We represent the program by a 0,1-matrix and assume no repetitions, i.e., P has $\binom{n}{q}$ rows. Each row in the matrix corresponds to a transformation group. We may reconstruct the transformation group by copying the row in q copies and replacing the ith nonzero with "q" in the ith copy for $i = 1,\dots,q$.

Now apply a partition A on P, and consider a specific row in P. The 1's which are in partition sets where there are $u+1$ 1's or more will result in extra misses. The cache set can only handle u variables, and the worst case program implies that the variables are referenced cyclically, resulting in only extra misses.

On the other hand, 1's which are in a set with u 1's or less will give no extra misses. Here the cache set can contain all the variables.

Assume that the ith partition set A_i of the partition A has w_i columns. How many rows (permutations) have j 1's in A_i?

Since P contains one copy of each possible permutation of q 1's in n positions, it is an easy combinatorial task to give an exact answer to this question. There are $\binom{w_i}{j}$ different ways of distributing these 1's in A_i, and there are $\binom{n-w_i}{q-j}$ different ways of distributing the other $q-j$ 1's in the remaining $n-w_i$ positions. We obtain extra misses only if $j \ge u+1$, and then we obtain j extra misses. The contribution to the number of extra misses from the ith set is thus

$$\sum_{j=u+1}^{\min(q,w_i)} j \binom{w_i}{j} \binom{n-w_i}{q-j}.$$

By adding the contributions from all sets, we get two alternative formulas for $r(n,k,u,q)$ with any partition A:

$$\begin{aligned} r(n,k,u,q) &= \frac{1}{q\binom{n}{q}} \min_A \left(\sum_{i=1}^{k} \sum_{j=u+1}^{\min(q,w_i)} j \binom{w_i}{j} \binom{n-w_i}{q-j} \right) \\ &= 1 - \max_A \frac{1}{q\binom{n}{q}} \sum_{i=1}^{k} \sum_{j=1}^{\min(u,w_i)} j \binom{w_i}{j} \binom{n-w_i}{q-j}. \end{aligned}$$

Essentially, in the first form we count the number of 1's which appear. In the second form, we count the number of 1's which do not appear and subtract this number from the total number of 1's, which is $q\binom{n}{q}$.

Using Theorem 3.7, we may invoke the set sizes of uniform and quasi-uniform partitions. Hence, by inserting in the above formula

$$(w_1,\dots,w_k) = \left(i, \left\lceil \frac{n-i}{k-1} \right\rceil, \dots, \left\lceil \frac{n-i}{k-1} \right\rceil, \left\lfloor \frac{n-i}{k-1} \right\rfloor, \dots, \left\lfloor \frac{n-i}{k-1} \right\rfloor \right),$$

where $i = \lceil n/k \rceil, \max(\lceil n/k \rceil + 1, \lceil \frac{un}{q-1} \rceil), \ldots, n$, we obtain the formula in Theorem 3.2. Since we have at most three set sizes, $i, \lceil \frac{n-i}{k-1} \rceil$, and $\lfloor \frac{n-i}{k-1} \rfloor$, we obtain only three terms in the sum. □

Proof of Theorem 3.4. Suppose that P_0 is a complete matrix and A_0 is an optimal mapping for P_0. Hence

$$r(n,k,u,q) = \max_{\text{programs } P \text{ with } n \text{ variables}} \frac{em(P,k,u,q)}{h(P,1,q)} = \frac{em(P_0,k,u,q)}{h(P_0,1,q)} = \frac{em(A_0,P_0,k,u,q)}{h(P_0,1,q)}.$$

We next construct a program P_1 by adding to program P_0 a new variable, of size exactly one block, as the very last reference. Since this variable is not referenced before, it is a miss using any cache. Hence we can obtain an optimal mapping A_1 to the program P_1 by adding the new variable to any set. The program P_1 has $n+1$ variables but may not be optimal. Thus

$$\frac{em(P_0,k,u,q)}{h(P_0,1,q)} = \frac{em(P_1,k,u,q)}{h(P_1,1,q)} \leq \max_{\text{programs } P \text{ with } n+1 \text{ variables}} \frac{em(P,k,u,q)}{h(P,1,q)} = r(n+1,k,u,q). \quad \square$$

9. Discussion. In this section we discuss the practical implications of the results as well as the assumptions on which they are based. Subsection 9.1 demonstrates how our results can be used for obtaining a bound on the worst case performance. In subsection 9.2 we discuss how the results can be used for comparing set-associative caches with direct-mapped caches, or other set-associative caches. In the remaining subsections we discuss some of the restrictions and assumptions in the definition of the problem. More specifically, in subsection 9.3 we discuss how replacement policies other than LRU would affect our results. Subsection 9.4 discusses the restriction that each variable must fit in one cache block. This discussion shows that the results obtained can be used also for programs with large variables. In subsection 9.5 we discuss large scientific programs which manipulate large matrices in different ways, e.g., matrix multiplication and addition.

9.1. Applying the results. The results reported here make it possible to obtain optimal worst case bounds on the cache hit ratio for set-associative and direct-mapped caches. This can best be understood by considering the following example.

Consider a program P, with n variables, for which we have obtained a certain hit ratio X using a $(1,q)$-cache. Now we are interested in the hit ratio Y for P using an optimal mapping of variables to memory addresses for a different cache organization—a (k,u)-cache—such that $q < ku$ and $u < q$. We know that $X \leq h(P,1,q)/m(P)$ and that $Y = h(P,k,u)/m(P)$. Consequently, we know from Theorem 3.1(ii) that $X(1 - r(n,k,u,q)) \leq Y$. One practical implication of this is that if we should obtain a hit ratio which is smaller than $X(1-r(n,k,u,q))$ using a (k,u)-cache, then we know that the mapping of variables to memory addresses is not optimal. Generally we do not know if $X = h(P,1,q)/m(P)$ or not, and Theorem 3.1(ii) gives an optimal lower bound on $h(P,k,u)/m(P)$. Consequently, based on the information we have the lower bound on Y is optimal.

In the above example the results were used for evaluating the performance (hit ratio) of one particular program P. However, it is also possible to use our results when

evaluating different cache architectures, e.g., $r(100, k, u, q)$ tells us how much worse a hit ratio we may obtain for a (k, u)-cache compared to a $(1, q)$-cache for programs with 100 variables or less (Lemma 3.3 shows that $r(n, k, u, q)$ is an increasing function of n). However, the results can also be used for more complex scenarios. Consider, for example, two cache memory systems: in system one we have a $(1, 20)$-cache, and in system two we have a $(20, 2)$-cache. The main memory access time is the same in both cases, and the access time for the $(1, 20)$-cache is eight times faster than a main memory access; the access time for the $(20, 2)$-cache is, on the other hand, ten times faster than a main memory access, i.e., the $(20, 2)$-cache is slightly faster than the $(1, 20)$-cache.

Let M be the access time for one memory reference on the main memory, and let C denote the access time on the cache memory. For a program P with length L and hit ratio H on system one we obtain the total access time (AT).

$$AT(P) = L(CH + M(1 - H)) = L(M - (M - C)H).$$

Let H' be the hit ratio with system two. From Theorem 3.1 we know that $H(P)(1 - r(n, k, u, q)) \leq H'(P) \leq 1$. Lower and upper bounds of the ratio $AT_2(P)/AT_1(P)$ when using caches with access times C_1 and C_2, respectively, follow:

$$\frac{C_2/C_1}{M/C_1 - (M/C_1 - 1)H} \leq \frac{AT_2(P)}{AT_1(P)} \leq \frac{M/C_1 - (M/C_1 - C_2/C_1)H(1 - r(100, k, u, q))}{M/C_1 - (M/C_1 - 1)H}.$$

In section 11.3 the bounds $f_1(H)$ and $f_2(H)$, respectively, are plotted as a function of the program parameter H when system one is a $(1, 20)$-cache with $M/C_1 = 8$, and system two is a $(20, 2)$-cache with $M/C_2 = 10$, i.e., $C_2/C_1 = 1.25$. Here we consider programs with 100 variables or less, and we thus need $r(100, 20, 2, 20) = 0.1652$. The functions $f_1(H)$ and $f_2(H)$ are interesting when we want to compare the two cache organizations.

9.2. Comparing set-associative and direct-mapped caches. Consider two set-associative cache memories. Cache M_1 is a (k, uc_1)-cache, and M_2 is a (kc_2, u)-cache. We assume that $c_1 \geq 1$ and $c_2 \geq 1$.

THEOREM 9.1.

$$\frac{h(P, kc_2, u)}{h(P, k, uc_1)} \geq 1 - r(nk, c_2, u, c_1)$$

for all programs P with n variables.

Proof. We first consider an arbitrary program P with n variables and an optimal mapping A_1 for M_1, i.e., we have $h(A_1, P, k, uc_1) = h(P, k, uc_1)$. We then construct another program P' with $n' = nk$ variables and a mapping A_1'. P' is identical to P with the exception that $n' - n$ references to the $n' - n$ new variables are added at the end of the program; one reference is added for each new variable. Mapping A_1 is used to define a mapping A_1'. Mapping A_1' is identical to A_1 for the n original variables. The remaining $n' - n$ variables are mapped in such a way that the number of variables in each set equals n. The mapping of the variables $n' - n$ does not affect the number of hits since the references to these variables are initial references, which are always cache misses according to LRU policy. As a result, $h(A, P, k, uc_1) = h(A', P', k, uc_1)$ using any mapping A, and $h(P, kc_2, u) = h(P', kc_2, u)$.

P' can be viewed as the sum of k interleaved programs P_i', $i = 1, \ldots, k$. A program P_i' contains the references to the variables in set i $(1 \leq i \leq k)$ in M_1 using mapping A_1'.

We then consider a mapping A_2 for M_2 such that variables which are mapped to set i in A_1' $(1 \le i \le k)$ are mapped to one of the c_2 sets $i + kj$ $(j = 0, \ldots, c_2 - 1)$ in A_2. For instance, if $k = 2$ and $c_2 = 2$, then all variables mapped to set zero in A_1' are mapped to sets zero or two in A_2, and all variables mapped to set one in A_1' are mapped to sets one or three in A_2.

The mapping A_2 defines, of course, a mapping of the variables in the set i in A_1', i.e., the variables of the program P_i'. Denote this submapping of A_2—the mapping of the variables of P_i' only—by $A_{2,i}$. Using this mapping, each of the k sets in M_1 can be treated as a fully associative cache for P_i' containing uc_1 blocks. This is compared to a (c_2, u)-cache. Using Theorem 3.1 and Lemma 3.3 we have

$$\frac{h(P_i', c_2, u)}{h(P_i', 1, uc_1)} \ge 1 - r(nk, c_2, u, uc_1)$$

for all programs P_i'.

For any two positive sequences $a_1, \ldots, a_n$ and $b_1, \ldots, b_n$ we have $\sum_i a_i / \sum_i b_i \ge \min_i a_i/b_i$. Hence,

$$\frac{\sum_{i=1}^{k} h(P_i', c_2, u)}{\sum_{i=1}^{k} h(P_i', 1, uc_1)} \ge 1 - r(nk, c_2, u, uc_1).$$

The subprograms P_i' were constructed from the optimal mapping A_1' for the program P'. It thus follows that

$$h(P', k, uc_1) = \sum_{i=1}^{k} h(P_i', 1, uc_1) = \sum_{i=1}^{k} h(A_1', P_i', 1, uc_1).$$

The mapping A_2 is not usually optimal; hence

$$h(P', kc_2, u) \ge h(A_2, P', kc_2, u) = \sum_i h(P_i', c_2, u).$$

It follows that

$$\frac{h(P, kc_2, u)}{h(P, k, uc_1)} \ge 1 - r(nk, c_2, u, uc_1)$$

for all programs P with n variables. □

9.3. Other replacement policies. In this paper we have used the LRU replacement policy throughout. In this subsection we will, however, show that some of the results can be applied to systems with other replacement policies.

Consider a fully associative cache, i.e., a $(1, q)$-cache, and a direct-mapped cache, i.e., a $(k, 1)$-cache.

THEOREM 9.2.

(i) *With any replacement policy we have*

$$\frac{h(P, k, 1)}{h(P, 1, q)} \ge 1 - r(n, k, 1, q)$$

for all programs P with n variables. We assume the same replacement policy for the $(1, q)$-cache and the $(k, 1)$-cache.

(ii) *Consider a program P with n variables and suppose that the hit ratio $h(P,1,q)/m(P)$ for P with a $(1,q)$-cache is known. For all replacement policies then*

$$\frac{h(P,k,1)}{m(P)} \geq (1 - r(n,k,1,q))\frac{h(P,1,q)}{m(P)}.$$

Proof. Consider a hit in the $(1,q)$-cache. We now temporarily redefine the (q,n)-hit vectors. We denote by q the variable which, according to the present replacement policy, is next to be excluded from the cache in the case of a cache miss. The variables marked with "1" are the remaining variables in the cache.

Now the same proofs can be used for showing that complete programs represent the worst case scenario when comparing a direct-mapped cache with a fully associative cache. We do not know, however, if it is possible to obtain a sequence of (q,n)-hit vectors corresponding to a complete program. Consequently, Theorem 9.2 follows from Theorem 3.1 and the fact that the complete program is still the worst case program when comparing a fully associative cache to a direct-mapped cache. In contrast to Theorem 3.1, however, the bound is not necessarily optimal. □

9.4. Large variables. In some cases, one variable may be too large to fit within one cache block. Examples of such variables are strings, records, and arrays which are accessed sequentially. (We will discuss matrices separately in the next subsection.) Accesses to such large variables consist of a number of clustered accesses to the different blocks in which the variable is contained.

THEOREM 9.3. *Consider programs P with n large variables, each consisting of v blocks. These blocks are mapped to consecutive memory addresses. Each access to one of these variables results in a sequence of accesses to the v blocks.*

(i)
$$\frac{h(P,k,u)}{h(P,1,q)} \geq 1 - r(n, \lfloor k/v \rfloor, u, \lceil q/v \rceil)$$

for all programs P with n variables, each variable consisting of v blocks.

(ii) *Suppose that the hit ratio $h(P,1,q)/m(P)$ for P with a $(1,q)$-cache is known. Then*

$$\frac{h(P,k,u)}{m(P)} \geq (1 - r(n, \lfloor k/v \rfloor, u, \lceil q/v \rceil))\frac{h(P,1,q)}{m(P)}.$$

Proof. Consider a $(1,q')$-cache, where $q' = v\lceil q/v \rceil$, and a (k',u)-cache, where $k' = v\lfloor k/v \rfloor$. Since $q' \geq q$ and $k' \leq k$, it is clear that $h(P,k,u)/h(P,1,q) \geq h(P,k',u)/h(P,1,q')$ for any program P. As a result, it is sufficient to show that

$$\frac{h(P,k',u)}{h(P,1,q')} \geq 1 - r(n, \lfloor k/v \rfloor, u, \lceil q/v \rceil) = 1 - r(n, k'/v, u, q'/v)$$

for all programs P with n variables with a size of v blocks each.

Consider the set of mappings for the $(1,q')$-cache where the first block in each variable is mapped to set i, where $i = 0, v, 2v, \ldots, k' - v - 1$. Let $h'(P,k',u)$ be the maximum number of cache hits for program P within this set of mappings. Obviously, $h'(P,k',u) \leq h(P,k',u)$. If we restrict ourselves to these mappings, we end up in a situation which is identical to having n one block variables and comparing a $(1,q'')$-cache to a (k'',u)-cache, where $q'' = q'/v = \lceil q/v \rceil$ and $k'' = k'/v = \lfloor k/v \rfloor$.

For such systems Theorem 3.1 guarantees the following.

(i) The worst case ratio of cache hits comparing two caches is given by

$$\inf_P \frac{h(P,k'',u)}{h(P,1,q')} = 1 - r(n,k'',u,q''),$$

where the infimum is taken over all programs P with n variables.

(ii) Consider a program P with n variables, and suppose that the hit ratio $h(P,1,q'')/m(P)$ for P with a $(1,q'')$-cache is known. Then an optimal estimate of the hit ratio for the program P using a (k'',u)-cache is given by

$$\inf_P \frac{h(P,k'',u)}{m(P)} = (1 - r(n,k'',u,q''))\frac{h(P,1,q'')}{m(P)}.$$

Consequently, Theorem 9.3 follows from Theorem 3.1 and the fact that

$$\begin{aligned} h(P,k,u)/h(P,1,q) &\geq h(P,k',u)/h(P,1,q') \\ &\geq h'(P,k',u)/h(P,1,q') = h(P,k'',u)/h(P,1,q'') \end{aligned}$$

for any program P with n variables consisting of v blocks. □

9.5. Scientific programs with large matrices. If the colliding variables are two elements in a large matrix, then the compiler can obviously not map them independently of each other. One scenario where these types of cache collisions occur are scientific applications, where large matrices are manipulated in different ways. In such programs, there are large loops that access rows, columns, diagonals, etc. Such loops contain long sequences of memory accesses for which the addresses are separated by a fixed distance called the *stride*. Some scientific applications are not well suited to cache memories, since the matrices are too large to fit into a cache. The bound is obviously valid (but not optimal) for such programs. In some cases, however, the access pattern can be adapted to cache memory systems, e.g., by dividing the computation into a number of steps and then accessing a limited part of the matrices in each step. In other cases, the matrix may be small enough to fit into the cache.

Consider a system with k cache sets of unit size and a program accessing a $b \times k$-matrix, i.e., a matrix with b rows and k columns, which is mapped row-by-row starting at memory address zero. This means that the ith element in each row is mapped to set i. As long as the program accesses each row sequentially, there are no collisions, i.e., there will be no cache misses after the first iteration through a row. If, however, the program accesses each column sequentially, there will be a collision on every access, i.e., for column j all memory accesses will be made to cache set j. As a result, there will be no cache hits when we iterate through a column.

It has been shown that these kinds of collisions can be handled by adding a number of dummy columns or, alternatively, dummy rows. By adding a dummy column in the example above, we can access a column sequentially with a minimum number of collisions [15]. Other access patterns, e.g., diagonals, can be handled in a similar way. However, for some combinations of access patterns some collisions always remain [15], i.e., if we eliminate the collisions for columns and diagonals, we may create conflicts for other access patterns.

We are interested in the minimum ratio $h(P,k,u)/h(P,1,q)$ for all programs P. It is clear that the minimum ratio occurs for programs with exactly q accesses in each loop. This is explained by the fact that, except for the initial misses, there will be no misses in the $(1,q)$-cache as long as the number of accesses in each iteration in the

loop is less than or equal to q. When the number of accesses in each loop iteration exceeds q there will be no hits at all using the $(1, q)$-cache. Clearly, the number of access conflicts per iteration is a nondecreasing function of the number of accesses in each loop iteration when using the (k, u)-cache.

By using the same kind of proof as in [15], we see that the worst case programs are the ones where all loops are equally long, and all strides occur equally frequently. For these kinds of programs we cannot improve the hit ratio for a (k, u)-cache by adding dummy columns. As a result, we know that the worst case program contains all combinations of the possible (bk, q)-hit vectors. (There are bk elements in the array, i.e., $n = bk$.) This is, in fact, the same kind of situation as for programs with any arbitrary access pattern and where we are free to map any variable to any address. However, in the case of matrices we obtain uniform partitions. Consequently, by disregarding the quasi-uniform partitions in the formula in Theorem 3.2 we obtain an optimal bound on programs containing large matrices. Disregarding quasi-uniform partitions has a very limited impact on $r(n, k, u, q)$. The limited possibility of mapping elements in an array independently of each other is thus almost completely compensated for by the fact that the array accesses must be separated by a fixed distance, i.e., by the stride.

Another, somewhat unusual, approach is to use nonlinear mapping functions. When ordinary linear mapping is used, every kth memory block is mapped to the same set. However, in some cases nonlinear mapping functions have been suggested [1, 18]. In such cases, the problem of optimally mapping variables to memory addresses is transformed into the problem of defining an optimal mapping function. In this case, different parts of large arrays can be mapped independently of each other.

10. Conclusions. The average behavior of set-associative caches has been studied extensively. One rule of thumb is that a $(k, 1)$-cache gives about the same hit ratio as a $(k/4, 2)$-cache [3]. On [3, p. 421] there is a table where the hit ratio using different cache sizes and different degrees of associativity is shown for a "typical" work load. A smaller version of the same table is also shown in [5]. These tables show that the hit ratio gain resulting from going from a $(k, 1)$-cache to a $(k/2, 2)$-cache is larger than the gain resulting from going from a $(k/2, 2)$-cache to a $(k/4, 4)$-cache; going from a $(k/4, 4)$-cache to a $(k/8, 8)$-cache results in an even smaller gain.

The plot in section 11.1 shows that the worst case scenarios behave rather differently to the average case scenarios. When $q << n$ (left side of the plot) we see that the maximum performance drop of using a $(k/2, 2)$-cache instead of a $(1, k)$-cache $(q = k)$ is smaller than the maximum performance drop of using a $(k, 1)$-cache instead of a $(1, k)$-cache. The plot also shows that the maximum performance drop of using a $(k/4, 4)$-cache instead of a $(1, k)$-cache $(q = k)$ is smaller than the maximum performance drop of using a $(k/2, 2)$-cache instead of $(1, k)$-cache. This is what one would intuitively expect. Quantifying the different worst case scenarios would not be possible, however, without the result presented here.

The plot in section 11.1 also shows that the worst case behavior is counter-intuitive (at least for us) and does not conform to the average case behavior when q is almost as large as n (right side of the plot). In this case the maximum performance drop of using a $(k/2, 2)$-cache instead of a $(1, k)$-cache $(q = k)$ is larger than the maximum performance drop of using a $(k, 1)$-cache instead of a $(1, k)$-cache and so on, i.e., the curves cross each other. In order to understand this we will consider the case when $n = 7$ and $q = k = 6$. In this case the worst case program corresponds to the following matrix of (q, n)-hit vectors.

q	1	1	1	1	1	0
1	q	1	1	1	1	0
1	1	q	1	1	1	0
1	1	1	q	1	1	0
1	1	1	1	q	1	0
1	1	1	1	1	q	0
q	1	1	1	1	0	1
1	q	1	1	1	0	1
1	1	q	1	1	0	1
1	1	1	q	1	0	1
1	1	1	1	q	0	1
1	1	1	1	1	0	q
q	1	1	1	0	1	1
1	q	1	1	0	1	1
1	1	q	1	0	1	1
1	1	1	q	0	1	1
1	1	1	1	0	q	1
1	1	1	1	0	1	q
q	1	1	0	1	1	1
1	q	1	0	1	1	1
1	1	q	0	1	1	1
1	1	1	0	q	1	1
1	1	1	0	1	q	1
1	1	1	0	1	1	q
q	1	0	1	1	1	1
1	q	0	1	1	1	1
1	1	0	q	1	1	1
1	1	0	1	q	1	1
1	1	0	1	1	q	1
1	1	0	1	1	1	q
q	0	1	1	1	1	1
1	0	q	1	1	1	1
1	0	1	q	1	1	1
1	0	1	1	q	1	1
1	0	1	1	1	q	1
1	0	1	1	1	1	q
0	q	1	1	1	1	1
0	1	q	1	1	1	1
0	1	1	q	1	1	1
0	1	1	1	q	1	1
0	1	1	1	1	q	1
0	1	1	1	1	1	q

Consider a $(6, 1)$-cache. The optimal mapping for this case is to map two variables to one set, and one variable to each of the five other sets. The number of extra misses in this case is 10. Then consider a $(3, 2)$-cache. The optimal mapping for this case is to map three variables to one set, and two variables to each of the other two sets. The number of extra misses in this case is 12. Consequently, the worst case program performs better on a $(6, 1)$-cache than on a $(3, 2)$-cache using the LRU replacement policy. In this case the worst case program is a program with seven variables and seven equally long loops, where each loop cyclically references six of the seven variables; each loop corresponds to one of seven possible ways of selecting six variables. One contribution of this study is that this kind of counter-intuitive behavior can be identified and indeed even quantified.

It is well known that one problem with direct-mapped (and set-associative) caches as compared to fully associative caches is that they exhibit an extremely poor worst case performance [4]. Without our results, however, it has not previously been possible to quantitatively compare the worst case behavior of direct-mapped and set-associative caches to fully associative caches. Section 11.2 displays a plot of this performance relation.

On [3, pp. 408–425] there is a very comprehensive discussion of the performance implications of various cache parameters, e.g., the LRU replacement policy outperforms random replacement, particularly for caches with relatively few blocks (up to a couple of thousand blocks). Moreover, the "optimal" block size increases with increasing cache size. Based on these observations it is reasonable to assume that large caches will have large blocks and that the LRU replacement policy is used within each set. Consequently, the LRU assumption made in this report is reasonable. The assumption that each variable can be mapped to one block becomes more reasonable for large cache blocks.

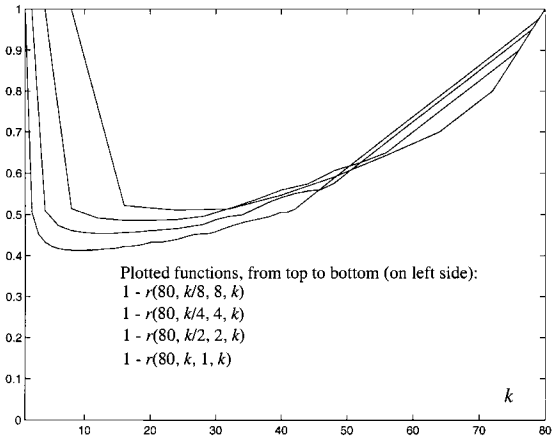
The discussion in section 9 shows that our results are applicable to a number of real scenarios. Moreover, the discussion in section 9 shows that a number of our assumptions with regard to the replacement policy, variable sizes, etc., can be relaxed, thus making the results applicable to a large number of cases.

11. Graphics section.

11.1. Increasing degree of associativity. This plot shows the worst case number of cache hits for four degrees of associativity, compared to full associativity. Here we compare only organizations with the same number of cache memory blocks. The worst case corresponds to the infinum

$$\inf_P \frac{h(P,k,u)}{h(P,1,q)} = 1 - r(80,k,u,q)$$

taken over programs of at most 80 variables.



Notes.

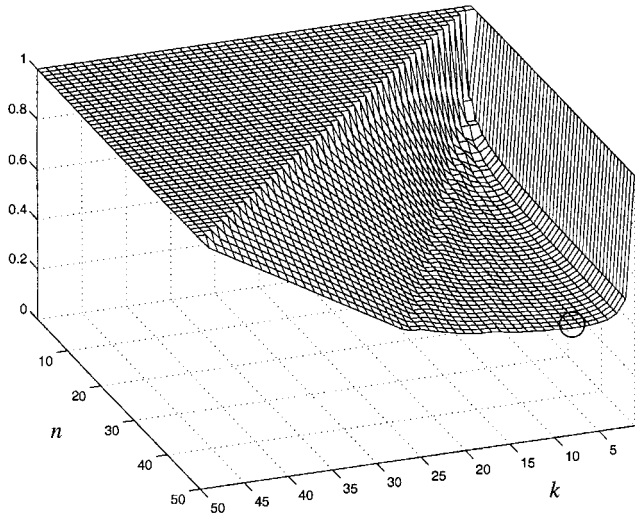
(1) The function r is defined only for integers—values representing noninteger arguments are plotted by linear interpolation.

(2) The improvement in a relative number of cache hits for worst case programs, as a function of the associativity, occurs fairly slowly. This is in accordance with [5]; see discussion in section 10.

(3) If the number of cache memory blocks (q) is close to the number of variables (n) (right side of the plot), less associativity can be favorable compared to more associativity with a constant number of cache memory blocks. In section 10 we describe programs which provide examples of this phenomenon.

11.2. Direct mapping versus full associativity. This plot shows the worst case hit ratio comparing a direct-mapped cache with a fully associative cache. The cache has k blocks, and programs of n variables are considered. The function plotted is

$$\inf_P \frac{h(P,k,1)}{h(P,1,k)} = 1 - r(n,k,1,k).$$

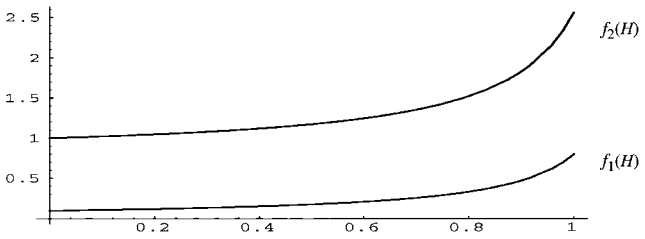


The minimum value for $1 - r(n,k,1,k)$ for n and k at most 50 is 0.4262 and occurs at $(n,k) = (50,8)$.

Suppose we want to compare a direct-mapped cache to a fully associative cache with equal number of cache blocks, and we are interested in the number of cache hits for programs with at most 50 variables. Then the ratio of cache hits in the two cases can vary more for different programs if we have eight memory blocks than any other number of cache blocks. If we have eight memory blocks in the cache, the ratio is in the worst case 0.4262.

11.3. Maximal and minimal access time. As described in section 9, estimates on cache hits can easily be transformed to estimates on access times. Consider a (20,2)-cache with cache access time 1/8; the main memory access time is normalized to 1. Denote the hit ratio for the program P on this system by H and the access time $A_1(H)$. Next the same program is to be executed on a (1,20)-cache with cache access time 1/10. Here optimal upper and lower bounds $f_1(H)$ and $f_2(H)$ for the access time $A_2(H)$ on the second system, relatively $A_1(H)$, are plotted.

$$f_1(H) = \frac{0.8}{8-7H} \le \frac{A_2(H)}{A_1(H)} \le \frac{8-5.635H}{8-7H} = f_2(H).$$



Notes.

(1) The lower bound is trivial, while the upper bound follows from Theorem 3.1. Both bounds are optimal, i.e., there are programs with any access time in the interval given by $f_1(H)$ and $f_2(H)$.

(2) Usually higher associativity results in slightly higher cache access time. If this relationship is known, i.e., if $C(k, u)$ is known, the above argument can be extended to provide optimal associativity in the sense of minimal access time for worst case programs.

REFERENCES

[1] F. Dahlgren and P. Stenström, *On reconfigurable on-chip data caches*, in Proceedings of the 24th Annual International Symposium on Microarchitecture, Albuquerque, NM, 1991, pp. 189–198.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, CA, 1990.

[4] M. D. Hill, *A case for direct-mapped caches*, IEEE Computer, 21 (1988), pp. 25–40.

[5] M. D. Hill and A. J. Smith, *Evaluating associativity in CPU caches*, IEEE Trans. Comput., 38 (1989), pp. 1612–1630.

[6] IBM Corporation, *IBM System/*370 *Model* 168 *Functional Characteristics* (GA22-7010), 3rd ed., IBM Systems Products Division, Poughkeepsie, NY, 1974.

[7] H. Lennerstad and L. Lundberg, *An optimal execution time estimate for static versus dynamic allocation in multiprocessor systems*, SIAM J. Comput., 24 (1995), pp. 751–764.

[8] H. Lennerstad and L. Lundberg, *Optimal combinatorial functions comparing multiprocess allocation performance in multiprocessor systems*, SIAM J. Comput., 29 (2000), pp. 1816–1838.

[9] H. Lennerstad and L. Lundberg, *Optimal scheduling results for parallel computing*, SIAM News, 27 (1994), pp. 16–18.

[10] L. Lundberg and H. Lennerstad, *An optimal upper bound on the minimal completion time in distributed supercomputing*, in Proceedings of the 8th ACM Conference on Supercomputing, Manchester, England, 1994, pp. 196–203.

[11] L. Lundberg and H. Lennerstad, *Bounding the gain of changing the number of memory modules in shared memory multiprocessors*, Nordic J. Comput., 4 (1997), pp. 233–258.

[12] L. Lundberg and H. Lennerstad, *An optimal lower bound on the maximal speedup in multiprocessors with clusters*, in Proceedings of the First International Conference on Algorithms and Architectures for Parallel Processing, Brisbane, Australia, 1995, pp. 640–649.

[13] L. Lundberg and H. Lennerstad, *Using recorded values for bounding the minimum completion time in multiprocessors*, IEEE Trans. Parallel and Distributed Systems, 9 (1998), pp. 346–358.

[14] L. Lundberg and H. Lennerstad, *An optimal bound on the gain of permitting dynamic allocation of communication channels in distributed processing*, in Proceedings of the International Conference on Principles of Distributed Systems, Picardie, France, 1997, pp. 29–44. An extended version of this paper is also published in Acta Informatica, 36 (1999), pp. 425–446.

[15] L. Lundberg and D. Haggander, *An optimal performance bound on the gain of optimizing data layout in vector processors*, in Proceedings of the ACM International Conference on Supercomputing, Melbourne, Australia, 1998, pp. 235–242.

[16] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, *Evaluation techniques for storage hierarchies*, IBM Systems J., 9 (1970), pp. 78–117.
[17] S. Przybylski, M. Horowitz, and J. Hennessy, *Performance tradeoffs in cache design*, in Proceedings of the International Symposium on Computer Architecture, Honolulu, HI, 1988.
[18] A. Seznec, *A case for two-way skewed-associative caches*, in Proceedings of the 20th Annual International Symposium on Computer Architecture, San Diego, CA, 1993.
[19] A. J. Smith, *A comparative study of set associative memory mapping algorithms and their use for cache and main memory*, IEEE Trans. Software Engineering, SE-4 (1978), pp. 121–130.
[20] A. J. Smith, *Cache memories*, ACM Computing Surveys, 14 (1982), pp. 473–530.
[21] A. Silberschatz and P. B. Galvin, *Operating Systems Concepts*, Addison-Wesley, Reading, MA, 1994.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 906–922

MARKOV PAGING*

ANNA R. KARLIN†, STEVEN J. PHILLIPS‡, AND PRABHAKAR RAGHAVAN§

Abstract. This paper considers the problem of paging under the assumption that the sequence of pages accessed is generated by a Markov chain. We use this model to study the fault-rate of paging algorithms. We first draw on the theory of Markov decision processes to characterize the paging algorithm that achieves optimal fault-rate on any Markov chain. Next, we address the problem of devising a paging strategy with low fault-rate for a given Markov chain. We show that a number of intuitive approaches fail. Our main result is a polynomial-time procedure that, on any Markov chain, will give a paging algorithm with fault-rate at most a constant times optimal. Our techniques show also that some algorithms that do poorly in practice fail in the Markov setting, despite known (good) performance guarantees when the requests are generated independently from a probability distribution.

Key words. competitive analysis, paging, Markov chains, locality of reference

AMS subject classification. 68

PII. S0097539794268042

1. Introduction.

This paper considers the problem of paging in a two-level store under the assumption that the sequence of pages accessed (henceforth called the *reference string*) is generated by a Markov chain. Each page of virtual memory is represented by a state (or node) of the Markov chain M, whose transition probabilities p_{ij} specify the probability that an access to i is immediately followed by an access to j.

Sleator and Tarjan [19] initiated the worst-case study of paging, introducing a style of worst-case analysis that has come to be known as *competitive analysis* [12]. We wish to address some shortcomings of the traditional adversarial analysis of paging [19]. Borodin et al. [2] provided an important step toward bridging the gap between such worst-case analysis and reality by providing a model that captured the essential aspects of locality of reference in the reference string. Here we follow their lead and assume that the reference string is generated by a Markov chain (thereby removing the adversary's role).

We refer to on-line paging with the reference string generated by such a Markov chain as *Markov paging*. We focus on questions such as the following: Given a Markov chain, what is the best paging algorithm for reference strings generated by it and how can this algorithm be computed? Is there a simple algorithm that performs near-optimally on every Markov chain? Can Markov paging explain why some paging algorithms perform poorly in practice?

Some salient features of our work are the following:

*Received by the editors April 15, 1994; accepted for publication (in revised form) May 31, 1997; published electronically August 9, 2000. A preliminary version of this paper appeared in the *Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, Pittsburgh, IEEE Press, Piscataway, NJ, 1992, pp. 208–217.

http://www.siam.org/journals/sicomp/30-3/26804.html

†Department of Computer Science and Engineering, University of Washington, Seattle, WA (karlin@cs.washington.edu).

‡AT&T Bell Laboratories, Murray Hill, NJ (phillips@research.att.com). This research was supported by NSF grant CCR-9010517 and grants from Mitsubishi and OTL. This work was done while the author was at Stanford University and DEC SRC.

§IBM Almaden Research Center, Almaden, CA 95120 (pragh@almaden.ibm.com).

- There is recent interest in the systems community in designing paging algorithms that adapt to the locality characteristics of a program [5, 15]. Thus insights derived from theoretical studies may have an impact on implementations.
- Markov paging, like the access graph model in [2, 10], offers a clean theoretical abstraction for locality of reference in a program. Unlike the models in [2, 10, 12, 19], there is no adversary generating the reference string, much as in a real program. Furthermore, certain simple properties of real programs—such as the fact that a data-dependent loop typically gets executed many times before exiting—can be modeled well.
- Practitioners study the *page-fault rate* for a program and a paging algorithm, rather than competitiveness. Page-fault rate has little meaning in an adversarial model but is eminently suited to a probabilistic model. In Markov paging, there is (as we shall see) a precise meaning to the page-fault rate of an algorithm as well as the best page-fault rate achievable on the Markov chain. Note that if each request is drawn independently from a probability distribution, the problem of devising the paging algorithm with lowest fault-rate has an easy solution [9].
- Markov paging enables us to provide a mathematical basis for the poor performance of certain paging algorithms (such as *random replacement* (RR) and *frequency count* (FC)). For example, the FC algorithm can be bad in the Markov paging setting, whereas one might think that on a probabilistic input this algorithm would perform well.
- Somewhat surprisingly, we find that several plausible approaches to devising paging algorithms will fail for Markov paging. For example, all *marking algorithms*, provably best in an adversarial model, suffer from page-fault rates that are far from optimal in Markov paging. This includes probabilistic versions of the FAR algorithm of [2, 10]. We also find that natural approximations of the optimal off-line paging algorithm MIN are far from optimal in Markov paging.
- We present an on-line paging algorithm that is computable in polynomial time and achieves a fault-rate within a constant times the best possible fault-rate on every Markov chain.

While our approach enables us to move away from competitive analysis toward a performance measure of greater interest to practitioners, two practical limitations of our work should be noted. First, in practical settings, page reference sequences may not be accurately modeled by a Markov chain in which pages are equated with chain states, since each page contains many memory locations. For example, a single page may have a number of basic blocks of code, and the next page to be referenced will depend on which basic block the program is currently in. Second, while the algorithm we present is polynomial-time computable, it requires the estimation and storage of commute times between pairs of pages of memory. The time and memory involved are considerable, so a simpler algorithm may be more applicable in practice.

1.1. Model. We have n pages that may be either in memory or on disk. Only k pages may be in memory at any time. An *on-line paging algorithm* is presented with a sequence of page requests. If the page currently requested is in memory, (a *hit*), no cost is incurred. However, if the page requested is not in memory (a *fault*), it must be fetched from the disk into memory for a unit cost. Furthermore, if there are already k pages in memory, one of the pages in memory must be evicted to make room for

the requested page. The decision of which page to evict must be made by the paging algorithm without detailed knowledge of future requests.

We now augment this basic model with a Markov source of references. Let M be an irreducible Markov chain whose state space is the set of n pages, and let A be an on-line paging algorithm. We define $f_A(M, k)$ to be the long-term frequency of faults incurred by A running on page request sequences generated by M and using a memory that can hold k pages. This is formalized in section 2. It is also shown in section 2 that there is an *optimal fault-rate* for every Markov chain M, which we denote by $f^*(M, k)$.

Note that we are not concerned here with the problem of inferring the Markov chain by observation of the page request sequence. We assume that the transition probabilities are already known. For example, they could be approximated to arbitrary accuracy by sampling a large enough initial prefix of the reference string.[1]

1.2. Related previous work. The underpinnings of competitive paging were laid in [12]. The literature of computer performance modeling and analysis contains related work, both theoretical and empirical. Denning [6] (and references therein) developed the *working set* model of program behavior for capturing locality of reference. Spirn [20] gives a comprehensive survey of models for program behavior. Franaszek and Wagner [9] studied a model in which every request is drawn independently from the same probability distribution. Shedler and Tung [18] and Lewis and Shedler [14] study paging in a Markov chain whose states represent *least recently used (LRU) stack distances*, a model convenient for studying the LRU paging algorithm. Denning and Spirn showed empirically that in order for a first-order Markov model to reasonably approximate real program behavior, it is necessary to separate data and instruction reference streams. Borodin et al. [2] and Irani, Karlin, and Phillips [10] have studied locality of reference in paging, though with an adversarial model.

Irani et al. [11] have studied other on-line problems when requests are independently drawn from a probability distribution. Vitter and Krishnan [21] considered the problem of prefetching into a cache when the reference string is generated by a Markov source (or mth-order Markov source). Under the assumption that there is always sufficient time to prefetch as many pages as wanted, Vitter and Krishnan show that data compression techniques can be used to obtain algorithms with optimal limiting page-fault rates.

1.3. Guided tour of the paper. Section 2 draws on the theory of Markov decision processes to characterize paging algorithms that achieve optimal fault-rate on any Markov chain. Theorem 2.1 shows that there is a memoryless, deterministic optimal on-line algorithm. Theorem 2.2 shows that there is a linear program that determines the optimal on-line algorithm, but this algorithm may have running time exponential in k. Section 3 shows that a number of plausible approaches for designing an efficient and provably good Markov paging algorithm will fail. Our main result, in section 4, gives an efficiently computable paging algorithm whose fault-rate is within a constant factor of the best possible on every Markov chain.

2. Markov decision theory and optimal paging. We begin by studying the relationship between page replacement policies and Markov decision theory [7].

A *Markov decision process* can be described as follows. Consider a discrete time Markov chain, whose state at time t is Y_t. After each observation of the system, one

[1] For this reason, the results we obtain assuming the Markov chain is known hold also in a limiting sense in the case where the Markov chain is not known.

of a set of possible actions is taken. Say that K_i is the set of actions possible when the system is in state i. Let A_t be the action taken at time t. A (possibly randomized) policy R is a set of functions $D_a(H_{t-1}, Y_t)$, where a is an action in K_{Y_t}, meaning that if H_{t-1} is the history of states and actions up to time $t-1$, and Y_t is the state at time t, then the probability action a is taken at time t is $D_a(H_{t-1}, Y_t)$. The actions can be such that they change the state of the system. We define this precisely by saying that $q_{ij}(a)$ is the probability of the system being in state j at the next instant, given that the system is in state i and action a is taken; i.e.,

$$Pr(Y_{t+1} = j \mid H_{t-1}, Y_t = i, A_t = a) = q_{ij}(a).$$

An additional set of parameters associated with a Markov decision process are costs: when the chain is in state i and action a is taken, a known cost w_{ia} is incurred.

Let $S_{R,T}$ be the expected cost of operating a system up to time T using policy R. ($S_{R,T} = \sum_{0 \le t \le T} \sum_j \sum_a P_R(Y_t = j, A_t = a) w_{ja}$.)

A standard cost criterion in Markov decision theory is to minimize the expected average cost per unit time, i.e., to find a policy R to minimize

$$\limsup_{T \to \infty} \frac{S_{R,T}}{T}. \tag{2.1}$$

We formalize the notion of expected faulted rate $f_A(M, k)$ of a paging algorithm A running on request sequences generated by the Markov chain M as follows. Let S be the state space of M (remember S is just the set of pages). Define the following augmented Markov chain whose state space is S'. A state in S' has two components: r and I. Here r is the most recent request, and I is a subset of S of size k representing the set of pages that are in memory immediately before servicing the request r.

When the system is in the state (r, I), and $r \notin I$, then there are k actions that can be taken by the on-line algorithm: for each $x \in I$, x can be evicted. The effect of the actions is described as follows. Suppose that p_{ij} are the transition probabilities of the underlying Markov chain. For each $r' \in S$,

$$Pr(Y_{t+1} = (r', I') \mid H_t, Y_t = (r, I), A_t = x) = p_{rr'},$$

where $I' = I \setminus \{x\} \cup \{r\}$.

When the system is in state (r, I) and $r \in I$, only the trivial action can be taken. The algorithm A is simply a policy for this Markov decision problem.

We set

$$w_{(r,I),a} = \begin{cases} 1 & \text{if } r \notin I, \\ 0 & \text{otherwise} \end{cases}$$

and define the expected fault rate $f_A(M, k)$ to be the expected average cost per unit time, as defined in (2.1) above.

The following two theorems instantiate basic results in Markov decision theory (see [7, Chapter 3, Theorem 2, and Chapter 6, Lemma 4]) to the case of Markov paging.

THEOREM 2.1. *For a given Markov chain there is an on-line page replacement policy that has minimum fault rate and is memoryless, time-invariant, and deterministic.*

Thus for the optimum algorithm, the decision of which page to evict on a fault depends only on the current request and the k pages in memory and is deterministic.

THEOREM 2.2. *The problem of computing the optimal on-line paging strategy on Markov chain M can be expressed as a linear programming problem in $n\binom{n}{k}$ variables.*

Note that this linear programming formulation gives us an algorithm for computing the optimal paging strategy on M whose running time may be exponential in k. We know of no technique to improve this upper bound, and this leads us to the study of efficiently computable strategies that approximate $f^*(M)$ well on every M. Note that when k is close to zero or to n, the linear program gives an efficient algorithm for computing the optimal on-line paging strategy. In fact, it is particularly instructive to consider the case $n = k + 1$, both for its own sake and because it has traditionally given good insights in paging [2, 8].

THEOREM 2.3. *When $n = k + 1$, for every M, there is an efficiently computable deterministic paging strategy that evicts only one of two fixed nodes ($k - 1$ pages are never evicted) whose fault-rate is at most $2f^*(M, k)$.*

Proof. Let G be the complete directed graph on n states, weighted by expected hitting times in M (the weight of edge (i, j) being $H_{i,j}$). For any memoryless, time-invariant, and deterministic algorithm, in any execution, the "hole" (the page that isn't in the memory) must eventually get into some cycle in G. Once in a cycle, the expected fault rate is just the reciprocal of the average edge weight around the cycle. Thus the optimal paging algorithm must use a *max mean cycle*. (See [13] for an algorithm to find a max mean cycle.) It follows that there is a cycle of length 2 that has page-fault rate at most twice optimal: just pick a pair (i, j) of adjacent vertices on a max mean cycle such that $H_{i,j}$ is at least the mean. Note that in the case that the chain is reversible, the mean hitting time around a cycle is independent of direction around the cycle, so there must be a cycle of length 2 with optimal page-fault rate. □

The following theorem compares the optimal on-line fault-rate to the optimal off-line fault-rate.

THEOREM 2.4. *The expected page-fault rate of the optimal on-line algorithm (OPT) is at most $O(\log k)$ times the expected page-fault rate of the optimal off-line algorithm (that sees the entire request string output by the Markov chain, then serves it optimally).*

This follows from the existence of an $O(\log k)$-competitive randomized paging algorithm [8], combined with the von Neumann minimax principle [16]. The bound is the best possible: when the Markov chain is the random walk on the $k + 1$ node complete graph K_{k+1}, any on-line algorithm has fault-rate $\Omega(\log k)$ times that of the optimal off-line algorithm [16].

3. Negative results. We begin this section by establishing that two algorithms that have been proposed and found to fare poorly in practice are far from optimal in Markov paging. Following this, we begin our quest for a simple, efficiently computable paging algorithm that has page-fault rate within a constant multiple of the best possible page-fault rate on every Markov chain. We show in this section that a number of intuitively "obvious" algorithms for Markov paging fail to achieve this goal and pave the way for our optimal algorithm in section 4. In all our negative results, $n = k + 1$; however, this is for simplicity of presentation only and can be generalized.

Remember that the hitting time $h_{x,y}$ from a state x in a Markov chain to another state y is defined to be the expected number of steps taken to first reach y starting from x, while the commute time $C_{x,y}$ is the expected number of steps to reach y from x and then return to x. We mention also that the commute time is related to the resistance [4], and thus the hitting times in most examples used here can be easily

computed by the electrically literate reader.

The RR paging algorithm will, on a fault, evict a random page in memory. Although optimal in the competitive setting against an adaptive on-line adversary [17], it performs relatively poorly in practice.

THEOREM 3.1. *There is a constant c such that for every $k > 2$, there is a Markov chain M for which $f_{RR}(M,k) \geq ckf^*(M,k)$.*

DEFINITION 3.2. *For positive integers a, b, the* lollipop graph $L(a,b)$ *is formed by attaching one end of a path through b nodes to a complete graph K_a on a nodes.*

Proof of Theorem 3.1. Let $n = k+1$, and let M be the Markov chain representing the simple random walk on $L(k-1,2)$. The maximum hitting time is $\Omega(k^2)$, say between nodes x and y. The algorithm that alternates its hole between x and y (as in Theorem 2.3) thus has a fault rate of $O(1/k^2)$, since the expected fault rate is $1/C_{x,y}$ by a result of renewal theory (see, for example, [7, p. 147]). Therefore $f^*(M,k)$ is $O(1/k^2)$. On the other hand, the expected time between faults incurred by RR is the average over all pairs of nodes x, y in the graph of the hitting time from x to y. This average is $O(k)$. □

The FC algorithm maintains, for each of the n pages, a count of the number of times that page has been accessed. On a fault, it evicts the page in memory that has been accessed least often. When every request is drawn independently from a probability distribution, FC converges to the optimal algorithm. However, it performs poorly in practice, since it ignores locality of reference. This is reflected by the fact that it is far from optimal in Markov paging (Theorem 3.3).

THEOREM 3.3. *There is a constant c such that for every $k > 2$, there is a Markov chain M for which $f_{FC}(M,k) \geq ckf^*(M,k)$.*

DEFINITION 3.4. *For positive integers a, b, the* forked lollipop graph $FL(a,b)$ *is formed from $L(a,b-1)$ by connecting two new nodes to the external end-node of the $(b-1)$-path.*

Proof of Theorem 3.3. Let $n = k+1$, and let M be the Markov chain representing the simple random walk on $FL(k/2,k/2)$. The two prongs have lowest stationary probability on this chain, so will eventually have the smallest frequency count in any request sequence. Thus FC will eventually alternate the "hole" (the node not in memory) between these two nodes. The expected time between faults incurred by FC is $O(k^2)$ (which is the expected hitting time from one prong to the other), whereas $f^*(M,k)$ is $O(1/k^3)$ (obtained by alternating the hole between one prong and a node in the clique). □

Next, we show that many algorithms that intuitively should perform well in Markov paging will in fact perform poorly on some Markov chains.

A class of on-line algorithms that one may expect to perform well are *marking algorithms*. Marking algorithms use a notion of *phases*. A *new page* is one that wasn't requested in the previous phase. A new phase begins with a request to a new page. When a page is requested, it is marked. As soon as k distinct pages are marked, the phase ends, and all pages become unmarked. A marking algorithm has the property that it never evicts a marked page.

It has been shown that there are optimal marking algorithms under competitive analysis for paging with locality of reference [2] as well as randomized paging algorithms (arbitrary request sequences) [8, 10]. Therefore, one might think that the same would hold when reference strings are generated from a Markov chain. The following theorem shows that our search for a good algorithm should exclude marking algorithms. The lower bound in the theorem cannot be improved, since there

is a marking algorithm A which is $O(\log k)$-competitive [8], and therefore there is a constant c such that $f_A(M,k) \le c(\log k) f^*(M,k)$ for any Markov chain M.

THEOREM 3.5. *There is a constant c such that for any $k > 2$, there is a Markov chain M for which $f_A(M,k) \ge c(\log k) f^*(M,k)$ for any marking algorithm A.*

Proof. Let M be the Markov chain corresponding to a simple random walk on the lollipop graph $L(k/2+1, k/2)$. Let y denote the node at the end of the path. A phase starts with a request at some vertex of the graph and ends just before all nodes in the graph have been requested. The last node to be requested starts a new phase. The important property of this Markov chain that defeats any marking algorithm is that once a node in the clique is requested, with high probability all the nodes in the clique will be requested before the node y is requested. Hence on average, almost half the phases will begin on y. By a standard argument, if a phase begins on y, any marking algorithm will incur an expected $\Omega(\log k)$ faults in the clique before the phase ends. Thus any marking algorithm will incur $\Omega(\log k)$ faults per phase on average. On the other hand, the on-line algorithm that alternately evicts a node in the clique and y will incur an average of one fault per phase. □

The optimal off-line algorithm for any reference string is commonly called MIN. MIN always replaces the page that will be requested furthest in the future. We now consider various on-line algorithms that mimic MIN on a Markov chain. The *maximum hitting time* (*MHT*) algorithm replaces, on a fault, that page in memory for which the *expected time* to the next request is the largest. Indeed, when the requests are drawn independently from a probability distribution, this algorithm performs well [9]; again, the locality of reference captured by Markov paging proves to be the undoing of this algorithm.

THEOREM 3.6. *For every $k > 10$, there is a Markov chain M on $k+1$ nodes and a constant c such that $f_{MHT}(M,k) \ge ckf^*(M,k)$.*

Proof. Consider the forked lollipop $G = FL(2k/3, k/3)$. Suppose that the Markov chain generating the reference string is the simple random walk on G.

If *MHT* is run on the reference string generated by this Markov chain, eventually one of the two prongs at the end of the path will be evicted. From either prong, the *MHT* is $4n^2/9$ to the other. On the other hand, the *MHT* from a prong to any node in the clique is $2n^2/9 + O(n)$. Thus the page not in memory (the "hole") will thereafter oscillate forever between the two prongs, so that $f_{MHT}(M,k)$ is $\Omega(1/k^2)$. The optimal algorithm will alternately evict a node in the clique and a prong with fault-rate $O(1/k^3)$. □

Let *LAST* be the algorithm that on a fault evicts the page that has the highest probability of being the last of the k pages in memory to be requested. An attractive variation on *LAST* is an algorithm we call max rank (*MR*) defined as follows. Suppose that at the time of a fault, S is the set of pages in memory. Then there is some permutation on S that describes the order in which these pages will subsequently be seen. For each page $i \in S$, and $1 \le j \le k$, let $p_i(j)$ be the probability that page i is the jth page in S that will be seen. Define the expected rank of page i, R_i to be $\sum_j j p_i(j)$. Then *MR* evicts the page $p \in S$ such that R_p is maximum.

THEOREM 3.7. *There is a constant c such that for any k* (i) *there is a Markov chain M such that $f_{LAST}(M,k) \ge ckf^*(M,k)$ and* (ii) *there is a Markov chain M such that $f_{MR}(M,k) \ge ckf^*(M,k)$.*

Proof. (i) Consider the Markov chain corresponding to the standard random walk on an undirected $k+1$ node cycle: all nodes are equally likely to be visited last. The following proof of this fact is credited by Broder [3], without further reference, to

Avrim Blum, Ernesto Ramos, and Jim Saxe, independently. Consider a point a on the cycle. Let its neighbors be b and c. Before visiting a, the walk will first visit one of its neighbors, say b. Given this fact, the probability that a is last is the probability that the walk will visit c, starting from b, before it visits a. This is clearly independent of the position of a.

Consequently, on a fault, $LAST$ might always evict the neighbor of the faulted node. Therefore, $LAST$ can have expected fault-rate $O(1/k)$ (since the expected time to hit a neighbor in a cycle is $O(k)$). On the other hand, the algorithm which alternates the hole between two antipodal points has an expected fault-rate of $O(1/k^2)$.

(ii) Consider a directed cycle on k nodes, with an extra node z that has edges to and from two antipodal nodes x and y. Let $p_{xz} = p_{yz} = p = 5/k$, and let $p_{zx} = p_{zy} = 1/2$. We show that starting from any node w on the cycle, the node of maximum rank is the node w^- that precedes w on the cycle. Indeed, the probability of avoiding half the cycle before hitting w^- by going through z is $p/2$, since the walk from w must reach z at the first chance (probability p), then can stay within $\{x, y, z\}$ for a while. The last time it leaves z before venturing outside $\{x, y, z\}$, it must move to whichever of x and y is closer to w^- (probability 1/2). The probability of avoiding half the cycle before hitting z is just p. Hence $R_{w^-} \geq k - 1 - \frac{p}{2}(\frac{k}{2} - 1)$, while $R_z \leq k - p(\frac{k}{2} - 1)$, and the former is larger for $k > 20$.

In contrast, the optimal on-line algorithm will evict z whenever there is a fault on the cycle. In this case, MR incurs $\Omega(k)$ times as many faults as the optimal on-line algorithm. □

Finally, an algorithm which is very close in spirit to our nearly optimal on-line algorithm of section 4 is the maximum commute time (MCT) algorithm. On a fault for page v, MCT evicts the page w in memory that maximizes the *commute time between v and w*.

THEOREM 3.8. *For every k, there is a Markov chain M for which $f_{MCT}(M, k) \geq kf^*(M, k)$.*

Proof. Consider the Markov chain corresponding to a directed cycle $k + 1$ nodes. Then every pair of nodes in the graph has the same commute time $(k+1)$. Therefore, on a fault at some node v, MCT might always evict the successor of v, incurring a fault on every request. Since the optimal on-line algorithm has a fault rate of $1/k$, the claim is proven. □

4. A provably good algorithm.

4.1. Description of the algorithm. The commute algorithm (CA) operates in phases. It keeps a window of the last $k + i$ requested pages, for some $0 \leq i \leq k$. (We assume $n \geq 2k$. For the case $n \leq 2k$, a simpler algorithm in the same spirit can be shown to have page-fault rate that is within a constant factor of optimal.) At the beginning of a phase the window is just the k most recently requested pages; these pages are resident in memory. When a "new" page p (one that hasn't been requested in the current or last phase) is requested, it is added to the window. The phase ends (and the window shrinks back to size k) when the $(k+1)$st distinct page is requested in the current phase. At that time, CA performs the minimum number of swaps necessary to ensure that the k most recently requested pages are again resident in memory, and the window again becomes the k most recently requested pages.

When the window contains $k + i$ pages, CA maintains a partial matching of i disjoint pairs of pages $\{(u_1, v_1), \ldots, (u_i, v_i)\}$. The commute algorithm maintains the following invariant:

For each j, $1 \le j \le i$, exactly one of u_j and v_j is in memory.

On a fault at u_j the page v_j is evicted, and vice versa. (Observe that *CA* is not a marking algorithm.) When a new page p is added to the window, it is paired with a page q that is in the window but not in the matching such that the commute time $C_{p,q}$ is maximized. The new pair is added to the partial matching, and may be involved in a single "switch," described below.

To describe the notion of a switch, we need some notation. For states a, b, u, v, we define the *relative distance* of the pair (a, b) to u to be

$$d[(a,b),u] = \frac{\min\{C_{a,u}, C_{b,u}\}}{C_{a,b}}.$$

Define also the relative distance of (a, b) to (u, v) to be

$$d[(a,b),(u,v)] = \min\{d[(a,b),u], d[(a,b),v]\} = \frac{\min\{C_{a,u}, C_{b,u}, C_{a,v}, C_{b,v}\}}{C_{a,b}}.$$

Intuitively, if $d[(a, b), u]$ is large, then both a and b are much further (in the commute time metric) from u than they are from each other. Similarly, if $d[(a, b), (u, v)]$ is large, then both a and b are much further from both u and v than they are from each other.

When a new pair (p, q) is added to the matching, *CA* does the following:

Case 1. If $d[(p, q), (u_j, v_j)] \le c_s$ for all j, then there is no switch: we service the fault at p by evicting q. Here c_s, the "constant for switching," is a suitably chosen constant; the reader can verify during the proof below that by choosing $c_s = 2$, all inequalities involving c_s hold.

Case 2 (switch). Otherwise, choose j so that $d[(p, q), (u_j, v_j)]$ is maximized, and replace the matched pairs (p, q) and (u_j, v_j) by (u_j, p) and (q, v_j). Service the fault at p by evicting whichever of u_j or v_j is in memory. This restores the invariant mentioned above.

In the upcoming proofs, we will distinguish one node in each pair, and therefore we may need to reverse the roles of u_ℓ and v_ℓ for some pairs, whether or not a switch has been done. The analysis below shows how this should be done.

As for the running time of *CA*, the complexity of computing the commute times in a Markov chain is polynomial in n. Therefore, with an initial preprocessing step that constructs the matrix of commute times, the complexity of running the algorithm *CA* is $O(k)$ per fault.

4.2. Analysis of the algorithm.

THEOREM 4.1. *There is a constant c such that for any Markov chain M and any k,*

$$f_{CA}(M,k) \le c f^*(M,k).$$

We prove the theorem by establishing a strict relationship between the pairs in the matching maintained by *CA*.

Consider one phase of the algorithm: let the rth subphase be the time when the window size is $k + r$. Let M_r and W_r be the matching and the set of pages in the window, respectively, during the rth subphase. Last, let V_{M_r} be the pages involved in M_r. Notice that $W_1 \subset W_2 \subset \cdots \subset W_k$ and $V_{M_1} \subset V_{M_2} \subset \cdots \subset V_{M_k}$. We will refer to one node in each pair (u_i, v_i) in M_r as *distinguished*. Without loss of generality the distinguished node will always be u_i.

We say that M_r is a *good* matching if the following hold:

1. For all pairs (u_i, v_i) and $(u_j, v_j) \in M_r$, $i \neq j$, $d[(u_i, v_i), u_j] \leq c_d$. Here c_d is the maximum allowable distance to distinguished nodes. The fact that u_j is involved in this condition, rather than v_j, is what makes u_j distinguished.
2. For each pair $(u_i, v_i) \in M_r$ and $a \in W_r \setminus V_{M_r}$, $d[(u_i, v_i), a] \leq c_u$. Here c_u is the maximum allowable distance to unmatched nodes.

The reader can verify during the proof that by choosing $c_u = 1$ and $c_d = 6$, all inequalities involving these constants hold.

We will be using the following facts about commute times.

LEMMA 4.2.

1. *Commute times satisfy the triangle inequality.*
2. $d[(a,b),c] \leq \frac{C_{a,c}}{C_{a,b}}$ *and* $d[(a,b),c] \leq \frac{C_{b,c}}{C_{a,b}}$.
3. $\frac{C_{a,c}}{C_{a,b}} \leq d[(a,b),c] + 1$ *and* $\frac{C_{b,c}}{C_{a,b}} \leq d[(a,b),c] + 1$.
4. $d[(a,b),c] \leq d[(a,b),d] + \frac{C_{c,d}}{C_{a,b}}$.
5. *If* $d[(x,y),z] \geq k_0$, $d[(a,b),x] \leq k_1$ *and* $d[(a,b),z] \leq k_2$, *then*

$$C_{x,y} \leq \frac{(k_1 + k_2 + 1)}{k_0} C_{a,b}.$$

Proof. Parts 1 and 2 follow immediately from the definition of commute time and relative distance. For part 3,

$$\frac{C_{a,c}}{C_{a,b}} \leq \frac{\min(C_{a,c}, C_{a,c} - C_{a,b}) + C_{a,b}}{C_{a,b}} \leq \frac{\min(C_{a,c}, C_{b,c})}{C_{a,b}} + 1 \leq d[(a,b),c] + 1,$$

where the second inequality follows from part 1. The other case is similar. For part 4,

$$d[(a,b),c] = \frac{\min(C_{a,c}, C_{b,c})}{C_{a,b}} \leq \frac{\min(C_{a,d}, C_{b,d}) + C_{d,c}}{C_{a,b}} = d[(a,b),d] + \frac{C_{d,c}}{C_{a,b}}.$$

Finally, part 5 follows from the application of part 2, part 1, and part 3 to give

$$C_{x,y} \leq \frac{C_{x,z}}{k_0} \leq \frac{(\min(C_{x,a}, C_{x,b}) + \max(C_{a,z}, C_{b,z}))}{k_0} \leq \frac{(d[(a,b),x] + d[(a,b),z] + 1)}{k_0} C_{a,b}.$$

Finally, use the fact that $d[(a,b),x] \leq k_1$ and $d[(a,b),z] \leq k_2$. ∎

LEMMA 4.3. *The matching maintained by CA is always good.*

Proof. The proof is by induction. Consider first the matching M_1 at the start of a phase: there is a single matched pair (p, q), consisting of a page p, whose request started the phase, and a page q that was chosen (out of the shrunk window of the k most recently requested pages) to maximize $C_{p,q}$. In this base case, the distinguished node can be chosen arbitrarily. Clearly, for $a \in W_r \setminus V_{M_1}$, we have $d[(p,q),a] \leq 1$.

Now suppose that M_r is a good matching; we will show that M_{r+1} is also good. We take q to be the distinguished node in the new pair. By the choice of q (maximizing $C_{p,q}$) and the assumption on M_r, M_{r+1} satisfies the following goodness conditions:

1. $d[(p,q),a] \leq c_u$ for any $a \in W_{r+1} \setminus V_{M_{r+1}}$.
2. For any pairs $(u_i, v_i), (u_j, v_j) \in M_r$, $d[(u_i, v_i), u_j] \leq c_d$, and for any $(u_i, v_i) \in M_r$ and $a \notin V_{M_{r+1}}$, $d[(u_i, v_i), a] \leq c_u$.
3. For each pair $(u_i, v_i) \in M_r$, $d[(u_i, v_i), q] \leq c_u$ since q was in the window but not in the matching.

There are two cases to consider, which follow.

Case 1. We did not switch. ($d[(p,q),(u_j,v_j)] \le c_s \ \forall j$.)
Remember that $d[(p,q),(u_i,v_i)] = \min(d[(p,q),u_i], d[(p,q),v_i])$. It might be that $d[(p,q),u_i] > c_d$. But in this case, $d[(p,q),v_i] \le c_s$, otherwise we would have switched. We show that for all pairs (u_j,v_j) in M_r, $d[(u_j,v_j),v_i] \le c_d$, so v_i can play the former distinguished role of u_i.
Let $(u_j,v_j) \in M_r$, $j \ne i$. We have

$$d[(u_j,v_j),v_i] \le d[(u_j,v_j),q] + \frac{C_{q,v_i}}{C_{u_j,v_j}} \le c_u + \frac{(c_s+1)C_{p,q}}{C_{u_j,v_j}},$$

where both inequalities follow from Lemma 4.2, the first from part 4 and the second from part 3. However, from part 5 of Lemma 4.2, since $d[(p,q),u_i] \ge c_d$, $d[(u_j,v_j),q] \le c_u$, and $d[(u_j,v_j),u_i] \le c_d$, it follows that

$$C_{p,q} \le \frac{c_u + c_d + 1}{c_d} C_{u_j,v_j}.$$

Substituting into the previous equation gives

$$d[(u_j,v_j),v_i] < c_u + \frac{(c_s+1)(c_u+c_d+1)}{c_d} \le c_d.$$

Case 2. We did switch the following.
Consider the new edges (p,u_j) and (q,v_j) (chosen so that $d[(p,q),(u_j,v_j)]$ is maximized). We show that p and q become distinguished vertices and that for some other pairs (u_k,v_k), v_k must become the distinguished vertex. The detailed proof that M_{r+1} is good follows.

We will need to use the following three inequalities:

- Since $d[(p,q),(u_j,v_j)] > c_s$,

$$C_{p,q} < \frac{\min\{C_{p,u_j}, C_{p,v_j}, C_{q,u_j}, C_{q,v_j}\}}{c_s}. \tag{4.1}$$

- From Lemma 4.2, part 5, since $d[(p,q),u_j] > c_s$ (because we switched), $d[(u_k,v_k),q] \le c_u$, and $d[(u_k,v_k),u_j] \le c_d$ (both because M_r was good), it follows that for any pair $(u_k,v_k) \in M_r$, $k \ne j$,

$$C_{p,q} < C_{u_k,v_k} \frac{c_u + c_d + 1}{c_s}. \tag{4.2}$$

- Last,

$$d[(p,q),(u_k,v_k)] \le d[(p,q),(u_j,v_j)]. \tag{4.3}$$

Many of the goodness conditions follow immediately from the goodness of M_r. The ones that require proof are described below, each with a derivation.

1. $d[(p,u_j),q] \le c_d$, and $d[(q,v_j),p] \le c_d$. This follows from (4.1).
2. $d[(p,u_j),a] \le c_u$ and $d[(q,v_j),a] \le c_u$ for any unmatched $a \in W_{r+1}$. For the first inequality, applying part 2 of Lemma 4.2, the fact that $C_{p,q} \ge C_{p,a}$ for all unmatched a, and (4.1) gives

$$d[(p,u_j),a] \le \frac{C_{p,a}}{C_{p,u_j}} \le \frac{C_{p,q}}{C_{p,u_j}} \le \frac{1}{c_s}.$$

Similarly, for the second inequality,

$$d[(q, v_j), a] \leq \frac{C_{q,a}}{C_{q,v_j}} \leq \frac{C_{q,p} + C_{p,a}}{C_{q,v_j}} \leq \frac{2}{c_s}.$$

3. For each pair $(u_k, v_k) \in M_r$, $k \neq j$, $d[(u_k, v_k), q] \leq c_d$ and $d[(u_k, v_k), p] \leq c_d$. The first inequality follows from the fact that M_r was good, so $d[(u_k, v_k), q] \leq c_u$. As for the second inequality,

$$\begin{aligned} d[(u_k, v_k), p] &\leq d[(u_k, v_k), q] + \frac{C_{p,q}}{C_{u_k,v_k}} \quad \text{(by part 4 of Lemma 4.2)}, \\ &\leq c_u + \frac{C_{p,q}}{C_{u_k,v_k}} \quad \text{(since } d[(u_k, v_k), q] \leq c_u\text{)} \\ &\leq c_u + \frac{c_u + c_d + 1}{c_s} \quad \text{(by (4.2))} \\ &\leq c_d. \end{aligned}$$

4. For each pair $(u_k, v_k) \in M_r$, either both $d[(p, u_j), u_k] \leq c_d$ and $d[(q, v_j), u_k] \leq c_d$ or v_k can become distinguished.
To prove this, assume that $d[(p, u_j), u_k] > c_d$. By assumption

$$C_{p,u_k} > c_d C_{p,u_j} \tag{4.4}$$

and from (4.1) above,

$$C_{p,q} < C_{p,u_j}/c_s. \tag{4.5}$$

Combining these two facts with part 3 of Lemma 4.2, we obtain

$$\begin{aligned} d[(p, q), u_k] &\geq \frac{C_{p,u_k}}{C_{p,q}} - \frac{C_{p,q}}{C_{p,q}} \\ &> \frac{c_d C_{p,u_j}}{C_{p,q}} - \frac{C_{p,u_j}}{c_s C_{p,q}} \\ &> (c_d - 1/c_s)\frac{C_{p,u_j}}{C_{p,q}}, \end{aligned}$$

and thus

$$d[(p, q), u_k] > d[(p, q), u_j]. \tag{4.6}$$

Therefore, by the choice of j,

$$d[(p, q), v_k] < d[(p, q), (u_j, v_j)]. \tag{4.7}$$

Therefore

$$\begin{aligned} C_{p,v_k} &\leq C_{p,q}(1 + d[(p, q), v_k]) \quad \text{(by part 4 of Lemma 4.2)} \\ &\leq C_{p,q}(1 + d[(p, q), u_j]) \quad \text{(by (4.7))} \\ &\leq C_{p,q} + C_{p,u_j} \\ &\leq C_{p,u_j}(1 + 1/c_2) \quad \text{(by (4.1))} \end{aligned}$$

so $d[(p,u_j),v_k] \le c_d$. The previous four lines of equations also establish that $d[(q,v_j),v_k] \le c_d$ by substituting q for p and v_j for u_j.
Now let $(u_l,v_l) \in M_r$. From (4.4) and (4.5) we have

$$\begin{aligned} C_{q,u_k} &\ge C_{p,u_k} - C_{pq} \\ &> (c_d - 1/c_s)C_{p,u_j} \\ &> (c_d - c_s/2)C_{p,u_j}. \end{aligned}$$

However,

$$\begin{aligned} C_{q,u_k} &\le C_{u_l,v_l}(1 + d[(u_l,v_l),q] + d[(u_l,v_l),u_k]) \\ &\le C_{u_l,v_l}(1 + c_u + c_d) \end{aligned}$$

and thus

$$C_{p,u_j} \le \frac{1 + c_u + c_d}{c_d - c_s/2} C_{u_l,v_l}. \tag{4.8}$$

We have

$$\begin{aligned} d[(u_l,v_l),v_k] &\le d[(u_l,v_l),q] + \frac{C_{p,q}(1 + d[(p,q),v_k])}{C_{u_l,v_l}} \\ &\le c_u + \frac{C_{p,q}(1 + d[(p,q),u_j])}{C_{u_l,v_l}} \quad \text{(by 4.7)} \\ &\le c_u + \frac{3C_{p,u_j}}{2C_{u_l,v_l}} \quad \text{(by 4.1)} \\ &\le c_d \quad \text{(by 4.8).} \qquad \square \end{aligned}$$

The following lemma shows why we are interested in relative distances.

LEMMA 4.4. *Let M be a Markov chain. Let u,v,h be three states in M, and let $\gamma = \Pr(h$ visited during a (u,v) commute$)$. Then*

$$\frac{C_{uv}}{C_{uh}} \ge \gamma \ge \frac{C_{uv}}{C_{uv} + C_{uh}}.$$

Proof. We use the following well-known proposition from renewal theory (see, for example, [7, p. 147]): Consider a Markov chain started in state i. Let $0 < S < \infty$ be a stopping time such that $X_S = i$. Let j be an arbitrary state. Then

$$E_i(\#\text{ of visits to } j \text{ before time } S) = \pi_j E_i(S),$$

where π_j is the stationary probability of state j and $E_i(X)$ is the expected value of random variable X when the chain is started in state i.

Let u,v,h be three states in M. Using the proposition, we obtain that

$$E_u(\#\text{ of visits to } h \text{ during a } (u,v) \text{ commute}) = \pi_h(E_u(T_v) + E_v(T_u)) = \pi_h C_{uv}. \tag{4.9}$$

Consider a random walk starting at u. Let p_1 be the probability that h is visited before v and let p_2 be the probability that h is first visited after v, but before a (u,v) commute has completed. Clearly $\gamma = p_1 + p_2$. Furthermore,

$$\begin{aligned} &E_u(\#\text{ of visits to } h \text{ during a } (u,v) \text{ commute}) \\ &\quad = p_1 E_h(\#\text{ of visits to } h \text{ during the time it takes to go from } h \text{ to } v \text{ to } u \text{ back to } h) \\ &\quad + p_2 E_h(\#\text{ of visits to } h \text{ during the time it takes to go from } h \text{ to } u \text{ and back to } h). \end{aligned}$$

Once again using the proposition, we obtain

$$\begin{aligned} &E_u(\#\text{ of visits to } h \text{ during a } (u,v) \text{ commute}) \\ &\quad = p_1\pi_h(E_h(T_v) + E_v(T_u) + E_u(T_h)) + p_2\pi_h(E_h(T_u) + E_u(T_h)). \end{aligned}$$

Combining this with (4.9), we obtain

$$C_{uv} = p_1(E_h(T_v) + E_v(T_u) + E_u(T_h)) + p_2 C_{uh}.$$

Since $(E_h(T_v) + E_v(T_u) + E_u(T_h)) \geq C_{uh}$, we obtain

$$\gamma \leq \frac{C_{uv}}{C_{uh}}.$$

On the other hand, $(E_h(T_v) + E_v(T_u) + E_u(T_h)) \leq C_{uv} + C_{uh}$, and thus

$$\gamma \geq \frac{C_{uv}}{C_{uv} + C_{uh}},$$

completing the proof of the theorem. □

We are ready to prove the main theorem.

Proof of Theorem 4.1. We show that *CA* incurs an expected number of faults, which is at most a constant times the expected number incurred by the *OPT*.

Let a *hole* of *OPT* be any page in W_r that *OPT* does not have in memory. We maintain a 1:1 mapping from pairs in the matching to holes of *OPT*, satisfying the following two properties:

- If h_i is the *OPT*-hole associated with pair (u_i, v_i) for some i, then $d[(u_i, v_i), h_i] = O(1)$.
- The mapping of pairs to holes is changed only when there is either an *OPT*-fault or a new pair is added to the matching. In both cases, the association changes for $O(1)$ pairs.

Say that an (x, y) *commute begins at time* t if (x, y) is a pair in the matching at time t (i.e., $(x, y) = (u_i, v_i)$ or $(x, y) = (v_i, u_i)$ for some i), *CA* had a fault on the most recent request, and the most recent request was at x. Let $T(x, y)$ be the set of times t such that an (x, y) commute begins at time t. Define a *new* node to be a node that is visited in the current phase but wasn't one of the nodes visited in the previous phase, and let G be the total number of new nodes seen in all phases in the request sequence.

The number of faults incurred by *CA* during a page request sequence, denoted $C(CA)$, satisfies

$$C(CA) = \left(\sum_{(x,y)} \sum_{t \in T(x,y)} 1 \right) + O(G),$$

where the last term comes from ensuring that the last k requests are in memory at the end of each phase.

Let $h(x, y, t)$ be the *OPT*-hole associated with pair (x, y) at time t for $t \in T(x, y)$. Let $X_{x,y,t}$ be the indicator random variable that is 1 if $t \in T(x, y)$ and $h(x, y, t)$ is requested during the (x, y) commute beginning at time t.

For $t \in T(x, y)$, let $t \in Q(x, y)$ if either $h(x, y, t)$ gets swapped to another pair or the phase containing t ends before the (x, y) commute starting at time t completes.

Otherwise let $t \in R(x, y)$. Let $C(OPT)$ be the number of faults incurred by OPT. For any page request sequence, we have

$$\sum_{(x,y)} \sum_{t \in R(x,y)} X_{x,y,t} \leq 2C(OPT)$$

since each OPT-fault can be accounted for at most twice, namely by an (x, y) commute and a (y, x) commute, for some pair (u_i, v_i).

The mapping of pairs to holes changes only if OPT incurs a fault or if a new pair is added to the matching, and then only $O(1)$ pairs are affected. A new pair is added to the matching only when a new node is visited. At most $2g$ commutes are in progress at the end of a phase, where g is the number of new nodes visited during the phase. Thus we have

$$\sum_{(x,y)} \sum_{t \in Q(x,y)} X_{x,y,t} = O(C(OPT) + G).$$

(Note that in fact $\sum_{(x,y)} \sum_{t \in Q(x,y)} 1 = O(C(OPT) + G)$.) Therefore,

$$\begin{aligned} \sum_{(x,y)} \sum_{t \in T(x,y)} X_{x,y,t} &= O(C(OPT) + G) \\ &= O(C(OPT) + k), \end{aligned}$$

where k is the number of pages that can be held in memory, since it is known [8] that any paging algorithm incurs $\Omega(G - k)$ faults.

Last, since $d[(x, y), h(x, y, t)] = O(1)$, by Lemma 4.4 there is a constant $p > 0$ such that for all u, v, t such that $t \in T(u, v)$,

$$E[X_{x,y,t}] \geq p.$$

Putting all this together, we obtain by linearity of expectations that

$$E[C(CA)] = O(E[C(OPT)] + k),$$

so the fault rate of CA is at most a constant factor greater than that of OPT.

It remains only to describe the mapping of pairs to holes.

At the beginning of a phase, the single matched pair is associated with any OPT-hole in the window—note that there is at least one. In general, if h_j is the hole associated with (u_j, v_j), we maintain the following invariant:

$$h_j \text{ is unmatched, or is } u_j, v_j, \text{ or } \{h_j, h_k\} = \{u_k, v_k\}, \text{ for some } k \neq j.$$

Notice that the invariant remains unchanged when the distinguished node in a pair changes. (In the case where $\{h_j, h_k\} = \{u_k, v_k\}$, for some $k \neq j$, and the designated node for pair $\{u_k, v_k\}$ changes, swapping h_j and h_k ensures that the invariant remains true.) The following procedure ensures that at any time h_j is changed, it is set to an unmatched node, or u_j, v_j, or u_k for some $k \neq j$, so $d[(u_j, v_j), h_j] = O(1)$, as required.

The first case to consider is when no switch was performed, and the pair (p, q) needs to find a hole $h(p, q)$. We have the following cases.

1. If $p = h_j$ for some j and $q = h_k$ for some k, then p becomes $h(p, q)$, h_k remains q, and we continue with (u_j, v_j) in place of (p, q).

2. If only one of p or q is associated with a pair, say $q = h(u_k, v_k)$, then q becomes $h(p, q)$, and we continue with (u_k, v_k).
3. At this point, there must be an OPT-hole that is unmatched because there are at least i holes total and only $i - 1$ pairs are associated with a hole. We consider three cases depending on what this unassociated OPT-hole is.
   (a) If the OPT-hole is unmatched, or is p or q, set $h(p, q)$ to this OPT-hole.
   (b) If the OPT-hole is u_k or v_k such that $h_k \in \{u_k, v_k\}$, set $h(p, q)$ to u_k and h_k to v_k.
   (c) If the OPT-hole is u_k or v_k such that $h_k \notin \{u_k, v_k\}$, set $h(p, q)$ to h_k and h_k to the unassociated OPT-hole.

The second case is when a switch was performed, producing pairs (p, u_j) and (q, v_j). In this case, we first unmatch h_j from (u_j, v_j) and then apply two steps according to the directions in the no-switch case above—first for (u_j, p) and then for (q, v_j).

When OPT incurs a fault at a hole h_j, the page is loaded into memory, so it is no longer a hole. The pair (u_j, v_j) then finds an unassociated OPT-hole as in the no-switch case above. This scheme satisfies the two required properties of the mapping from pairs to holes. □

Acknowledgment. We would like to thank the reviewers for extensive comments and excellent suggestions.

REFERENCES

[1] S. Ben-David, A. Borodin, R. M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in on-line algorithm*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing, ACM, New York, 1990, pp. 379–388.

[2] A. Borodin, S. Irani, P. Raghavan, and B. Schieber, *Competitive paging with locality of reference*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 249–259.

[3] A. Z. Broder, *Generating random spanning trees*, in Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 1989, pp. 442–447.

[4] A. K. Chandra, P. Raghavan, W. L. Ruzzo, R. Smolensky, and P. Tiwari, *The electrical resistance of a graph captures its commute and cover times*, in Proceedings of the 21st Annual ACM Symposium on Theory of Computing, ACM, New York, 1989, pp. 574–586.

[5] D. Cheriton and K. Harty, *Application-Controlled Physical Memory Using External Page-Cache Management*, Tech. report, Department of Computer Science, Stanford University, Stanford, CA, 1991.

[6] P. J. Denning, *Working sets past and present*, IEEE Trans. Software Engrg., SE-6 (1980), pp. 64–84.

[7] C. Derman, *Finite State Markov Decision Processes*, Academic Press, New York, 1970.

[8] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. Sleator, and N. Young, *On competitive algorithms for paging problems*, J. Algorithms, 12 (1991), pp. 685–699.

[9] P. A. Franaszek and T. J. Wagner, *Some distribution-free aspects of paging performance*, J. Assoc. Comput. Mach., 21 (1974), pp. 31–39.

[10] S. Irani, A. R. Karlin, and S. Phillips, *Strongly competitive algorithms for paging with locality of reference*, in Proceedings of the Third Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, PA, 1992, pp. 228–236.

[11] S. Irani, R. Karp, M. Kearns, and M. Luby, *private communication*, University of California, Berkeley, CA, 1991.

[12] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 70–119.

[13] R. M. Karp, *A characterization of the minimum cycle mean in a digraph*, Discrete Math., 23 (1978), pp. 309–311.

[14] P. A. W. Lewis and G. S. Shedler, *Empirically derived models for sequences of page exceptions*, IBM J. Res. Develop., 17 (1973), pp. 86–100.

[15] D. McNamee and K. Armstrong, *Extending the Mach External Pager Interface to Accommodate User-Level Page Replacement Policies*, Tech. report 90-09-05, Department of Computer Science and Engineering, University of Washington, Seattle, WA, 1990.

[16] P. Raghavan, *Lecture Notes on Randomized Algorithms*, Tech. report RC 15340, IBM Research, Almaden, CA, 1990.

[17] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, in 16th International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 372, Springer-Verlag, New York, 1989, pp. 687–703; revised version also available as IBM Research Report RC15840, IBM Research, Almaden, CA, 1990.

[18] G. S. Shedler and C. Tung, *Locality in page reference strings*, SIAM J. Comput., 1 (1972), pp. 218–241.

[19] D. D. Sleator and R. E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[20] J. R. Spirn, *Program Behavior: Models and Measurements*, Elsevier Computer Science Library, Elsevier, Amsterdam, 1977.

[21] J. S. Vitter and P. Krishnan, *Optimal prefetching via data compression*, in Thirty-Second Annual IEEE Symposium on Foundations of Computer Science, IEEE Press, Piscataway, NJ, 1991.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 923–964

ASYMPTOTIC BEHAVIOR OF THE HEIGHT IN A DIGITAL SEARCH TREE AND THE LONGEST PHRASE OF THE LEMPEL–ZIV SCHEME*

CHARLES KNESSL† AND WOJCIECH SZPANKOWSKI‡

Abstract. We study the height of a *digital search tree* (DST) built from n random strings generated by an unbiased memoryless source (i.e., all symbols are equally likely). We shall argue that the height of such a tree is equivalent to the length of the longest phrase in the Lempel–Ziv parsing scheme that partitions a random sequence into n phrases. We also analyze the longest phrase in the Lempel–Ziv scheme in which a string of *fixed* length m is parsed into a *random* number of phrases. In the course of our analysis, we shall identify four natural regions of the height distribution and characterize them asymptotically for large n. In particular, for the region where most of the probability mass is concentrated, the asymptotic distribution of the height exhibits an exponential of a Gaussian distribution (with an oscillating term) around the most probable value $k_1 = \lfloor \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} \rfloor + 1$. More precisely, we shall prove that the asymptotic distribution of a DST is concentrated on either the one point k_1 or the two points $k_1 - 1$ and k_1, which actually proves (slightly modified) Kesten's conjecture quoted in [*Probab. Theory Related Fields*, 79 (1988), pp. 509–542]. Finally, we compare our findings for DST with the asymptotic distributions of the height for other digital trees such as tries and PATRICIA tries. We derive these results by a combination of analytic methods such as generating functions, Laplace transform, the saddle point method, and ideas of applied mathematics such as linearization, asymptotic matching, and the WKB method. Our analysis makes certain assumptions about the forms of some of the asymptotic expansions as well as their asymptotic matching. We also present detailed numerical verification of our results.

Key words. digital search trees, Lempel–Ziv algorithm, height distribution, longest phrase distribution, Laplace transform, saddle point method, matched asymptotics, linearization, WKB method, elliptic theta function

AMS subject classifications. 68Q25, 68P05

PII. S0097539799356812

1. Introduction.

The heart of some universal data compression schemes is the parsing algorithm due to Lempel and Ziv [32] (cf. also [31]). It partitions a sequence into phrases (blocks) of variable sizes such that a new block is the shortest substring not seen in the past as a phrase. For example, the string 11001010001000100 is parsed into (1)(10)(0)(101)(00)(01)(000)(100). This parsing algorithm plays a crucial role in numerous applications such as efficient transmission of data discriminating between information sources, test of randomness, estimating the statistical model of individual sequences, and so forth. The parameters of interest in these applications are the number of phrases, the number of phrases of a given size, the size of a phrase, the length of a sequence built from a given number of phrases, the length of the longest phrase, etc. Some of them have already been analyzed (e.g., number of phrases [2, 13], the size of a typical phrase [15, 23]). Here we study the distribution of the longest phrase.

*Received by the editors June 15, 1999; accepted for publication (in revised form) January 20, 2000; published electronically August 9, 2000.

http://www.siam.org/journals/sicomp/30-3/35681.html

†Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, Chicago, IL 60607-7045 (knessl@uic.edu). The research of this author was supported by DOE grant DE-FG02-96ER25168.

‡Department of Computer Science, Purdue University, W. Lafayette, IN 47907 (spa@cs.purdue.edu). The research of this author was supported by NSF grants NCR-9415491 and CCR-9804760.

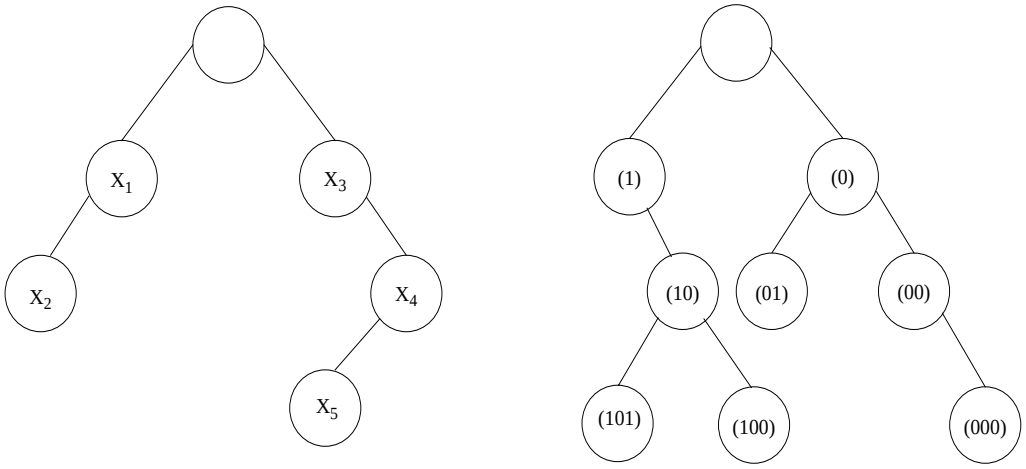


FIG. 1. *DST built from (left) five strings* $X_1 = 11100\ldots$, $X_2 = 11011\ldots$, $X_3 = 00110\ldots$, $X_4 = 00001\ldots$, $X_5 = 001011\ldots$*; (right) eight Lempel–Ziv phrases of the string* $11001010001000100\ldots$, *that is,* (1)(10)(0)(101)(00)(01)(000)(100).

The Lempel–Ziv parsing scheme can be efficiently implemented by a special digital tree called the *digital search tree* (DST). This finds myriad applications in computer science and telecommunications such as dynamic hashing, partial match retrieval of multidimensional data, searching and sorting, pattern matching, conflict resolution algorithms for broadcast communications, data compression, coding, security, genes searching, DNA sequencing, genome maps, and so forth. The DST is constructed as follows (cf. Figure 1). We consider n, possibly infinite, strings of symbols from a finite alphabet Σ (however, for simplicity we work only with the binary alphabet $\Sigma = \{0, 1\}$). The empty string is stored in the root, while the first string occupies the right or the left child of the root depending on whether its first symbol is "1" or "0." The remaining strings are stored in available nodes (that are directly attached to nodes already existing in the tree). The search for an available node follows the prefix structure of a string. The rule is simple: if the next symbol in a string is "1" we move to the right; otherwise we move to the left. The resulting tree has $n + 1$ internal nodes. The details can be found in [17] and [24].

The parsing scheme according to Lempel–Ziv with a *fixed* number, n, of phrases (cf. [11, 13, 23]) can be accomplished on the associated DST as follows: We consider an infinite sequence of binary symbols and partition it until we create the first n phrases. Assuming the first phrase is an *empty* one, we store it in the root of a DST, and all other phrases are stored in internal nodes. When a new phrase is created, the search starts at the root and proceeds down the tree as directed by the input symbols exactly in the same manner as in the DST construction. For example, for the binary alphabet, "0" in the input string means move to the left and "1" means proceed to the right. The search is completed when a branch is taken from an existing tree node to a new node that has not been visited before. Then an edge and a new node are added to the tree. Phrases created in such a way are stored directly in the nodes of the tree (cf. Figure 1).

We also study another model, called the *Lempel–Ziv model*, in which a string of *fixed* length m is parsed according to the Lempel–Ziv algorithm. We can again use

the associated DST to parse the string, but this time the number of phrases, M_m, and hence the number of nodes in the DST, is random. It is known (cf. [2, 32]) that the number of phrases $M_m \sim mh/\log m$ almost surely (a.s.) where h is the entropy of the source.

In this paper, we consider DSTs built over n randomly and independently generated strings of binary symbols. We assume that every symbol is equally likely, and thus we are within the framework of the so-called *symmetric Bernoulli* model. In other words, the strings are emitted by an *unbiased memoryless* source. Our interest lies in establishing the *asymptotic distribution* of the height, $\mathcal{H}_n$, for random DST. The height is the longest path in such trees, and its distribution is of considerable interest for several applications (e.g., the length of the longest phrase in the Lempel–Ziv scheme, maximum search time, and sorting).

We now summarize our main results. First of all, as a consequence of our analysis we prove that the average height of a DST built from n independent strings is $\mathbf{E}[\mathcal{H}_n] = \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + O(1)$. However, our main contribution lies in establishing the asymptotic distribution $\mathsf{Pr}\{\mathcal{H}_n \le k\}$ of the height as $n \to \infty$. In Theorem 2.1 we shall identify four natural regions of this distribution and characterize them asymptotically as n and k become large. In particular, we show that for the region where most of the probability mass is concentrated, the asymptotic distribution of the height exhibits an exponential of a Gaussian distribution (with an oscillating term) around the most probable value $k_1 = \lfloor \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} \rfloor + 1$. In fact, we shall prove that the asymptotic distribution of a DST is concentrated on either the one point k_1 or the two points $k_1 - 1$ and k_1. More precisely, we show the existence of certain subsequences of n such that the asymptotic distribution of the height concentrates only on k_1, or on k_1 and $k_1 - 1$. This proves (slightly modified) Kesten's conjecture quoted in Aldous and Shields [2, p. 538]. Finally, we obtain the asymptotic distribution of the length of the longest phrase in the Lempel–Ziv model (cf. Theorem 2.3).

Digital trees, that is, tries, PATRICIA tries, and DSTs, have been extensively analyzed in the past (cf. [2, 5, 6, 7, 8, 13, 14, 16, 17, 20, 22, 23, 24, 26, 27, 29, 30]). However, relatively little is known about the height of DSTs. An exception is the paper by Pittel [26] who proved that the height $\mathcal{H}_n \sim \log_2 n$ a.s. Later Aldous and Shields [2] improved Pittel's result to $\mathcal{H}_n \sim \log_2 n + \sqrt{2\log_2 n}$ a.s. No distributional result for the height is known to us. We fill this gap by presenting a complete characterization of the asymptotic distribution of the height.

Finally, we say a few words about our method of derivation and put our results in a larger perspective. From a mathematical view point, we study a nonlinear recurrence equation. The distribution $h_n^k = \mathsf{Pr}\{\mathcal{H}_n \le k\}$ satisfies recurrence (2.2) with the initial condition (2.3). We shall solve asymptotically this recurrence by methods of applied mathematics such as linearization, matched asymptotics, and the WKB method (cf. section 5.2). In passing, we mention that in a companion paper [19] we analyzed two similar recurrences: The so-called b-tries recurrence satisfies

$$h_n^{k+1} = 2^{-n} \sum_{i=0}^{n} \binom{n}{i} h_i^k h_{n-i}^k, \qquad k \ge 0,$$

with the initial condition $h_n^0 = 1$ for $n = 0, 1, 2, \ldots, b$ and $h_n^0 = 0$ for $n > b$. For

PATRICIA tries the distribution $h_n^k = \mathsf{Pr}\{\mathcal{H}_n^P \le k\}$ of the height $\mathcal{H}_n^P$ satisfies

$$h_n^{k+1} = 2^{-n+1} h_n^{k+1} + 2^{-n} \sum_{i=1}^{n-1} \binom{n}{i} h_i^k h_{n-i}^k, \quad k \ge 0,$$

with the initial conditions $h_0^0 = h_1^0 = 1$ and $h_n^0 = 0$ for $n \ge 2$. Surprisingly enough, although these recurrences resemble the DST recurrence, they lead to quite different solutions. The reader is referred to [19] for details.

The paper is organized as follows. In the next section, we present our main results for DSTs (cf. Theorem 2.1) and the Lempel–Ziv model (cf. Theorem 2.3). In section 3 we present detailed numerical results and discuss consequences of our findings. The proofs of these results are relegated to sections 4 and 5. These make certain assumptions about the forms of the asymptotic expansions and their matching.

2. Main results. We let $\mathcal{H}_n$ be the height of a DST and we denote its probability distribution by

$$h_n^k = \mathsf{Pr}\{\mathcal{H}_n \le k\}. \tag{2.1}$$

It satisfies the difference equation

$$h_{n+1}^{k+1} = \sum_{i=0}^{n} \binom{n}{i} 2^{-n} h_i^k h_{h-i}^k, \quad k \ge 0, \tag{2.2}$$

with the initial condition

$$h_0^0 = h_1^0 = 1; \quad h_n^0 = 0, \quad n \ge 2. \tag{2.3}$$

This follows from $\mathcal{H}_{n+1} = \max\{\mathcal{H}_i^{LT}, \mathcal{H}_{n-i}^{RT}\} + 1$, where $\mathcal{H}_i^{LT}$ and $\mathcal{H}_{n-i}^{RT}$ denote, respectively, the left subtree and the right subtree of sizes i and $n-i$, which happens with probability $2^{-n}\binom{n}{i}$. The root contains one string (or an empty string).

We can easily show that $h_n^k = 0$ for $n \ge 2^{k+1}$ (i.e., a balanced tree) and $h_n^k = 1$ for $k \ge n-1$ (i.e., a skewed tree). It therefore suffices to consider the range $k+2 \le n \le 2^{k+1} - 1$. We also note that the exponential generating function

$$H_k(z) = \sum_{n=0}^{\infty} \frac{h_n^k z^n}{n!} = \sum_{n=0}^{2^{k+1}-1} \frac{h_n^k z^n}{n!} = \sum_{n=0}^{k+1} \frac{z^n}{n!} + \sum_{n=k+2}^{2^{k+1}-1} h_n^k \frac{z^n}{n!} \tag{2.4}$$

satisfies

$$H'_{k+1}(2z) = H_k^2(z), \qquad k \ge 0, \tag{2.5}$$

with $H_0(z) = 1 + z$. Thus, for any k, the generating function $H_k(z)$ is a polynomial of degree $2^{k+1} - 1$ and the leading $k+2$ coefficients in this polynomial are the same as those in the Taylor series of e^z.

Below we give the exact expressions for h_n^k for a few values of n that are close to either k or 2^{k+1}:

$$h_n^{n-2} = 1 - 2^{-n^2/2} 2^{3n/2} 2^{-1}, \qquad n \ge 2; \tag{2.6}$$

$$h_n^{n-3} = 1 - (n-2) 2^{-n^2/2} 2^{5n/2} 2^{-3}, \qquad n \ge 4; \tag{2.7}$$

$$h_n^{n-4} = 1 - \left(\frac{n^2}{2} - \frac{5n}{2} + \frac{8}{3} - \frac{2}{3} 4^{5-n} \right) 2^{-n^2/2} 2^{7n/2} 2^{-6}, \quad n \ge 6; \tag{2.8}$$

and

$$h^k_{2^{k+1}-1} = (2^{k+1}-1)!2^{-k2^{k+1}}4^{2^k-1}\prod_{\ell=1}^{k}\left(\frac{1}{2^{\ell+1}-1}\right)^{2^{k-\ell}}, \qquad k \geq 0, \tag{2.9}$$

$$h^k_{2^{k+1}-2} = (\,2^{k+1}-2)!2^{-(2k+1)2^k}2^{3k+1} \times \prod_{\ell=1}^{k}\frac{1}{1-2^{-\ell}}\left[\prod_{m=1}^{\ell-1}(1-2^{-m-1})^{-2^{-m}}\right]^{2^{\ell-1}}, \quad k \geq 1. \tag{2.10}$$

We next consider the asymptotic limit $n \to \infty$. A singular perturbation analysis of the problem (2.2) shows that there are four cases of k that lead to different asymptotic expansions of h^k_n: $n \to \infty$ with $n-k$ fixed; $n,k \to \infty$ with $0 < k/n < 1$; $n,k \to \infty$ with $\xi = n2^{-k}$ fixed and $0 < \xi < 2$; and $n,k \to \infty$ with $2^{k+1} - n = O(1)$. We note that the last limit is only possible if n is close to a power of 2.

Using ideas of applied mathematics, such as linearization and asymptotic matching, we obtain the following results, listed in decreasing size of k/n. The derivation of these results is presented in sections 4 and 5, where we make certain assumptions about the forms of the asymptotic expansions, as well as the asymptotic matching between the various scales. In particular, the result in case (iii) (central regime) assumes a WKB expansion in the form (5.31) (cf. section 5 and [18, 21]).

THEOREM 2.1. *The distribution of the height for DSTs built from n independent strings generated by an unbiased memoryless source (and thus the length of the longest phrase in the Lempel–Ziv algorithm with a fixed number, n, of phrases) has the following asymptotic expansions:*

(i) FAR RIGHT-TAIL REGIME: $n,k \to \infty$, $n-k = j = O(1)$, $j \geq 2$,

$$\mathsf{Pr}\{\mathcal{H}_n > k\} = 1 - h^k_n \sim 2^{-j^2/2}2^{j/2}\frac{1}{(j-2)!}n^{j-2}2^{-n^2/2}2^{(j-1/2)n}. \tag{2.11}$$

(ii) RIGHT-TAIL REGIME: $n,k \to \infty$, $0 < k/n < 1$ *or* $\alpha \equiv 1/(1-k/n) = n/(n-k) \in (1,\infty)$,

$$1 - h^k_n \sim 2^{-k^2/2}2^{-k/2}F^k_n, \tag{2.12}$$

where

$$F^k_n = \frac{n^n}{(n-k)^{n-k}k^k}\frac{\sqrt{\pi n}(n-k)^{3/2}}{\sqrt{2}\ k^{5/2}}I(\alpha), \tag{2.13}$$

$$I(\alpha) = \frac{1}{\pi}\sum_{m=0}^{\infty}\frac{(-1)^m}{(\alpha-1)^m}\left[\prod_{\ell=1}^{m}(1-2^{-\ell})\right]^{-1}, \qquad \alpha > 2,$$

$$= \frac{1}{2\pi i}\int_{\frac{1}{2}-i\infty}^{\frac{1}{2}+i\infty}\frac{e^{z\log(\alpha-1)}}{\sin(\pi z)}\prod_{\ell=1}^{\infty}\exp\left(\frac{1-2^{\ell z}}{\ell(2^\ell-1)}\right)dz, \qquad \alpha > 1. \tag{2.14}$$

(iii) CENTRAL REGIME: *Under the WKB hypothesis discussed in section 5.2, for $n,k \to \infty$, $\xi = n2^{-k}$, ξ fixed and $0 < \xi < 2$, we have*

$$h^k_n \sim A(\xi)e^{-n\Phi(\xi)}, \tag{2.15}$$

$$A(\xi) = e^{-\xi\Phi'(\xi)-\Phi(\xi)}\sqrt{1+2\xi\Phi'(\xi)+\xi^2\Phi''(\xi)}. \tag{2.16}$$

By asymptotic matching (cf. section 5), we find that the function $\Phi(\xi)$ satisfies, for $\xi \to 2^-$,

$$\Phi(\xi) \sim \frac{1}{2}\log\left(\frac{e^2}{2C_0}\right) + \frac{1}{2}(2-\xi)\log(2-\xi) + \frac{1}{4}(\xi-2)\log(2C_0), \tag{2.17}$$

$$C_0 = \prod_{\ell=1}^{\infty}(1-2^{-\ell-1})^{-2^{-\ell}} = 1.20675\ldots, \tag{2.18}$$

and for $\xi \to 0^+$

$$\Phi(\xi) \sim 2^{-9/8}\frac{(\log 2)^{3/2}}{(\log\xi)^2}\xi^{1/2-1/\log 2}\exp\left(-\frac{[\log\xi - \log(-\log_2\xi)]^2}{2\log 2}\right) \times Q_*\left(\log_2\xi - \log_2(-\log_2\xi) + \frac{1}{2}\right). \tag{2.19}$$

Here $Q_*(z)$ is a periodic function of period one:

$$Q_*(z) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-s^2/2}Q\left(z + \frac{is}{\sqrt{\log 2}}\right)ds,$$

where $Q(z)$ has the Fourier series

$$Q(z) = 2^{13/12}\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-1}\exp\left[-\frac{\pi^2}{3\log 2} - \sum_{m=1}^{\infty}\frac{\cos(2m\pi z)}{m\sinh(2\pi^2 m/\log 2)}\exp\left(-\frac{2\pi^2 m}{\log 2}\right)\right]. \tag{2.20}$$

An alternate representation is

$$Q(z) = \frac{e^{-\pi i z}}{\sin(\pi z)}e^{p(z)},$$

where

$$e^{p(z)} = e^{-(\log 2)z^2/2}e^{\pi i(z+\frac{3}{2})}2^{1/8}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\vartheta_1\left(\frac{i}{2}z\log 2\right)$$

and $\vartheta_1(u)$ is the Jacobi theta function defined by (cf. [3]*)*

$$\vartheta_1(u) = \vartheta_1(u|\tau) = 2\sum_{m=1}^{\infty}(-1)^{m+1}q^{(m-1/2)^2}\sin((2m-1)u) = 2q^{1/4}\sin(u)\prod_{m=1}^{\infty}(1-2\cos(2u)q^{2m}+q^{4m})(1-q^{2m}), \tag{2.21}$$

where $q = 1/\sqrt{2} = e^{\pi i\tau}$ with $\tau = (i\log 2)/(2\pi)$.

(iv) Left-tail regime: $n \to \infty$, $2^{k+1} - n = M = O(1)$, $M \geq 1$,

$$h_n^k \sim \frac{2\sqrt{\pi}}{(M-1)!}\left(\frac{2C_0}{e^2}\right)^{2^k}2^{(M-1/2)k}, \tag{2.22}$$

where C_0 is defined in (2.18).

Remark. We note that the leading term for h_n^k or $1-h_n^k$ is completely determined for cases (i), (ii), and (iv). However, for case (iii) the result involves the function $\Phi(\xi)$. We have not been able to determine Φ completely, except for its behaviors as $\xi \to 0$ and $\xi \to 2$. In section 3 we discuss the numerical computation of Φ and also give a refinement of (2.19). Our analytical results predict that Φ is finite at $\xi = 2$, with $\Phi(2) = 1-\log(\sqrt{2C_0}) = .559461\ldots$, but Φ' has a logarithmic singularity with $\Phi'(\xi) \sim -\log(\sqrt{2-\xi})$ as $\xi \to 2$. As $\xi \to 0^+$, (2.19) shows that Φ as well as all its derivatives vanish. The dominant term in the right side of (2.19) is $\exp[-(\log 2)(\log_2 \xi)^2/2]$. It is interesting to note that in [19] we obtained results analogous to (2.15) for b-tries and for PATRICIA trees. For b-tries the corresponding Φ satisfies $\Phi(\xi) \sim \xi^b/(b+1)!$ as $\xi \to 0$, so that this function has a zero of order b at $\xi = 0$. For the PATRICIA model the dominant term is the same as that for the DST model (i.e., $\log \Phi$ has the same leading term for the two models). More precisely, Φ in PATRICIA becomes

$$\Phi(\xi) \sim \frac{1}{2}\rho_0 e^{\varphi(\log_2 \xi)} \xi^{3/2} \exp\left(-\frac{\log^2 \xi}{2\log 2}\right), \qquad \xi \to 0^+,$$

where $\rho_0 \approx 1.73137$ and $\varphi(\cdot)$ is periodic with period one. However, by examining the second terms for $\log \Phi$ we see that Φ is flatter near $\xi = 0$ for the DST model. For both DST and PATRICIA, the function Φ shows oscillatory behavior as $\xi \to 0$, while this is not the case for b-tries.

Since $\Phi(\xi) > 0$ for $0 < \xi \le 2$ (see also the numerical results in section 3), h_n^k is exponentially small for cases (iii) and (iv), while for cases (i) and (ii) $1 - h_n^k$ is exponentially small. For case (i) $1 - h_n^k$ is (roughly) $O(2^{-n^2/2})$, for case (ii) $1 - h_n^k = O(2^{-k^2/2})$, while for cases (iii) and (iv) $h_n^k = O(e^{-n\Phi(\xi)})$. For a fixed large n, most of the probability mass occurs in that range of k where h_n^k changes from $h_n^k \approx 0$ to $h_n^k \approx 1$. We think of plotting h_n^k as a function of k so that (i) and (ii) apply in the right tail of the distribution while (iii) and (iv) apply in the left tail. The mass must thus be concentrated in the asymptotic matching region between cases (ii) and (iii). If we let $\xi \to 0$ sufficiently rapidly so that $n\Phi(\xi) \to 0$, then we can approximate (2.15) by $Ae^{-n\Phi} \sim 1 - n\Phi$ and then (ii) and (iii) asymptotically match, as is shown in section 5. We can also consider a limit where $n \to \infty$, $\xi \to 0$ with $n\Phi$ bounded, or even $n\Phi \to \infty$. Then (2.15) can be approximated by $e^{-n\Phi}$ or by $Ae^{-n\Phi}$, with Φ replaced by its expansion (2.19) as $\xi \to 0^+$.

We summarize our observations in the following corollary.

COROLLARY 2.2. *The asymptotic distribution of DST height is concentrated among the two points $k_1 - 1$ and k_1 where*

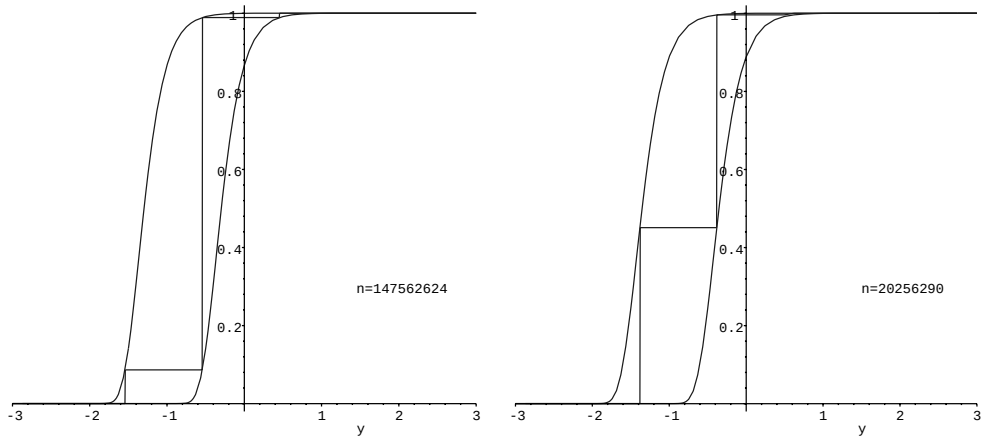
$$k_1 = k_1(n) = \left\lfloor \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} \right\rfloor + 1,$$

that is,

$$\mathsf{Pr}\{\mathcal{H}_n = k_1 - 1 \text{ or } k_1\} = 1 - o(1)$$

as $n \to \infty$. More precisely, (i) *there are subsequences n_i for which $\mathsf{Pr}\{\mathcal{H}_{n_i} = k_1\} = 1 - o(1)$ provided that*

$$\Delta(n_i) = \sqrt{2\log_2 n_i}\left\langle \log_2 n_i + \sqrt{2\log_2 n_i} - \log_2(\sqrt{2\log_2 n_i}) + \frac{1}{\log 2} - \frac{1}{2} \right\rangle - \frac{3}{2}\log_2(\sqrt{2\log_2 n_i}) \to \infty$$



(a) (b)

FIG. 2. *Asymptotic distributions for the heights of DSTs and their corresponding lower and upper bounds.*

as $i \to \infty$*;* (ii) *there are subsequences* n_i *for which* $\mathsf{Pr}\{\mathcal{H}_{n_i} = k_1 - 1 \text{ or } k_1\} = 1 - o(1)$ *provided that* $\Delta(n_i) = O(1)$.

Proof. To establish it we first simplify the expression in the matching region by setting

$$k_\ell = \left\lfloor \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} \right\rfloor + \ell \tag{2.23}$$
$$= \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} - \beta(n) + \ell,$$

where ℓ is an integer and

$$\beta(n) = \left\langle \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} \right\rangle. \tag{2.24}$$

Here $\langle x \rangle = x - \lfloor x \rfloor$ is the fractional part of x. By using (2.23) in $e^{-n\Phi}$ (with Φ replaced by (2.19)) and simplifying the expression for n large, we are led to

$$h_n^k \approx \exp\left(\frac{-Q_*}{2\sqrt{\log 2}} e^{\beta - \ell - 1/\log 2} 2^{\sqrt{2\log_2 n}(\beta-\ell) - (\beta-\ell)^2/2 - \frac{3}{4}\log_2(2\log_2 n)} \right), \tag{2.25}$$

where $Q_* = Q_*(\log_2 n - \log_2(\sqrt{2\log_2 n}) + 1/2)$ and $\beta = \beta(n)$ is as in (2.24). In (2.25) we write $\approx$ rather than $\sim$ since we neglected a factor $1 + O(1/\sqrt{\log n})$ in the exponent. Also, we note that $Q_*(z)$ is almost constant, with very small fluctuations, as explained below (2.29).

Now examine (2.25) for $\ell = 0$ and $\ell = 1$. If $0, < \beta < 1$ and $\ell = 0$, then h_n^k is small as $n \to \infty$. If $0 < \beta < 1$ and $\ell = 1$, h_n^k is close to one. This shows that for $0 < \beta(n) < 1$ the mass becomes concentrated at a single point as $n \to \infty$ (cf. Figure 2), corresponding to $\ell = 1$ in (2.23). It also follows from our analysis that the mean height is

$$\mathbf{E}[\mathcal{H}_n] = \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + O(1), \quad n \to \infty. \tag{2.26}$$

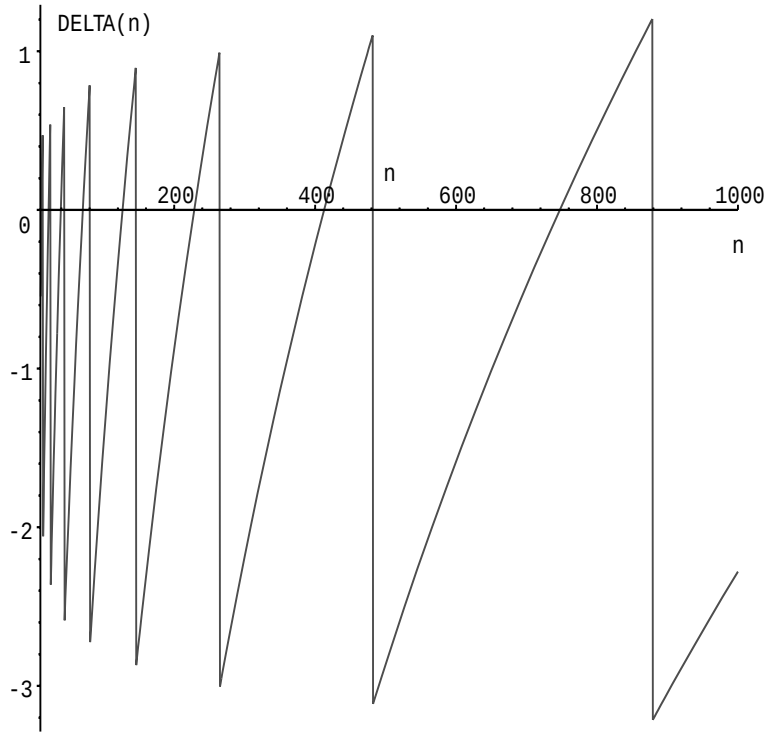


FIG. 3. *The function* $DELTA(n) = \Delta(n) = \beta(n)\sqrt{2\log_2 n} - 1.5\log_2\sqrt{2\log_2(n)}$ *versus* n.

In passing we recall that the first term of the above was already derived by Pittel [26], while the second term was established by Aldous and Shields [2]. In both cases the convergence is in probability sense.

We now show that one can find special sequences $n(i)$ such that $n(i) \to \infty$ as $i \to \infty$, and along these sequences there is probability mass concentrated at two points, corresponding to $\ell = 0$ and $\ell = 1$ in (2.23). We define

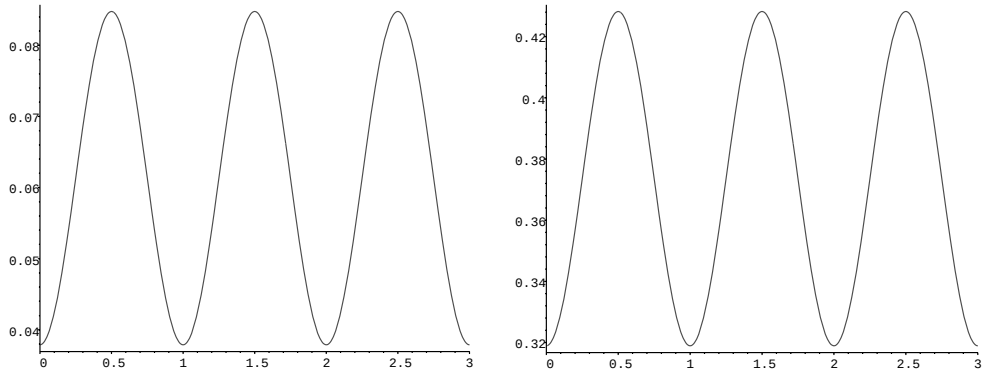
$$\Delta(n) = \beta(n)\sqrt{2\log_2 n} - \frac{3}{2}\log_2(\sqrt{2\log_2 n}) \tag{2.27}$$

and note that if $\Delta(n)$ is $O(1)$, then h_n^k in (2.25) is not asymptotically small at $\ell = 0$. We show in Figure 3 and more precisely in section 3 that we can find sequences $n(i)$ such that $\Delta(n(i))$ remains bounded, and in fact we can have $\Delta \to 0$ for some of these (cf. (3.13)). If we consider $n(i)$ with $\Delta(n(i)) \to 0$, then as $n \to \infty$ for $\ell = 0$ the right side of (2.25) asymptotically becomes

$$\exp\left[-\frac{Q_*}{2\sqrt{\log 2}}\exp\left(-\frac{1}{\log 2}\right)\right]. \tag{2.28}$$

This does not approach a limit as $n \to \infty$ due to the oscillatory behavior of $Q(z)$ and hence $Q_*(z)$. However, we show in section 4 that the nonconstant terms in the Fourier series for Q (to be precise, the series for $\log Q$) are *numerically* very small. Thus we can use the approximation resulting by using only the constant term, which yields

$$Q(z) \approx Q^0 = 2^{13/12}\left[\prod_{m=1}^{\infty}(1-2^{-m})^{-1}\right]\exp\left(-\frac{\pi^2}{3\log 2}\right) = .063716934676\ldots. \tag{2.29}$$



(a) (b)

FIG. 4. (a) *A plot of* $(Q(z) - 0.063716934676376354004922 00) \cdot 10^{24}$ *over three periods.* (b) *A plot of* $(Q_*(z) - 0.063716934676) \cdot 10^{12}$ *over three periods.*

Here $\approx$ means that $Q(z)$ is constant to the number of significant digits given in (2.29).

Now, $Q_*(z)$ is not the same as $Q(z)$. However, in Appendix A we derive the following alternate representation for $Q_*(z)$:

$$Q_*(z) = 2^{z^2/2}\sqrt{\frac{\log 2}{2\pi}}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}\frac{(-1)^{m+1}2^{-m^2/2}2^{m/2}(2^{m-1/2}-2^{-m+1/2})}{2^{m-1/2}+2^{-m+1/2}+2^z+2^{-z}}. \tag{2.30}$$

We tabulate both $Q(z)$ and $Q_*(z)$ in Table 1 (cf. section 3) and sketch these functions in Figure 4. This shows that $Q_*(z)$ is also nearly constant with $Q_*(z) \approx Q_*^0 \approx .0637169$, which agrees with (2.29) to the accuracy given. Here Q_*^0 is the constant term of the Fourier series for $Q_*(z)$. Using (2.29) in (2.28) gives an estimate of the mass at $\ell = 0$ along sequences $n(i)$ that have $\Delta(n(i)) \to 0$. Note that $\Delta \to 0$ implies that $\beta \to 0$, in view of (2.27). In section 3 we test the accuracy of (2.28). □

We are now in a position to compare our results to the corresponding results for PATRICIA trees that we recently obtained in [19]. With regards to the average $\mathbf{E}[\mathcal{H}_n]$ given by (2.26) the first two terms are the same as for PATRICIA (see also Devroye [5]), but the third term for PATRICIA is $O(1)$, while for DST it is $O(\log\log n)$. Thus (2.26) shows that the improvement of the *DST trees over the PATRICIA tries* appears only at the third term in the asymptotics of $\mathbf{E}[\mathcal{H}_n]$. Since the coefficient of the $\log\log n$ term in (2.26) is negative, $\mathbf{E}[\mathcal{H}_n]$ is asymptotically smaller for DST. With respect to the limiting distribution in the central regime, both are of an exponential of a Gaussian type. However, the DST distribution function contains an additional term $O(\log\log n)$ in the double exponent. This additional term prohibits the limiting distribution for DST to be concentrated in some cases on k_1 and $k_1 + 1$, which does occur for PATRICIA height.

Finally, we shall discuss the Lempel–Ziv model in which a random string of *fixed* length, m, is partitioned into a random number, M_m, of phrases. We shall use the results of Theorem 2.1 to prove the asymptotic distribution of the longest phrase among the M_m random phrases. Let us first introduce for any $\varepsilon > 0$

$$\delta_m = \mathsf{Pr}\{M_m \notin [(1-\varepsilon)\mu(m), (1+\varepsilon)\mu(m)]\},$$

where $\mu(m) = m/\log_2 m$ (i.e., the typical number of phrases). It is known that $\delta_m \to 0$ as $m \to \infty$ (cf. [2, 13, 32]). In fact, a stronger result is known. Jacquet and Szpankowski [13] proved that $\delta_m = O(e^{-R\varepsilon\sqrt{m}})$ for some $R > 0$.

Now, we can formulate our second result.

THEOREM 2.3. *Consider the Lempel–Ziv model in which a fixed string of length m is parsed according to the Lempel–Ziv algorithm. Let $\mathcal{H}_m^{LZ}$ be the length of the longest phrase (among random number, M_m, of phrases), while $\mathcal{H}_n$ is still the height of a DST built from n independently generated strings, as studied in Theorem* 2.1. *Then for all $k \geq 0$ and $\varepsilon > 0$*

$$\mathsf{Pr}\{\mathcal{H}_{(1+\varepsilon)\mu(m)} \leq k\} - \delta_m \leq \mathsf{Pr}\{\mathcal{H}_m^{LZ} \leq k\} \leq \mathsf{Pr}\{\mathcal{H}_{(1-\varepsilon)\mu(m)} \leq k\} + \delta_m. \tag{2.31}$$

In particular,

$$\mathbf{E}[\mathcal{H}_m^{LZ}] = \log_2(m/\log_2 m) + \sqrt{2\log_2(m/\log_2 m)} - \frac{1}{2}\log_2\log_2(m/\log_2 m) + O(1), \tag{2.32}$$

and most of the probability mass is concentrated either at k_1^{LZ} or $k_1^{LZ} - 1$, where

$$k_1^{LZ} = \left\lfloor \log_2(m/\log_2 m) + \sqrt{2\log_2(m/\log_2 m)} - \frac{1}{2}\log_2\log_2(m/\log_2 m) + \frac{1}{\log 2} - 1 \right\rfloor + 1$$

for large m.

Proof. We proceed as follows

$$\begin{aligned}
\mathsf{Pr}\{\mathcal{H}_m^{LZ} \leq k\} &= \sum_{n\geq 0} \mathsf{Pr}\{\mathcal{H}_n \leq k,\ M_m = n\} \\
&\leq \delta_m + \sum_{(1-\varepsilon)\mu(m)\leq n\leq(1+\varepsilon)\mu(m)} \mathsf{Pr}\{\mathcal{H}_n \leq k,\ M_m = n\} \\
&\leq \delta_m + \mathsf{Pr}\{\mathcal{H}_{(1-\varepsilon)\mu(m)} \leq k\},
\end{aligned}$$

where the last inequality is a consequence of the fact that $\mathcal{H}_m^{LZ}$ is a nondecreasing sequence with respect to m. The lower bound of (2.31) can be proved in a similar manner. The rest is a simple consequence of (2.31). ∎

3. Discussion and numerical results. We shall discuss the accuracy of the various asymptotic results and also numerically calculate the hitherto undetermined function $\Phi(\xi)$. We begin by making some general comments on how to use the asymptotic formulae.

It is most natural to view the problem as starting with a fixed (large) n and then varying k. We let k^* be the minimum integer such that $2^{k+1} - 1 \geq n$. More precisely, we set

$$k^* = \begin{cases} \log_2(n+1) - 1 & \text{if} \quad n+1 = \text{ power of 2,} \\ \lfloor \log_2(n+1) \rfloor & \text{if} \quad n+1 \neq \text{ power of 2,} \end{cases} \tag{3.1}$$

and note that h_n^k is only nonzero for $k \geq k^*$. If, say, $n = 100$, we have $k^* = 6$. Then $2^{k^*+1} - n = 28$ so that we are probably out of the range where the M-scale expansion applies. For $k = k^*$ we have $\xi = n2^{-k} = 25/16$, and this is well within the range $0 < \xi < 2$, where Theorem 2.1(iii) applies. By increasing k to $k^* + 1$, $k^* + 2, k^* + 3, \ldots$, we obtain the values $25/32$, $25/64$, $25/128, \ldots$ for ξ, so that ξ

TABLE 1
Comparison of $Q(z)$ and $Q_*(z)$.

z	$Q(z)$	$Q_*(z)$
0	.063716934676373540049220037 86	.0637169346763188
.05	.3901	.3215
.1	.4234	.3293
.15	.4753	.3414
.2	.5406	.3566
.25	.6130	.3735
.3	.6854	.3904
.35	.7508	.4056
.4	.8026	.4177
.45	.8359	.4255
.5	.8474	.4281
.55	.8359	.4255
.6	.8026	.4177
.65	.7508	.4056
.7	.6854	.3904
.75	.6130	.3735
.8	.5406	.3566
.85	.4753	.3414
.9	.4234	.3293
.95	.3901	.3215
1	.3786	.3188

becomes small rapidly and it thus may be desirable to use the WKB approximation (cf. section 5) and further replace A and Φ by their small ξ expansions, which we have derived explicitly. When k further increases to a significant fraction of n (e.g., $k = 20$), then we should use the expansion (ii) of Theorem 2.1, which applies on the α-scale (where $\alpha = n/(n-k) > 1$). When k further increases to a value close to n, such as 95, we can use the expansion that applies for $j = n - k$ fixed (cf. Theorem 2.1(i)). Of course, for $k \geq n - 1$, we have $h_n^k = 1$.

If we start with $n = 127$, then $k^* = 6$, which correspond to $M = 1$ and $\xi = 127/64$. Thus for $k = k^*$ we can use the M-scale result (cf. Theorem 2.1(iv)), but increasing k to $k^* + 1 = 7$ puts us well within the region $0 < \xi < 2$, where Theorem 2.1(iii) applies. If $n = 128$, then $k^* = 7$ and $2^{k^*+1} - n = 128$. This corresponds to $M = 128$ and $\xi = 1$, and this indicates the ξ-scale result should be used. Since $\xi = 1$ is not close to either 0 or 2, we must use the numerical value of $\Phi(1)$.

We define k_1 by (2.23) with $\ell = 1$. According to our analysis, as $n \to \infty$ the probability mass should be concentrating at the single point k_1 or the two points $k_1 - 1$, k_1 (with the former being more likely). If $n = 100$, we have $k_1(100) = 10$ so that $k_1 - k^* = 4$, and hence the "left-tail" really consists of only the four points with $k = 6 - 9$.

In using the asymptotic results that involve the periodic functions $Q(z)$ and $Q_*(z)$, we approximate them by the numerical value in (2.29), which is correct to 11 significant digits for both of them. Figure 4 and Table 1 show that the oscillations in Q_* occur only in the 12th significant digit, while those in Q are even smaller, occurring in the 25th digit. From the Fourier series for Q given by (2.20) we can analytically estimate the oscillations by using only the term with $m = 1$ and approximating $\sinh(a)$ by $e^a/2$, which yields

TABLE 2
Left-tail comparison.

n	k	M	h_n^k (exact)	h_n^k (2.22)
7	2	1	$7.81250(10^{-2})$	$8.06991(10^{-2})$
6	2	2	.312500	.322796
15	3	1	$1.27852(10^{-3})$	$1.29902(10^{-3})$
14	3	2	$1.02282(10^{-2})$	$1.03922(10^{-2})$
13	3	3	$4.09126(10^{-2})$	$4.15688(10^{-2})$
31	4	1	$2.36143(10^{-7})$	$2.38012(10^{-7})$
30	4	2	$3.77829(10^{-6})$	$3.80820(10^{-6})$
29	4	3	$3.02264(10^{-5})$	$3.04656(10^{-5})$
28	4	4	$1.61555(10^{-4})$	$1.62483(10^{-4})$
63	5	1	$5.62788(10^{-15})$	$5.65000(10^{-15})$
62	5	2	$1.80092(10^{-13})$	$1.80800(10^{-13})$
61	5	3	$2.88147(10^{-12})$	$2.89280(10^{-12})$
60	5	4	$3.07515(10^{-11})$	$3.08566(10^{-11})$
59	5	5	$2.46852(10^{-10})$	$2.46390(10^{-10})$

TABLE 3
Far right-tail comparison.

n	k	j	$1-h_n^k$ (exact)	$1-h_n^k$ (2.11)
10	8	2	$1.45519(10^{-11})$	$1.45519(10^{-11})$
	7	3	$2.98023(10^{-8})$	$3.72529(10^{-8})$
	6	4	$1.31922(10^{-5})$	$2.38419(10^{-5})$
	5	5	$1.63566(10^{-3})$	$5.08626(10^{-3})$
15	13	2	$4.03897(10^{-28})$	$4.03897(10^{-28})$
	12	3	$4.30134(10^{-23})$	$4.96308(10^{-23})$
	11	4	$1.05258(10^{-18})$	$1.52446(10^{-18})$
	10	5	$7.82575(10^{-15})$	$1.56125(10^{-14})$
20	18	2	$3.34096(10^{-52})$	$3.34096(10^{-52})$
	17	3	$1.57646(10^{-45})$	$1.75162(10^{-45})$
	16	4	$1.75253(10^{-39})$	$2.29589(10^{-39})$
	15	5	$6.09591(10^{-34})$	$1.00309(10^{-33})$
25	23	2	$8.23609(10^{-84})$	$8.23609(10^{-84})$
	22	3	$1.58906(10^{-75})$	$1.72723(10^{-75})$
	21	4	$7.32182(10^{-68})$	$9.05568(10^{-68})$
	20	5	$1.07178(10^{-60})$	$1.58259(10^{-60})$

$$\frac{Q(z)}{Q_0} \approx 1 - 2e^{-2a}\cos(2\pi z), \qquad a = \frac{2\pi^2}{\log 2},$$
$$= 1 - 3.678\ldots(10^{-25})\cos(2\pi z).$$

We note that both Q and Q_* are symmetric about the midpoint $z = 1/2$ and achieve their maxima here.

We first discuss the accuracy of the expansion for M fixed, which corresponds to case (iv) of Theorem 2.1, as defined in section 2. In Table 2, we consider values of n that are of the form $2^{k+1} - 1$, or slightly smaller, and various M in the range 1 to 5. From the table we see that the asymptotic formula is highly accurate and that for larger k, we can allow M to be larger and still obtain good agreement. We recall that the error term in (2.22) is $O(2^{-k}) = O(n^{-1})$. The results in Table 2 suggest that the numerical coefficient of the correction term is fairly small.

We next consider the far right tail (cf. Theorem 2.1(i)), where h_n^k is close to one.

TABLE 4
Right-tail comparison.

n	k	α	$1-h_n^k$ (exact)	$1-h_n^k(2.12)$
16	12	4	$2.99991(10^{-22})$	$3.74045(10^{-22})$
20	15	4	$6.09591(10^{-34})$	$7.21250(10^{-34})$
24	18	4	$2.42022(10^{-48})$	$2.77231(10^{-48})$
28	21	4	$1.88363(10^{-65})$	$2.11079(10^{-65})$
32	24	4	$2.87538(10^{-85})$	$3.17143(10^{-85})$

In Table 3 we consider $10 \le n \le 25$ for $2 \le j \le 5$. We tabulate the exact values of $1-h_n^k$ and the result in (2.11). When $j=2$ the agreement is excellent for all n, as it should be since (2.6) shows that (2.11) is not only asymptotic but exact! However, for $j=3$ the error is about 25% when $n=10$; it decreases to below 10% when $n=25$. The situation becomes worse when j increases.

The correction term to (2.11) is $O(n^{-1})$. The data in Table 3 suggest that the numerical coefficient in the correction term is fairly large, and increases with j. This is certainly consistent with the exact results in (2.7) and (2.8), which show that the correction factor is of the form $1-2/n$ and $1-5/n$ for $j=3$ and $j=4$, respectively. Note also that if, say, $n=20$ and $j=5$, it is not a priori clear whether we should use the $j=O(1)$ result or that for $\alpha=n/j$ fixed.

In Table 4 we test the accuracy of (2.12). We fix $\alpha=n/(n-k)=4$ and consider $16 \le n \le 32$. Note that since $\alpha>2$, we can use the infinite sum in (2.13) to calculate the integral I, which yields $I(4)=.17398\ldots$. When $n=16$ the error is about 25% and decreases to 10% when $n=32$. While this is consistent with a correction term of $O(n^{-1})$, the data again suggest that the coefficient in the error term is fairly large, relative to one.

In obtaining the asymptotic results summarized in (2.11)–(2.22), we have written them in the simplest possible form. However, from our analysis we can easily generate results that are more uniform and more numerically accurate. By more uniform we mean they apply to larger ranges of k than those in (2.11)–(2.22). For example, in the right tail we find that

$$1-h_n^k \sim 2^{-k^2/2}2^{-k/2}\tilde{F}_j(n), \tag{3.2}$$

where $\tilde{F}_j(n)$ is given precisely by (4.15) of section 4. For j fixed and $n\to\infty$ (4.15) reduces to (2.11), for $\alpha=n/j>1$ and $n\to\infty$ it reduces to (2.12), and the analysis in section 4 shows that it applies even for $n\to\infty$ with $k/n\to 0$ as long as $k/\log_2 n=\nu>1$. Thus, the approximation holds anywhere in the right tail and we call $2^{-k^2/2}2^{-k/2}\tilde{F}_j(n)$ the "uniform right tail" (URT) approximation for $1-h_n^k$. In Table 5 we consider $n=10$, 20, 30, and 100 and various values of $k=n-j$. We decrease k until the exact value of $1-h_n^k$ exceeds .5 (i.e., $h_n^k<.5$). For $n=10$, Table 5 shows that for $k=8$ and $k=7$, the exact and URT results agree to 6 decimal places. As k decreases from 6 to 4, the error slowly increases, but remains under 2%. For $k=3$ we have $1-h_n^k>.5$, but even then URT is accurate to within 12%. By now we are well outside of the right tail. For $n=20$ and $k\ge 6$, URT is accurate to within .3% and agrees with the exact result to 6 decimal places for $k\ge 10$. When $k=5$ we have $1-h_n^k$ becoming $O(1)$, but still URT is accurate to within 5%. For $k=4$ the asymptotic result exceeds 1, but then we are clearly outside of the right tail. Similar trends are apparent for $n=30$ and $n=100$. Here we need only consider relatively

TABLE 5
Uniform right-tail approximation.

n	k	j	$1-h_n^k$ (exact)	$1-h_n^k$ (3.2)
10	8	2	$1.45519(10^{-11})$	$1.45519(10^{-11})$
	7	3	$2.98023(10^{-8})$	$2.98023(10^{-8})$
	6	4	$1.31922(10^{-5})$	$1.31925(10^{-5})$
	5	5	$1.63566(10^{-3})$	$1.63632(10^{-3})$
	4	6	$6.09603(10^{-2})$	$6.13033(10^{-2})$
	3	7	.617309	.687647
20	18	2	$3.34096(10^{-52})$	$3.34096(10^{-52})$
	16	4	$1.75253(10^{-39})$	$1.75253(10^{-39})$
	14	6	$7.43000(10^{-29})$	$7.43000(10^{-29})$
	12	8	$5.87353(10^{-20})$	$5.87353(10^{-20})$
	10	10	$1.09764(10^{-12})$	$1.09764(10^{-12})$
	8	12	$5.19019(10^{-7})$	$5.19043(10^{-7})$
	6	14	$5.92196(10^{-3})$	$5.93731(10^{-3})$
	5	15	.143413	.148697
	4	16	.855618	>1
30	10	20	$2.52277(10^{-10})$	$2.52278(10^{-10})$
	9	21	$1.24549(10^{-7})$	$1.24553(10^{-7})$
	8	22	$2.56647(10^{-5})$	$2.56706(10^{-5})$
	7	23	$2.16191(10^{-3})$	$2.16562(10^{-3})$
	6	24	$7.08821(10^{-2})$	$7.25828(10^{-2})$
	5	25	.668201	.920224
100	13	87	$3.80919(10^{-13})$	$3.80919(10^{-13})$
	12	88	$3.99988(10^{-10})$	$3.99991(10^{-10})$
	11	89	$1.86584(10^{-6})$	$1.86592(10^{-6})$
	10	90	$3.81030(10^{-5})$	$3.81128(10^{-5})$
	9	91	$3.33434(10^{-3})$	$3.34291(10^{-3})$
	8	92	.115924	.122474
	7	93	.866855	>1

small values of k/n, since even here URT is accurate to 6 decimal places. We also note that in each data point in Table 5, URT *overestimates* $1-h_n^k$. This suggests that it may be an upper bound for $1-h_n^k$ (thus a lower bound for h_n^k) for points in the right tail.

Now we test the accuracy of the asymptotics for points where there is appreciable mass. According to our result, this corresponds to $k-\log_2 n \approx \sqrt{2\log_2 n}$. Our approximation here is that in (2.15) with $\Phi(\xi)$ replaced by its small ξ expansion (2.19). However, the analysis in section 4 shows that the error term in (2.19) is only smaller than the leading term by a factor $1/\log\xi$ (with possibly some $\log(-\log\xi)$ factors). While we could compute these, it proves more efficient to use the full result in (4.34) established in section 4. If we set $n2^{-k}=\xi$ in (4.34) we find that $2^{-k^2/2}2^{-k/2}\times$ (4.34) has the form $n\times$ [function of ξ], so that the matching condition in (5.40) may be refined to

$$\Phi(\xi)\sim\Phi_0(\xi),\qquad \xi\to 0^+, \tag{3.3}$$

where

$$\Phi_0(\xi)=\pi\xi\frac{1}{2\pi i}\int_{-i\infty}^{i\infty}\frac{2^{s(s-1)/2}Q(s)}{\Gamma(3-s)}e^{-s\log\xi}ds. \tag{3.4}$$

While the error in (2.19) is only about $1+O(1/|\log\xi|)$, we believe that in (3.3) the

error is about $1+O(\xi)$, though we have not explicitly calculated the correction. Now, (3.4) is much less insightful than (2.19), but it is much more accurate asymptotically and, as we show, numerically. The calculations in section 4 show that the leading term in a saddle point expansion of (3.4) as $\xi \to 0^+$ yields precisely (2.19). We also note that the integrand in (3.4) is an entire function of s, as both $Q(\cdot)$ and $1/\Gamma(\cdot)$ are.

In Appendix B we obtain the following representation for Φ_0 as an infinite sum:

$$\Phi_0(\xi) = \frac{1}{4\xi}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{5m/2}\exp(-2\xi 2^{-m}). \tag{3.5}$$

For any fixed $\xi > 0$ this sum converges very rapidly as $m \to \infty$, and even more rapidly as $m \to -\infty$, due to the exponential factor. It is thus useful for numerical calculations. The asymptotics as $\xi \to 0^+$ are difficult to obtain from (3.5), due to the alternating sum. However, we can easily see that $\Phi_0(\xi) = o(\xi^N)$ for all $N \geq 1$. Indeed, we expand the exponent in Taylor series and exchange the order of the two summations. We have

$$\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{5m/2}2^{-mL} = 0 \tag{3.6}$$

for any integer L, as can be seen by the antisymmetry of the summand with respect to the shift $m \to 5-2L-m$. It thus follows that Φ_0 vanishes to all algebraic orders as $\xi \to 0^+$. A rough estimate of the leading behavior can be obtained by noting that for $\xi \to 0^+$ the important terms in the sum are those where $\xi 2^{-m} = O(1)$ so that $m \approx \log_2 \xi$. There the magnitude of the summand is roughly $2^{-m^2/2} = \exp\left[-\frac{1}{2}(\log 2)(\log_2 \xi)^2\right]$, which is the same as the dominant factor in (2.19).

For numerical calculations we use (3.5) to approximate Φ and A in (2.15), for $\xi \to 0^+$. Setting $z_0 = \Phi_0(\xi)$,

$$z_1 = (\xi\Phi_0)'(\xi) = -\frac{1}{2}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{3m/2}\exp(-2\xi 2^{-m}) \tag{3.7}$$

and

$$z_2 = \xi(\xi\Phi_0)''(\xi) = \xi\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{m/2}\exp(-2\xi 2^{-m}), \tag{3.8}$$

we thus obtain the asymptotic form of the (improved) WKB approximation as (cf. (2.15))

$$\sqrt{1+z_2}e^{-nz_0}e^{-z_1}. \tag{3.9}$$

We denote the above "refined" approximation by REF. In Table 6 we compare (3.9) to the exact values, for the few points where there is appreciable (numerical) mass. We also give the values of $\xi = n2^{-k}$, since (3.9) assumes that $\xi \to 0^+$. We consider n in the range $30 \leq n \leq 100$.

When $n = 40$ and $k = 5$ we have $\xi = 1.25$, and here (3.9) overestimates the true value by a factor of about 3. However, here we would not expect to be able to use the

TABLE 6
Refined approximation in the central regime.

n	k	ξ	h_n^k (exact)	h_n^k (3.9)
30	5	.938	.3318	.3921
	6	.469	.9291	.9293
	7	.234	.9978	.9979
	8	.117	1.0000	1.0000
40	5	1.25	.0206	.0747
	6	.625	.7134	.7220
	7	.313	.9851	.9852
	8	.156	.9997	.9997
	9	.078	1.0000	1.0000
50	6	.781	.3723	.4086
	7	.391	.9438	.9440
	8	.195	.9985	.9986
	9	.098	1.0000	1.0000
60	6	.938	.1051	.1541
	7	.469	.8518	.8532
	8	.234	.9948	.9948
	9	.117	.9999	.9999
70	6	1.09	.0122	.0375
	7	.547	.6972	.7035
	8	.273	.9857	.9857
	9	.137	.9998	.9998
	10	.068	1.0000	1.0000
80	6	1.25	.0004	.0060
	7	.625	.4950	.5114
	8	.313	.9673	.9674
	9	.156	.9993	.9993
	10	.078	1.0000	1.0000
90	6	1.41	.0000	.0007
	7	.703	.2902	.3177
	8	.352	.9350	.9351
	9	.176	.9984	.9984
	10	.088	1.0000	1.0000
100	7	.781	.1331	.1644
	8	.391	.8841	.8846
	9	.195	.9967	.9967
	10	.098	1.0000	1.0000

small ξ approximation to (2.15), but rather should compute $\Phi(1.25)$ numerically and use this value instead. We will discuss the numerical computation of Φ shortly. When $k = 6$ we have $\xi = .625$ which is certainly not small, but nevertheless (3.9) is accurate to about 1%. For $k = 7, 8$, and 9 the two results agree to 3 or 4 decimal places. For each n we increased k until h_n^k is one to 4 decimal places. Note that by the time k reaches this value, we are fairly well in the right tail, and there we showed that URT is highly accurate.

As we increase n, Table 6 shows that REF is not accurate whenever $\xi \geq 1$, but gives reasonable approximations for $\xi \leq .8$ and is very good for $\xi \leq .5$. Thus the small ξ approximation to (2.15) in (3.9) is quite robust. The numerical computation of Φ is most difficult when ξ is small, so that the numerical and asymptotic methods complement each other, as is often the case in applied problems.

We note that it was essential that we used the refined approximation in (3.9), rather than the leading term result in (2.19), which we denote by $\Phi_{LT}(\xi)$. In Table 7 we compare the values of Φ_0 and Φ_{LT}, starting at $\xi = .1$ and decreasing to $\xi = 10^{-14}$. When $\xi = .1$ we have $\Phi_{LT}/\Phi_0 = 7.31$ and even if $\xi = 10^{-14}$, this ratio is 1.79, which

TABLE 7
Different approximations of $\Phi(\xi)$.

ξ	$\Phi_{LT}(\xi)$	$\Phi_0(\xi)$	Φ_{LT}/Φ_0
.1	$3.99(10^{-6})$	$5.45(10^{-7})$	7.31
10^{-2}	$3.58(10^{-15})$	$7.54(10^{-16})$	4.75
10^{-3}	$6.62(10^{-28})$	$1.78(10^{-28})$	3.72
10^{-4}	$2.95(10^{-44})$	$9.33(10^{-45})$	3.16
10^{-6}	$1.26(10^{-87})$	$4.94(10^{-88})$	2.55
10^{-8}	$5.12(10^{-145})$	$2.29(10^{-145})$	2.23
10^{-10}	$2.81(10^{-216})$	$1.38(10^{-216})$	2.03
10^{-12}	$2.62(10^{-301})$	$1.38(10^{-301})$	1.89
10^{-14}	$4.89(10^{-400})$	$2.73(10^{-400})$	1.79

is far from the theoretical value (=1) as $\xi \to 0^+$. When $\xi = 10^{-14}$, both values are about 10^{-400}. By now we are so far in the right tail that we would never use the WKB result in the first place.

Finally, we discuss the accumulation of the probability mass at one or two points as $n \to \infty$, as our results predict. We define ℓ and $\beta = \beta(n)$ as in (2.23). We clearly have $0 \le \beta < 1$. It proves easiest to discuss the limit $n \to \infty$ along subsequences $n(i)$ that correspond to β nearly constant.

If we take a fixed $0 < \beta < 1$, then (2.25) predicts that the masses at $\ell = 0, 1, 2$ (corresponding to $k = k_1 - 1, k_1, k_1 + 1$) are approximately

$$
\begin{aligned}
m_0 &\approx \exp\left[-\frac{Q_*}{2\sqrt{\log 2}} e^{-\frac{1}{\log 2}} \frac{2^{\beta\sqrt{2\log_2 n}} 2^{-\beta^2/2} e^{\beta}}{(2\log_2 n)^{3/4}}\right], \\
m_1 &\approx 1 - \frac{Q_*}{2\sqrt{\log 2}} e^{-\frac{1}{\log 2}} \frac{2^{(\beta-1)\sqrt{2\log_2 n}} 2^{-(\beta-1)^2/2} e^{\beta-1}}{(2\log_2 n)^{3/4}}, \\
m_2 &\approx \frac{Q_*}{2\sqrt{\log 2}} e^{-\frac{1}{\log 2}} \frac{2^{(\beta-1)\sqrt{2\log_2 n}} 2^{-(\beta-1)^2/2} e^{\beta-1}}{(2\log_2 n)^{3/4}}.
\end{aligned}
\tag{3.10}
$$

Here $m_0 = h_n^{k_1-1} - h_n^{k_1-2} \sim h_n^{k_1-1}$, $m_1 = h_n^{k_1} - h_n^{k_1-1} \sim h_n^{k_1} (\sim 1)$, and $m_2 = h_n^{k_1+1} - h_n^{k_1} \sim 1 - m_1$. Thus all the mass should concentrate at $k = k_1$ as long as β remains bounded from 0 and 1. Even as $\beta \to 1^-$ we will have $m_2 \to 0$ and $m_1 \to 1$, due to the factor $(\log n)^{-3/4}$ in m_2 and $1 - m_1$.

Next consider subsequences along which $\beta \to 0$. As before, we define $\Delta(n)$ by (2.27). Using (2.27) in (2.25) (or (3.10)) we find that if $n \to \infty$ in such a way that $\Delta(n)$ is bounded, then $h_n^{k_1-1}$ is $O(1)$ and < 1. Thus, for such sequences there is mass at two points, corresponding to $k = k_1 - 1$ and $k = k_1$.

To construct such sequences $\Delta(n)$ we consider the equation

$$
N + \frac{3}{2}\frac{\log_2(\sqrt{2\log_2 n})}{\sqrt{2\log_2 n}} = \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2},
\tag{3.11}
$$

where N is an integer $N \ge 3$. For any N we can solve (numerically) the implicit relation (3.11) and generate a sequence of solutions, which we denote by $n_*(N)$. In view of the definition (2.24) of $\beta(n)$ we see that when $n = n_*(N)$ we have $\Delta(n) = 0$. Now, $n_*(N)$ is generally not an integer, but $\lfloor n_*(N) \rfloor$ or $\lfloor n_*(N) \rfloor + 1$ should correspond

to a local minimum of the sequence $|\Delta(n)|$. In Table 8 we compute the first few $n_*(N)$ and also give the sequence of local minima of $|\Delta(n)|$. We see that $\min|\Delta(n)|$ is the integer closest to $n_*(N)$ for all $3 \le N \le 15$. In Figure 4 we plot the sequence $\Delta(n)$. From (2.27) and the figure we see that the upper envelope grows (roughly) like $\sqrt{\log n}$ and the lower envelope like $-\log\log n$. The sequence increases, crosses zero, and then jumps back down. The jump corresponds to the fractional part β changing from 1^- to 0^+, which occurs by increasing n. The maxima of $\Delta(n)$ corresponds to $\beta \approx 1$, while the minima have $\beta \approx 0$. Note also that β is small at the minima of $|\Delta(n)|$.

We can easily solve (3.11) asymptotically as $N \to \infty$, and this yields the estimate

$$n_*(N) = 2^{\,N-\sqrt{2N}}\sqrt{2N}\,2^{3/2}e^{-1} \times \left[1 + \frac{\log 2}{\sqrt{2N}}\left(\frac{1}{2}\log_2(\sqrt{2N}) - 1\right) + O\left(\frac{\log N}{N}\right)\right]. \tag{3.12}$$

Using (3.12) to compute $\beta(n)$ and then $\Delta(n)$ for $N \to \infty$, we find that for $N \to \infty$ and $n - n_* = O(1)$

$$\Delta(n) \sim \frac{1}{n_*}(n - n_*)\left\{\frac{3}{4}\frac{\log_2(\sqrt{2\log_2 n_*})}{\log(n_*)} - \frac{3}{4\log 2}\frac{1}{\log(n_*)} + \frac{\sqrt{2}}{(\log 2)^{3/2}}\left(\sqrt{\log n_*} + \sqrt{\frac{\log 2}{2}} - \frac{1}{2\sqrt{\log n_*}}\right)\right\}. \tag{3.13}$$

This shows that as $n \to \infty$, $\Delta(n)$ is not only bounded but $\Delta(n) \to 0$ with (roughly) $\Delta(n_*(N)) = O(n^{-1})$. Thus, along sequences such as $n = \lfloor n_*(N)\rfloor$ or $n = \lfloor n_*(N)\rfloor + 1$, the mass m_0 should be

$$\exp\left[-\frac{Q_*}{2\sqrt{\log 2}}\exp\left(-\frac{1}{\log 2}\right)\right] \approx .990998897\ldots. \tag{3.14}$$

Even though this is *asymptotically* $O(1)$ and < 1, the small value of Q_* (cf. (2.29)) shows that *numerically* most of the mass will be at $k_1 - 1$, with the remaining mass at k_1. Expression (3.14) has some fluctuations, but these are very small in view of Table 1.

If we choose a sequence $n(i)$ such that $\beta(n) \to 1^-$, then the mass becomes concentrated at $\ell = 1$ ($k = k_1$), and the mass at $\ell = 2$ is

$$m_2 \sim \frac{Q_*}{2\sqrt{\log 2}}e^{-\frac{1}{\log 2}}\frac{1}{(2\log_2 n)^{3/4}}. \tag{3.15}$$

The right side of (3.15) approaches zero slowly, but in view of the small numerical value of $Q_* e^{-1/(\log 2)}\ 2^{-1}(\log 2)^{-1/2} \approx .009041857\ldots$, m_2 will be numerically small even for moderate values of n. We also note that choosing n to make $\beta \approx 1$ minimizes the mass at $\ell = 0$ ($k = k_1 - 1$). From an asymptotic point of view, the optimal way to choose $n(i)$ to get most rapid convergence to mass at the single point k_1 is to minimize $|\beta(n) - 1/2|$. This makes m_0 and m_2 both small. However, for moderate values of n, it is preferable to choose $\beta \approx 1$, in order that the factor $2^{\sqrt{2\log_2 n}}$ in the exponent in m_0 in (3.10) compensates for the numerically small value of Q_*. Thus for moderate n it is more essential to minimize m_0 rather than m_2 to obtain $m_1 \to 1$. For very large values of n it becomes desirable to minimize m_2, which requires $\beta \approx 0$ (but in such a way that $\Delta(n) \to -\infty$).

TABLE 8
Solutions $n_*(M)$ of (3.11).

N	$n_*(N)$	$\min \lvert\Delta(n)\rvert$
3	3.52	4
4	6.64	7
5	12.08	12
6	21.78	22
7	39.14	39
8	70.38	70
9	126.69	127
10	228.45	228
11	412.79	413
12	747.38	747
13	1355.94	1356
14	2464.88	2465
15	4489.33	4489

The numerical results in Table 6 show that for $k = 40$, 50, 60, 70, 80, 90, and 100, the most mass occurs, respectively, at $k = 6$, 7, 7, 7, 7, 8, and 8. The respective masses, obtained from $h_n^k - h_n^{k-1}$, are about .69, .57, .75, .69, .49, .64, and .75. The respective values of k_1 are 8, 9, 9, 9, 9, 10, and 10, so that the most mass is at $k_1 - 2$, not k_1! This apparent discrepancy, however, can easily be understood from the asymptotic analysis. In order to make the mass for $k \le k_1 - 2$ ($\ell \le -1$) about .1, even with an optimal value of $\beta \approx 1$, we need n large enough so that

$$.1 = \exp\left[-\frac{Q_*}{2\sqrt{\log 2}}\frac{1}{(2\log_2 n)^{3/4}}2^{2\sqrt{2\log_2 n}}2^{-2}e^{2-1/\log 2}\right]$$

so that $n \approx 22,123$. To make this mass .01, we need $n \approx 252,025$. To make the mass in the range $k \le k_1 - 1$ ($\ell \le 0$) .1 and .01 we need to have $n \approx 1.364(10^{10})$ and $n \approx 2.340(10^{12})$, respectively. This is well beyond the range of the values in Tables 2–6. Our asymptotic results predict that the mass should migrate from $k_1 - 2$ to $k_1 - 1$ and eventually to k_1, at least for values of β bounded away from zero. Note, however, that the numerical mass is well predicted by our asymptotic expansions, even for moderate values of n.

To better see the convergence of mass to one or two points, it is best to consider special subsequences $n(i)$ in order to avoid the oscillations caused by the appearance of $\beta(n)$. In Table 9, we consider the first few n which correspond to local minima of $|\Delta(n)|$, as given in Table 8. For these we give the exact masses $m_{-1} = h_n^{k_1-2} - h_n^{k_1-3}$, m_0 and m_1. As previously discussed the theory predicts that m_0 and m_1 will be close to .991 and .009. From Table 9 we see only that there is a gradual migration of mass from $k_1 - 2$ to $k_1 - 1$. However, even at $n = 127$ most of the mass is still at $k_1 - 2$.

In Table 10 we choose a sequence $n(i)$ so as to minimize $1 - \beta(n)$. This is the optimal way to "push" the mass out of $k = k_1 - 2$ and into $k = k_1$ for moderate n. The table clearly shows a gradual migration of mass from $k_1 - 2$ into $k_1 - 1$, and an increase of mass at k_1, though the latter is still below .05 when $n = 264$. Our asymptotic results predict that the migration from $k_1 - 1$ to k_1 along this subsequence would occur when $n \approx 10^{10}$, which is well beyond the range where it is feasible to do numerical calculations.

TABLE 9
Probability mass at k_1 only.

n	k_1	m_{-1}	m_0	m_1
7	5	.7715	.1455	.0049
12	6	.7301	.1530	.0067
22	7	.7235	.2122	.0110
39	8	.7120	.2446	.0124
70	9	.6849	.2885	.0141
127	10	.6345	.3461	.0162

TABLE 10
Probability mass at k_0 and k_1.

n	k_1	m_{-1}	m_0	m_1
7	5	.7715	.1455	.0049
13	6	.7280	.2192	.0117
24	7	.6643	.3028	.0190
44	8	.5801	.3903	.0261
80	9	.4946	.4723	.0320
145	10	.4129	.5495	.0369
264	11	.3272	.6297	.0423

We conclude by discussing the numerical computation of $\Phi(\xi)$. We define

$$\Phi_{NUM}(\xi;k) \equiv -\frac{1}{n}\log(h_n^k), \qquad \xi = n2^{-k}.$$

Our results predict that for each fixed $0 < \xi \le 2$, $\Phi_{NUM}(\xi;k)$ should approach $\Phi(\xi)$ as $k \to \infty$. In Figure 5 we sketch Φ_{NUM} for $k = 6$. The limit function $\Phi(\xi)$ appears smooth except at $\xi = 0$ and $\xi = 2$, where our analytic results apply.

When $\xi = 2$ we have the theoretical value exactly, as $\Phi(2) = 1 - \log(\sqrt{2C_0}) = .55946\ldots$. Note that $n = 2^{k+1} - 1$ corresponds to $\xi = 2 - 2^{-k}$. In view of (2.17) we should have, as $k \to \infty$,

$$\Phi(2 - 2^{-k}) = \Phi(2) - \frac{k}{2}2^{-k} - \frac{1}{4}\log(2C_0)2^{-k} + o(2^{-k}). \tag{3.16}$$

In Table 11 we give $\Phi_{NUM}(2-2^{-k};k) = -\log(h^k_{2^{k+1}-1})/(2^{k+1}-1)$ for k in the range [2,7]. We compare Φ_{NUM} to both $\Phi(2)$ and the 3-term approximation in (3.16). This shows that Φ_{NUM} is indeed converging to the theoretical value.

To summarize, we have shown that the asymptotic results provide, on the whole, very good approximations to h_n^k. To achieve this, however, it is sometimes necessary to obtain more uniform and/or higher order results than those in (2.11)–(2.22). The accumulation of mass at $k = k_1$ is not in good agreement with the numerical results, but the asymptotics explain this and estimate the size of n necessary for the mass to migrate to k_1.

4. The right-tail analysis. In this section, we derive parts (i) and (ii) of Theorem 2.1, that is, the far right-tail approximation (2.11) and the right-tail asymptotic expansion (2.12).

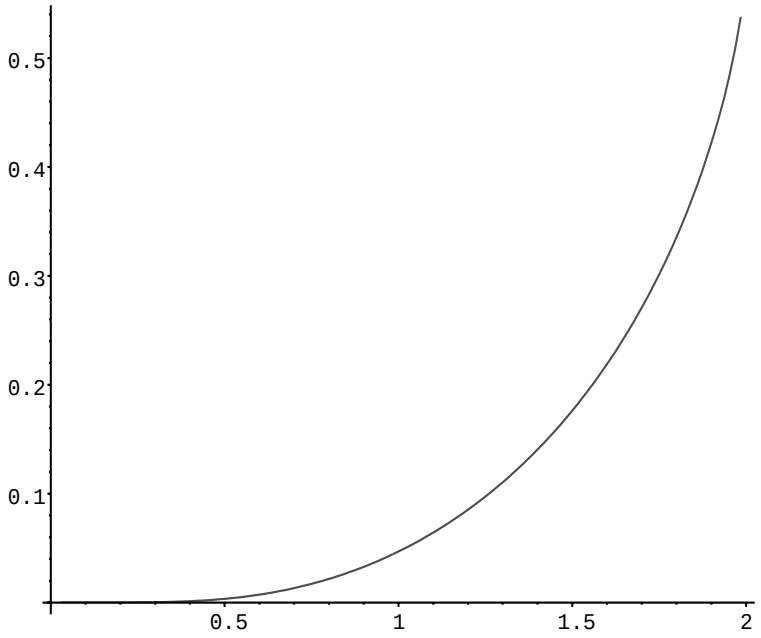


FIG. 5. *The functions* $\Phi_{NUM}(\xi;k)$ *versus* ξ *for* $k=6$.

TABLE 11
Numerical evaluation of $\Phi(2)$.

k	$\Phi_{NUM}(2-2^k)$	$\Phi(2)$	3-term approx.
2	.3642	.5595	.2544
3	.4441	.5595	.3444
4	.4922	.5595	.4207
5	.5208	.5595	.4744
6	.5375	.5595	.5091
7	.5472	.5595	.5304

4.1. The far right-tail analysis. Since we analyze h_n^k when the distribution is asymptotically close to one, it is convenient to set $j=n-k$ and

$$h_n^k = 1 - H_n^k, \qquad H_n^k = H_n^{n-j} = \mathcal{L}_j(n). \tag{4.1}$$

Using (4.1) in (2.2) we obtain

$$2^n\mathcal{L}_j(n+1) = 2\sum_{\ell=0}^{j-2}\binom{n}{\ell}\mathcal{L}_{j-\ell}(n-\ell) - \sum_{i=n-j+2}^{j-2}\binom{n}{i}\mathcal{L}_{i+j-n}(i)\mathcal{L}_{j-i}(n-i) \tag{4.2}$$

for $n \geq j$. Here we have multiplied (2.2) by 2^n and used the fact that $h_n^k = 1$ for $k \geq n-1$, which is equivalent to $\mathcal{L}_j(n) = 0$ for $j \leq 1$. We also note that if $k+1 \geq n$, $h_n^k = 1$ is an exact solution to (2.2), since every term in the sum lies in this range.

We have thus decomposed the nonlinear right side of (2.2) into a linear part and a nonlinear part, corresponding to the two sums in (4.2). For $n \geq 2j-3$ the second

sum is void and the equation is *exactly* linear. The initial condition (2.3) becomes

$$\begin{aligned} \mathcal{L}_j(j) &= 0, \qquad j = 0, 1, \\ \mathcal{L}_j(j) &= 1, \qquad j \geq 2. \end{aligned} \tag{4.3}$$

From (4.2) we can compute $\mathcal{L}_j(n)$ exactly for small values of $j \geq 2$. For example, setting $j = 2$ we obtain from (4.2), subject to $\mathcal{L}_2(2) = 1$,

$$\mathcal{L}_2(n) = 2^{-n^2/2} 2^{3n/2} 2^{-1}, \quad n \geq 2, \tag{4.4}$$

and this proves (2.6). Letting $j = 3$ in (4.2) we find that

$$\mathcal{L}_3(n) = \left(\frac{n}{8} - \frac{1}{4} \right) 2^{-n^2/2} 2^{5n/2}, \qquad n \geq 4, \tag{4.5}$$

which is equivalent to (2.7). Note that (4.5) remains valid if $n = 3$ since $\mathcal{L}_3(3) = 1$ by (4.3).

Next we discuss the asymptotics as $n \to \infty$. For any fixed j and $n \to \infty$, we can conclude that

$$\mathcal{L}_j(n) \sim 2^{-n^2/2} 2^{(j-1/2)n} n^{j-2} C(j). \tag{4.6}$$

To compute $C(j)$ we use (4.6) in (4.2) to obtain

$$\begin{aligned} 2^n C(j) \cdot (n+1)^{j-2} 2^{-n^2/2} 2^{-n} 2^{-1/2} 2^{(j-1/2)n} 2^{j-1/2} \\ \sim 2 \sum_{\ell=0}^{j-2} \binom{n}{\ell} C(j-\ell) (n-\ell)^{j-\ell-2} 2^{-(n-\ell)^2/2} 2^{(j-\ell-1/2)(n-\ell)}. \end{aligned} \tag{4.7}$$

Noting that $\binom{n}{\ell} n^{-\ell} \sim \frac{1}{\ell!}$ and $(n-\ell)^{j-\ell-2} \sim n^{j-2} n^{\ell}$ as $n \to \infty$ the leading terms in (4.7) yield

$$2^{j-1} C(j) = 2 \sum_{\ell=0}^{j-2} \frac{C(j-\ell)}{\ell!} 2^{\ell^2/2} 2^{\ell/2} 2^{-\ell j}, \qquad j \geq 2. \tag{4.8}$$

In view of (4.4) and (4.6) we have $C(2) = 1/2$. Solving (4.8) yields

$$C(j) = 2^{-j^2/2} 2^{j/2} \frac{1}{(j-2)!},$$

which completes the derivation of the asymptotic formula for $1 - h_n^k = H_n^k = \mathcal{L}_j(n)$ in (2.11).

4.2. The right-tail analysis. We next consider n and $j = n - k$ simultaneously large and such that $0 < k/n < 1$, so that we are in the range of Theorem 2.1(ii). We will show that the asymptotic relation (4.6) ceases to be valid if j is as large as $O(n)$. Note also that $j = n - k$ can be as large as $n - \log_2 n$. From our consideration of small values of j, it is easy to see that $F_j(n)$ has the form

$$F_j(n) = \tilde{F}_j(n) + O(\rho^n), \qquad n \geq 2j - 3, \tag{4.9}$$

where $\rho < 1$ and $\tilde{F}_j(n)$ is a polynomial of degree $j - 2$. The exponentially small term corresponds to terms that are $O(4^{-n}, 8^{-n}, 16^{-n}, \ldots)$, modulo some factors algebraic

in n, as $n \to \infty$. Our calculations also showed that $\tilde{F}_j(n)$ depends on the initial conditions (4.3) only through the value $\mathcal{L}_2(2) = 1$. We then used the fact that $\mathcal{L}_3(3) = 1$ to conclude that $c_1 = 0$, which implies that there are no terms of order $O(2^{-n})$ in the exponentially small term. Then the terms proportional to 4^{-n} are completely determined by the value of c_2, which follows from $\mathcal{L}_4(4) = 1$.

For the region $n/j > 2$ in the (j, n) plane the problem (4.2) is exactly linear. For $1 < n/j < 2$ the nonlinear terms are present, but as long as $\mathcal{L}_j(n)$ is asymptotically small (i.e., h_n^k is asymptotically close to 1), the nonlinear term is small compared to the linear term(s). Thus in the right tail we can drop the second sum in (4.2) and replace $=$ by $\sim$. Also, the exponentially small part of the solution $F_j(n)$ depends on the values of $F_j(n)$ when $n/j = 2$, which in turn depends on the initial condition(s) (4.3). The initial data propagates from $n/j = 1$ to the range $n/j > 2$ via the region where the nonlinear term is present. However, these terms are exponentially small compared to $\tilde{F}_j(n)$. This discussion shows that

$$F_j(n) \sim \tilde{F}_j(n) \tag{4.10}$$

as long as j is such that $\mathcal{L}_j(n)$ is asymptotically small.

We next compute $\tilde{F}_j(n)$ for arbitrary j and use the result to obtain the asymptotics of $\mathcal{L}_j(n)$ (and hence h_n^k) for $n, j \to \infty$ with n/j fixed and $n/j > 1$. This implies that $k, n \to \infty$ at the same rate and $0 < k/n < 1$.

Since $\tilde{F}_j(n)$ is a polynomial in n of degree $j-2$ we can write

$$\tilde{F}_j(n) = \binom{n}{j-2} + d_1(j)\binom{n}{j-3} + d_2(j)\binom{n}{j-4} + \cdots, \tag{4.11}$$

where the sum truncates after a finite number of terms. Here we have set $d_0(j) = 1$, which follows from (4.6). For a fixed j and $n \to \infty$ the successive terms in (4.11) are asymptotically smaller by factors of n^{-1}. However, we will show that this is no longer true if n and j are both large. Using (4.11) yields

$$\begin{aligned} 2^{j-2}\left[\binom{n+1}{j-2} + d_1(j)\binom{n+1}{j-3} + d_2(j)\binom{n+1}{j-4} + \cdots\right] \\ = \sum_{\ell=0}^{j-2}\binom{n}{\ell}\left[\binom{n-\ell}{j-\ell-2} + d_1(j-\ell)\binom{n-\ell}{j-\ell-3}\right. \\ \left. + d_2(j-\ell)\binom{n-\ell}{j-\ell-4} + \cdots\right]. \end{aligned} \tag{4.12}$$

Using the identities

$$\begin{aligned} \binom{n+1}{m} &= \binom{n}{m} + \binom{n}{m-1}, \\ \binom{n}{\ell}\binom{n-\ell}{j-\ell-m} &= \binom{n}{j-m}\binom{j-m}{\ell} \end{aligned}$$

and comparing coefficients of $\binom{n}{j-m}$ in (4.12) for $m \geq 2$, we are led to the recurrences

$$2^{j-2}[1 + d_1(j)] = \sum_{\ell=0}^{j-3} d_1(j-\ell)\binom{j-3}{\ell} \tag{4.13}$$

and

$$2^{j-2}[d_m(j)+d_{m+1}(j)] = \sum_{\ell=0}^{j-m-3} d_{m+1}(j-\ell)\binom{j-m-3}{\ell}, \quad m \geq 1. \tag{4.14}$$

From (4.13) it follows that $d_1(j) = -2$, which is independent of j. We then find from (4.14) that $d_m(j)$ is independent of j for all $m \geq 1$, and $d_m(j) = d_m$ satisfies

$$d_m + d_{m-1} = 2^{-m} d_m$$

so that $d_m = (-1)^m \prod_{N=1}^{m} \left(1-2^{-N}\right)^{-1}$ and hence

$$\tilde{F}_j(n) = \sum_{m=0}^{j-2} (-1)^m \binom{n}{j-2-m} \prod_{N=1}^{m} (1-2^N)^{-1}. \tag{4.15}$$

We can easily check that this agrees with our previous results for $2 \leq j \leq 4$.

Now consider the binomial coefficient in (4.15) in the limit $n, j \to \infty$ with m and $\alpha \equiv n/j$ fixed and α rational. Using Stirling's formula we obtain

$$\begin{aligned} \frac{n!}{(n+m-j+2)!}\frac{1}{(j-2-m)!} &\sim \frac{n!}{(n-j+2)!}\frac{1}{(n-j)^m}\frac{j^m}{(j-2)!} \\ &\sim \frac{n^n}{(n-j)^{n-j}j^j}\frac{j^{3/2}\sqrt{n}}{(n-j)^{5/2}}\left(\frac{1}{\alpha-1}\right)^m \frac{1}{\sqrt{2\pi}}. \end{aligned} \tag{4.16}$$

It follows that

$$\tilde{F}_j(n) \sim \frac{n^n}{(n-j)^{n-j}j^j}\frac{j^{3/2}\sqrt{n\pi}}{\sqrt{2}(n-j)^{5/2}} I(\alpha), \tag{4.17}$$

where

$$I(\alpha) \equiv \frac{1}{\pi}\sum_{m=0}^{\infty} \frac{(-1)^m}{(\alpha-1)^m}\left[\prod_{\ell=1}^{m}(1-2^{-\ell})\right]^{-1}. \tag{4.18}$$

The sum in (4.18) is absolutely convergent for $\alpha > 2$. We can extend its range of validity by writing the product as $\prod_{\ell=1}^{m}(\cdots) = \prod_{\ell=1}^{\infty}(\cdots)/\prod_{\ell=m+1}^{\infty}(\cdots)$ and subtracting $\prod_{\ell=1}^{\infty}(\cdots)$. The resulting sum will converge for $\alpha > 3/2$, but we will need to know the behavior of $\tilde{F}_j(n)$ as $n \to \infty$ for all $\alpha = n/j > 1$. Thus we need to obtain a different representation for $\tilde{F}_j(n)$, which will be more useful for asymptotic analysis.

We begin by considering the function

$$A(z) = \prod_{\ell=1}^{\infty} \exp\left(\frac{1-2^{\ell z}}{\ell(2^\ell-1)}\right), \qquad \Re(z) < 1. \tag{4.19}$$

From (4.19) we can easily show that $A(\cdot)$ satisfies the functional equation

$$A(z+1) = A(z)[1-2^z], \tag{4.20}$$

which can be used to analytically continue $A(\cdot)$ into the range $\Re(z) > 1$. Observe that

$$A(-m) = \prod_{N=1}^{m}(1-2^{-N})^{-1} = \prod_{\ell=1}^{\infty}\exp\left(\frac{1-2^{-m\ell}}{\ell(2^\ell-1)}\right).$$

Indeed, the above follows from

$$\prod_{\ell=1}^{\infty}\exp\left(\frac{1-2^{-m\ell}}{\ell(2^\ell-1)}\right)=\exp\left(\sum_{\ell=1}^{\infty}\frac{1}{\ell}2^{-\ell}\frac{1-2^{-\ell m}}{1-2^{-\ell}}\right)$$
$$=\exp\left(\sum_{N=1}^{m}\sum_{\ell=1}^{\infty}2^{-N\ell}\frac{1}{\ell}\right)$$
$$=\exp\left(-\sum_{N=1}^{m}\log(1-2^{-N})\right)=\prod_{N=1}^{m}(1-2^{-N})^{-1}.$$

Then, we can rewrite the sum in (4.15) as

$$\tilde{F}_j(n)=\sum_{\ell=0}^{j-2}(-1)^\ell e^{\pi ij}\binom{n}{\ell}A(\ell+2-j) \tag{4.21}$$
$$=\frac{n!}{2\pi i}(-1)^{n+j}\int_{\frac12-i\infty}^{\frac12+i\infty}\frac{\Gamma(z+j-n-2)}{\Gamma(z+j-1)}A(z)dz,$$

where $0<\Re(z)<1$ on the contour of integration. The last inequality is basically Rice's formula (cf. [7, 28]), but we can derive it directly. Indeed, in the region $\Re(z)<1$ the integrand has simple poles at $z=0,-1,\dots,-j+1,-j+2$ and $\mathrm{Res}[z=-m]=\frac{(-1)^{m-j+n}}{(j-m-2)!}\,\frac{A(-m)}{(n+m-j-2)!}$, $m\ge 0$, where $\mathrm{Res}[z=A]$ stands for the residue at A for the function under the integral. Next, we close the contour in the left half-plane and the integral is equal to the finite residue sum, which is the same as (4.15).

Noting that

$$\sum_{\ell=1}^{\infty}\frac{1-2^{\ell z}}{\ell(2^\ell-1)}=\sum_{\ell=1}^{\infty}\frac{2^{-\ell}}{\ell}\sum_{N=0}^{\infty}2^{-N\ell}-\sum_{\ell=1}^{\infty}\frac{2^{\ell(z-1)}}{\ell}\sum_{N=0}^{\infty}2^{-N\ell}$$
$$=-\sum_{N=0}^{\infty}\log(1-2^{-N-1})+\sum_{N=0}^{\infty}\log(1-2^{z-N-1}),$$

we find that the expression in (4.19) becomes

$$A(z)=\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-1}\prod_{m=1}^{\infty}(1-2^{z-m}), \tag{4.22}$$

which applies for all z, and thus gives the analytic continuation of (4.19) into the half-plane $\Re(z)>1$. Expression (4.22) also shows that $A(z)$ is an entire function, with zeros at $z=1,2,3,\dots$.

In order to find another representation for $A(z)$, we define $p(z)$ by the relation

$$A(z)=e^{-\pi iz}2^{z^2/2}2^{-z/2}e^{p(z)}\prod_{m=1}^{\infty}\exp\left(\frac{2^{-mz}}{m(1-2^{-m})}\right) \tag{4.23}$$

and we also note that for $\Re(z)>0$

$$\prod_{m=1}^{\infty}\exp\left(\frac{2^{-mz}}{m(1-2^{-m})}\right)=\prod_{m=0}^{\infty}\frac{1}{1-2^{-z-m}}. \tag{4.24}$$

Denoting the above product by $g(z)$, it satisfies $g(z+1) = g(z)(1-2^{-z})$ and $g(z) \sim 1$ as $z \to \infty$ with $\Re(z) > 0$. Using (4.23) in (4.20) then shows that $p(z+1) = p(z)$.

We next identify explicitly the periodic function $p(\cdot)$ in terms of the Jacobi elliptic theta function (cf. [3]):

$$\vartheta_1(u) = \vartheta_1(u|\tau) = 2q^{1/4}\sin(u)\prod_{m=1}^{\infty}(1-2\cos(2u)q^{2m}+q^{4m})(1-q^{2m}),$$

where $e^{\pi i\tau} = q$. Setting $q = 1/\sqrt{2}$ and $u = (i\log 2)z/2$ the above becomes

$$\vartheta\left(\frac{i}{2}(\log 2)z\right) = i2^{7/8}\left[\prod_{m=1}^{\infty}(1-2^{-m})\right]\sinh\left(\frac{\log 2}{2}z\right)\prod_{m=1}^{\infty}[1-2\cosh(z\log 2)2^{-m}+4^{-m}]$$

$$= i2^{-1/8}2^{z/2}(1-2^{-z})\prod_{m=1}^{\infty}(1-2^{-m})\prod_{m=1}^{\infty}(1-2^{z-m})\prod_{m=1}^{\infty}(1-2^{-z-m}). \tag{4.25}$$

Using (4.22), (4.24), and (4.25) in (4.23) we find that

$$e^{p(z)} = 2^{-z^2/2}e^{\pi iz}(-i)2^{1/8}\left[\prod_{m=1}^{\infty}(1-2^{-m})^{-2}\right]\vartheta_1\left(\frac{i}{2}(\log 2)z\right). \tag{4.26}$$

The periodicity of $p(z)$ in the real direction follows from the "quasi periodicity" of the theta function in the imaginary direction.

We return to the discussion of the asymptotics of $\tilde{F}_j(n)$ (cf. 4.21) for $n, j \to \infty$ with $\alpha > 1$. For z fixed and n, j large we have

$$\begin{aligned}\frac{\Gamma(z+j-n-2)}{\Gamma(z+j-1)} &= \frac{\pi}{\sin(\pi z)}\frac{(-1)^{n+j}}{\Gamma(n-j+3-z)}\frac{1}{\Gamma(z+j-1)}\\ &\sim \frac{\pi}{\sin(\pi z)}\frac{(-1)^{n+j}}{\Gamma(n-j+3)}\frac{1}{\Gamma(j-1)}\left(\frac{n}{j}-1\right)^z\\ &\sim \frac{(-1)^{n+j}}{\sin(\pi z)}\frac{1}{2}\frac{e^n}{(n-j)^{n-j}j^j}\frac{j^{3/2}}{(n-j)^{5/2}}(\alpha-1)^z.\end{aligned}$$

Here we used $\Gamma(N+z) \sim \Gamma(N)N^z$ for $N \to \infty$ and z fixed. Using the above in (4.21) and approximating $n!$ by Stirling's formula yields

$$\tilde{F}_j(n) \sim \frac{n^n}{(n-j)^{n-j}j^j}\frac{\sqrt{n}j^{3/2}}{(n-j)^{5/2}}\frac{\sqrt{\pi}}{\sqrt{2}}\frac{1}{2\pi i}\int_{\frac{1}{2}-i\infty}^{\frac{1}{2}+i\infty}\frac{(\alpha-1)^z}{\sin(\pi z)}A(z)dz, \tag{4.27}$$

which yields the second representation (2.14) for $I(\alpha)$ that appears in Theorem 2.1(ii). For $\alpha > 2$ we can close the integration contour in the left half-plane and recover (4.17). However, (4.27) applies for all $\alpha > 1$. To obtain (4.27) we took the large n limit of the integrand in (4.21). This can be justified using a dominated convergence argument.

4.3. Asymptotic matching for $\alpha \to 1^+$. For purposes of asymptotic matching, we will need to know the behavior of the approximation in (4.27) as $\alpha \to 1^+$. We first observe that since $A(z)$ is an entire function with zeros of $z = 1, 2, 3, \ldots$, the integrand in (4.27) is analytic for $\Re(z) > 0$. The asymptotics of the integral as $\alpha \to 1$ will be obtained by shifting the contour to the right. Since there are no singularities in the right half-plane, the asymptotics must be governed by a saddle point (cf. [4, 25]).

We use the representation (4.23) for $A(z)$ and note that the infinite product in (4.23) is $1+O(2^{-z})$ as $z\to\infty$ in the right half-plane. We thus write the integral in (4.27) as

$$J\equiv\frac{1}{2\pi i}\int_{\mathcal{B}}e^{-\lambda z}2^{z^2/2}2^{-z/2}Q(z)[1+O(2^{-z})]dz, \tag{4.28}$$

where $\mathcal{B}$ is any vertical contour with $\Re(z)>0$ and

$$Q(z)=\frac{e^{-\pi iz}}{\sin(\pi z)}e^{p(z)},\qquad \alpha-1=e^{-\lambda}. \tag{4.29}$$

The periodicity of $p(\cdot)$ implies that $Q(z+1)=Q(z)$ and $\alpha\to 1^+$ corresponds to $\lambda\to+\infty$.

For $\lambda\to\infty$ the integrand in (4.28) has a saddle point where

$$\frac{d}{dz}\left[-\lambda z+\frac{z^2}{2}\log 2-\frac{z}{2}\log 2\right]=z\log 2-\lambda-\frac{1}{2}\log 2=0.$$

Thus the saddle is at $z\approx\lambda/\log 2$ and we set

$$z=\frac{\lambda}{\log 2}+\frac{1}{2}+\zeta. \tag{4.30}$$

We now use (4.28)–(4.30) and obtain

$$J\sim e^{-\lambda/2}e^{-\lambda^2/(2\log 2)}2^{-1/8}\frac{1}{2\pi}\int_{-\infty}^{\infty}2^{-t^2/2}Q\left(\frac{\lambda}{\log 2}+\frac{1}{2}+it\right)dt \tag{4.31}$$

for $\lambda\to\infty$. By periodicity, the above integral is $O(1)$ in this limit. We use (4.31) in (4.27), set $j=n-k$, and expand the result for $k/n\to 0$. Noting that $e^{-\lambda/2}=\sqrt{k/(n-k)}\sim\sqrt{k/n}$ and $\lambda=\log(n/k)+O(k/n)$, we find that

$$\begin{aligned}\tilde{F}_j(n)\sim{}& e^{\,k\log n+k-k\log k}\frac{n^{3/2}}{k^2}\frac{2^{-9/8}}{\sqrt{\log 2}}Q_*\left(\log_2 n-\log_2 k+\frac{1}{2}\right)\\ &\times\exp\left[-\frac{1}{2\log 2}(\log n-\log k)^2\right],\end{aligned} \tag{4.32}$$

where

$$Q_*(z)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty}e^{-s^2/2}Q\left(z+\frac{is}{\sqrt{\log 2}}\right)ds. \tag{4.33}$$

It is important to note that (4.32) was obtained by first taking $n,j\to\infty$ with n/j fixed and then expanding the result for $n/j=\alpha\to 1$. Thus (4.32) applies for $k/n\to 0$, but we will shortly show that we also need $k/\log_2 n\to\infty$.

We next consider the limits $n\to\infty$ with $k-\log_2 n=O(1)$ and then $n\to\infty$ with $k/\log_2 n=\nu>1$. The first limit has no significance to the distribution h_n^k, since if $k=\log_2 n+O(1)$ we are no longer in the right tail and thus we cannot linearize (4.2). However, the function (4.15) is defined for *all* k. In section 5 we will show that the limit with $k/\log_2 n$ fixed >1 is important for the asymptotic matching between the left and right tails.

We return to the representation of $\tilde{F}_j(n)$ in (4.21) and set $z = n-j+s = k+s$. For k, n large and $s = O(1)$ we use $\Gamma(z+j-1) = \Gamma(n-1+s) \sim \Gamma(n+1)n^{-2}n^s = n!n^{s-2}$ and the approximation for $A(z)$ as $z \to \infty$. Then the asymptotic form of (4.21) is

$$\begin{aligned}(4.34)\quad \tilde{F}_j(n) &\sim \frac{1}{2\pi i}\int_{-i\infty}^{i\infty} \frac{\pi n^{2-s}}{\Gamma(3-s)} Q(s) \exp\left(\frac{\log 2}{2}(k+s)(k+s-1)\right) ds \\ &= \frac{1}{2\pi i}\pi 2^{k^2/2}2^{-k/2}n^2 \int_{-i\infty}^{i\infty} \frac{Q(s)}{\Gamma(3-s)} 2^{s^2/2}2^{-s/2}e^{(k\log 2 - \log n)s} ds,\end{aligned}$$

where the integral is $O(1)$ for $k = \log_2 n + O(1)$. If we use (4.34), we would find that $\mathcal{L}_j(n) = O(n)$, which corresponds to h_n^k large and negative!

Next we show that (4.34) asymptotically matches to (4.32) in an intermediate limit where $n, k \to \infty$ with $k/n \to 0$ but $k - \log_2 n \to \infty$. We also obtain an approximation to $\tilde{F}_j(n)$ that applies for $k/\log_2 n \in (1, \infty)$. We set $k = \nu \log_2 n$ with $\nu > 1$ and expand (4.34) for $n \to \infty$. To leading order, the saddle point equation for (4.34) is

$$\frac{d}{ds}\left[\frac{s^2}{2}\log 2 + (\nu-1)s\log n\right] = s\log 2 + (\nu-1)\log n = 0,$$

so the saddle is at $s \approx -(\nu-1)\log_2 n$.

Since this is asymptotically large, the factor $1/\Gamma(3-s)$ also affects the location of the saddle. For $s \to -\infty$ we thus approximate (cf. [1])

$$(4.35)\qquad \frac{1}{\Gamma(3-s)} = (-s)^{-5/2}\frac{1}{\sqrt{2\pi}}e^{s\log(-s)-s}\left(1 + O\left(\frac{1}{s}\right)\right).$$

Using (4.35) in (4.34) and expanding the integrand near the saddle $s = s_0 \equiv -(\nu-1)\log_2 n$ yields

$$\begin{aligned}(4.36)\quad \tilde{F}_j(n) \sim &\sqrt{\frac{\pi}{2}}2^{k^2/2}2^{-k/2}n^2\frac{1}{2\pi i}\int_{-i\infty}^{i\infty} Q(s)e^{F(s_0)}e^{F'(s_0)(s-s_0)}e^{F''(s_0)(s-s_0)^2/2} \\ &\times\ (-s)^{-5/2}\left[1 + O\left(\frac{1}{s}\right) + O\left(F'''(s_0)(s-s_0)^3\right)\right] ds,\end{aligned}$$

where

$$F(s) = -s + s\log(-s) + \frac{\log 2}{2}s^2 - \frac{\log 2}{2}s + (\nu-1)s\log n$$

so that

$$F'(s) = \log(-s) + (\log 2)s - \frac{1}{2}\log 2 + (\nu-1)\log n,$$

and hence $F'(s_0) = \log(-s_0) - (\log 2)/2$, $F''(s_0) = 1/s_0 + \log 2 \sim \log 2$. Setting $s = s_0 - \log_2(-s_0) + \frac{1}{2} + it$, the leading term for (4.36) becomes

$$\begin{aligned}\tilde{F}_j(n) \sim & \sqrt{\frac{\pi}{2}} 2^{k^2/2} 2^{-k/2} n^2 (-s_0)^{-2} e^{F(s_0)} \exp\left(-\frac{1}{2}(\log 2)[\log_2(-s_0)]^2\right) 2^{-1/8} \\ & \times \frac{1}{2\pi}\int_{-\infty}^{\infty} 2^{-t^2/2} Q\left(s_0 - \log_2(-s_0) + \frac{1}{2} + it\right) dt \\ & = \frac{2^{-9/8}}{\sqrt{\log 2}} (k - \log_2 n)^{-2} \exp\left(k\log n - \frac{1}{2}\frac{\log^2 n}{\log 2} + \frac{3}{2}\log n\right) \\ & \times \exp\left[-(k-\log_2 n)\log(k-\log_2 n) + k - \log_2 n - \frac{\log^2(k-\log_2 n)}{2\log 2}\right] \\ & \times Q_*\left(\log_2 n - \log_2(k-\log_2 n) + \frac{1}{2}\right), \end{aligned} \tag{4.37}$$

where $Q_*(z)$ is given by (4.33). Expression (4.37) holds in the limit $k, n \to \infty$ with $k/\log_2 n$ fixed and > 1. We can easily show that it matches to (4.32) in the intermediate limit where $k/n \to 0$, $k/\log_2 n \to \infty$. In this limit

$$\begin{aligned} -(k-\log_2 n)\log(k-\log_2 n) + k - \log_2 n - \frac{\log^2(k-\log_2 n)}{2\log 2} \\ = -k\log k + k + (\log_2 n)(\log k) - \frac{\log^2 k}{2\log 2} + O(1), \end{aligned}$$

so that (4.37) becomes the same as (4.32). This also shows that (4.32) applies only to the range where $k/\log_2 n \to \infty$. In order to asymptotically match the right and left tails, we shall need to use (4.37). We also note that if we multiply (4.37) by $2^{-k^2/2}2^{-k/2}$ (thus obtaining the corresponding approximation to $1 - h_n^k = H_n^k$), the result has the form $n \times$ [function of $(k - \log_2 n)$]. This is precisely what is needed in order to match to the WKB expansion that we will derive in section 5.

For $k, n \to \infty$ with $k/\log_2 n = \nu > 1$, we are still in the right tail. We can estimate the value of k where there is appreciable mass by using (4.37). We argue that when $H_n^k = 2^{-k^2/2}2^{-k/2}F_j(n)$ becomes $O(1)$ in n, then we are no longer in the tail. Since Q_* is clearly $O(1)$, this condition is equivalent to

$$\begin{aligned} & -\frac{k^2}{2}\log 2 - \frac{k}{2}\log 2 + k\log n + \frac{3}{2}\log n - \frac{1}{2}\frac{\log^2 n}{\log 2} - (k-\log_2 n)\log(k-\log_2 n) \\ & + k - \log_2 n - \frac{\log^2(k-\log_2 n)}{2\log 2} - 2\log(k-\log_2 n) = O(1). \end{aligned} \tag{4.38}$$

The largest terms in (4.38) are $-\frac{\log 2}{2}(k-\log_2 n)^2 + \log n$, which balance when $k - \log_2 n \sim \sqrt{2\log_2 n}$. This gives a rough argument that the mean $\mathbf{E}[\mathcal{H}_n]$ behaves as $\mathbf{E}[\mathcal{H}_n] - \log_2 n \sim \sqrt{2\log_2 n}$. A more careful analysis of (4.38) shows that the left side is $O(1)$ for

$$\begin{aligned} k = & \log_2 n + \sqrt{2\log_2 n} - \log_2(\sqrt{2\log_2 n}) + \frac{1}{\log 2} - \frac{1}{2} \\ & - \frac{3}{2}\frac{\log_2(\sqrt{2\log_2 n})}{\sqrt{2\log_2 n}} + O((\log n)^{-1/2}). \end{aligned} \tag{4.39}$$

We discussed the condition in (4.39) in more detail in section 3.

4.4. Another representation for $Q(z)$. We next obtain another representation for the periodic function $p(z)$ (and hence $Q(z)$), which involves a Fourier series that we need in some computations. We let $\tilde{A}(z)$ be a solution of (4.20) and set $\tilde{A}(z) = \exp[B(z)]$. Then $B(z)$ satisfies

$$B(z+1) - B(z) = \log(1 - 2^z). \tag{4.40}$$

Introducing the two-sided Laplace transform

$$B^*(s) = \int_{-\infty}^{\infty} B(z) e^{-sz} dz,$$

we find that

(4.41)

$$\begin{aligned}(e^s - 1)B^*(s) &= \int_{-\infty}^{\infty} \log(1 - 2^z) e^{-sz} dz \\ &= \int_{-\infty}^{0} \log(1 - 2^z) e^{-sz} dz + \int_0^{\infty} (\pm \pi i + \log(2^z - 1)) e^{-sz} dz \\ &= \int_0^{\infty} e^{sz} \log(1 - 2^{-z}) dz \pm \frac{\pi i}{s} + \frac{\log 2}{s^2} + \int_0^{\infty} e^{-sz} \log(1 - 2^{-z}) dz \\ &= \pm \frac{\pi i}{s} + \frac{\log 2}{s^2} + \sum_{\substack{m=-\infty \\ m \neq 0}}^{\infty} \frac{1}{m} \frac{1}{s - m \log 2},\end{aligned}$$

where both integrals converge in the strip $-\log 2 < \Re(s) < \log 2$, and the left side is analytic for $0 < \Re(s) < \log 2$. The standard inversion formula then yields

$$B(z) = \frac{1}{2\pi i} \int_{\frac{1}{2} - i\infty}^{\frac{1}{2} + i\infty} \frac{e^{sz}}{e^s - 1} \left[\frac{\log 2}{s^2} \pm \frac{\pi i}{s} + \frac{\pi}{s} \left(\cot\left(\frac{\pi s}{\log 2} \right) - \frac{\log 2}{\pi s} \right) \right] ds, \tag{4.42}$$

where $0 < \Re(s) < \log 2$ on the contour of integration. Here we have used the partial fractions expansion of $\cot(z)$ to evaluate the sum in (4.41).

The integrand $h(s)$ in (4.42) has a triple pole at $s = 0$, simple poles at $s = 2k\pi i$, $k = \pm 1, \pm 2, \ldots$, and simple poles where $s = \ell \log 2$, $\ell = \pm 1, \pm 2, \ldots$. To evaluate $B(z)$ as $z \to \infty$ we shift the contour to the left. A lengthy calculation shows that the residue at $z = 0$ is

$$\operatorname{Res}[h(s); s = 0] = \log 2 \left(\frac{z^2}{2} - \frac{z}{2} + \frac{1}{12} \right) \pm \pi i \left(z - \frac{1}{2} \right) - \frac{\pi^2}{3 \log 2}. \tag{4.43}$$

The residues along the imaginary axis combine with (4.43) to yield

$$\begin{aligned}B(z) = {} & \frac{\log 2}{2}(z^2 - z) \pm i\pi \left(z - \frac{1}{2} \right) + \frac{\log 2}{12} - \frac{\pi^2}{3 \log 2} \\ & + \sum_{\substack{m=-\infty \\ m \neq 0}}^{\infty} e^{2m\pi i z} \left[\pm \frac{1}{2m} - \frac{1}{2m} \coth\left(\frac{2\pi^2 m}{\log 2} \right) \right] + O(2^{-z})\end{aligned} \tag{4.44}$$

as $z \to \infty$. By shifting the contour to the right we find that $B(z) \sim -2^z$ as $z \to -\infty$.

We observe that $\tilde{A} = e^B$ and A in (4.19) (or (4.22)) are related by $\tilde{A}(z)/\tilde{A}(0) = A(z)$. For z real we have

$$\sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} \frac{1}{m} e^{2m\pi iz} = -2\pi i \left(\langle z \rangle - \frac{1}{2} \right),$$

where, to recall, $\langle z \rangle$ is the fractional part of z. By comparing (4.44) with (4.23) we find that

$$Q(z) = \frac{e^{-\pi iz}}{\sin(\pi z)} e^{p(z)} = \frac{1}{\sin(\pi z)} 2^{1/12} \exp\left(-\frac{\pi^2}{3\log 2} \pm i\pi \lfloor z \rfloor \right) \prod_{\ell=1}^{\infty} (1-2^{-\ell})^{-1} \times \exp\left[-\sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} \frac{e^{2m\pi iz}}{2m} \coth\left(\frac{2\pi^2 m}{\log 2} \right) \right]. \tag{4.45}$$

By periodicity it suffices to consider $0 < z < 1$. Writing

$$[\sin(\pi z)]^{-1} = e^{-\log(\sin \pi z)} = \exp\left[\log 2 + \frac{1}{2} \sum_{\substack{m=-\infty \\ m\neq 0}}^{\infty} \operatorname{sgn}(m) \frac{e^{2m\pi iz}}{m} \right]$$

($\operatorname{sgn}(x)$ is 1 for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$) and noting that

$$\coth(am) - \operatorname{sgn}(m) = \frac{e^{-a|m|}}{\sinh(am)}, \qquad a > 0,$$

we find from (4.44), (4.25), and (4.26) that

$$Q(z) = 2^{13/12} \left[\prod_{\ell=1}^{\infty} (1-2^{-\ell})^{-1} \right] \exp\left[-\frac{\pi^2}{3\log 2} - \sum_{m=1}^{\infty} \frac{e^{-am}}{m \sinh(am)} \cos(2m\pi z) \right] = \frac{2^{-z^2/2 + z/2}(1-2^{-z})}{\sin(\pi z)} \prod_{m=1}^{\infty} (1-2^{-m})(1-2^{z-m})(1-2^{-z-m}), \tag{4.46}$$

where

$$a = \frac{2\pi^2}{\log 2}.$$

This coincides with the representation of $Q(z)$ given in Theorem 2.1(iii). From the Fourier series we see that $Q(z)$ is nearly constant since

$$\left| \sum_{m=1}^{\infty} \frac{e^{-am}}{m \sinh(am)} \cos(2m\pi z) \right| \leq \sum_{m=1}^{\infty} \frac{e^{-am}}{m \sinh(am)} = 3.678\ldots(10^{-25}).$$

To summarize the calculations, we have obtained the leading term for $1 - h_n^k$ for the limits (i) $n \to \infty$, $n-k = j = O(1)$, (ii) $n, k \to \infty$ with $0 < k/n = 1 - 1/\alpha < 1$, and (iii) $k, n \to \infty$ with $k/\log_2 n = \nu > 1$. The last corresponds to the "left-most" right-tail region. The third region involves correction terms that are $O(1/\log n)$, while the first two had correction terms $O(1/n)$. Thus we expect that the third case will result in the worst numerical agreement (this was discussed in more detail in section 3).

5. The left-tail and central-regime analyses. We analyze here the distribution when h_n^k is asymptotically small. Then we can no longer linearize (2.2). Whereas the right tail was treated using purely linear analysis, the left tail requires different techniques such as the WKB method and matched asymptotics.

5.1. The left-tail analysis. We first prove part (iv) of Theorem 2.1. We set $2^{k+1} - n = M$, $M \geq 1$, and first show how to compute h_n^k explicitly for small values of M. Since the generating function in (2.4) is a polynomial of degree $2^{k+1} - 1$ we isolate the two leading coefficients by writing

$$H_k(z) = a(k)z^{2^{k+1}-1} + b(k)z^{2^{k+1}-2} + \cdots + z + 1. \tag{5.1}$$

It follows that

$$H_k^2(z) = a^2(k)z^{2^{k+2}-2} + 2a(k)b(k)z^{2^{k+2}-3} + O(z^{2^{k+2}-4}) \tag{5.2}$$

and

$$\begin{aligned} H'_{k+1}(2z) = \frac{1}{2}\Big[\, & a\ (k+1)2^{2^{k+2}-1}(2^{k+2}-1)z^{2^{k+2}-2} \\ & + b(k+1)2^{2^{k+2}-2}(2^{k+2}-2)z^{2^{k+2}-3} + O(z^{2^{k+2}-4})\Big]. \end{aligned} \tag{5.3}$$

By using (2.5), (5.2), and (5.3) we obtain the recurrences

$$2^{2^{k+2}}(2^{k+2}-1)a(k+1) = 4a^2(k), \qquad a(0) = 1, \tag{5.4}$$

$$2^{2^{k+2}}(2^{k+2}-2)b(k+1) = 16a(k)b(k), \qquad b(0) = 1. \tag{5.5}$$

Equation (5.4) is nonlinear but (5.5) is linear, once we compute $a(k)$.

To solve (5.4) we set $a(k) = e^{\alpha(k)}$ to get

$$\alpha(k+1) - 2\alpha(k) = -\log\left[\frac{1}{4}2^{2^{k+2}}(2^{k+2}-1)\right]. \tag{5.6}$$

If $\alpha(k) = 2^k\tilde{\alpha}(k)$, then (5.6) becomes

$$\tilde{\alpha}(k+1) - \tilde{\alpha}(k) = -2^{-k-1}\log\left[\frac{1}{4}2^{2^{k+2}}(2^{k+2}-1)\right]. \tag{5.7}$$

We solve (5.7) subject to $\tilde{\alpha}(0) = 0$ to obtain

$$\alpha(k) = -2^k\sum_{\ell=0}^{k-1} 2^{-\ell-1}\log\left[\frac{1}{4}2^{2^{\ell+2}}(2^{\ell+2}-1)\right]. \tag{5.8}$$

Exponentiating (5.8) and some rearrangement yields

$$\begin{aligned} a(k) &= \prod_{\ell=1}^{k}\left(\frac{4}{2^{2^{\ell+1}}(2^{\ell+1}-1)}\right)^{2^{k-\ell}} \\ &= 2^{-k2^{k+1}}4^{2^k-1}\prod_{\ell=1}^{k}\left(\frac{1}{2^{\ell+1}-1}\right)^{2^{k-\ell}}. \end{aligned} \tag{5.9}$$

We next obtain the asymptotics of $a(k)$ as $k \to \infty$. We write

$$2^k \sum_{\ell=1}^{k} 2^{-\ell} \log(2^{\ell+1}-1) = (3\cdot 2^k - k + 3)\log 2 + 2^k \sum_{\ell=1}^{k} 2^{-\ell}\log(1-2^{-\ell-1})$$

so that

$$a(k) = 2^{-(2k+3)2^k} 2^{2^{k+1}+k+1} \left\{ \prod_{\ell=1}^{k} (1-2^{-\ell-1})^{-2^{-\ell}} \right\}^{2^k} \tag{5.10}$$
$$= 2^{-(2k+1)2^k} 2^{k+1} C_0^{2^k} [1+O(2^{-k})], \qquad k \to \infty,$$

where

$$C_0 = \prod_{\ell=1}^{\infty} (1-2^{-\ell-1})^{-2^{-\ell}}, \tag{5.11}$$

whose numerical value was given in (2.18) of section 2.

Since $h^k_{2^{k+1}-1} = (2^{k+1}-1)!a(k) = \Gamma(2^{k+1})a(k)$, we use Stirling's formula to get

$$h^k_{2^{k+1}-1} = 2\sqrt{\pi} 2^{k/2} \left(\frac{2C_0}{e^2}\right)^{2^k} [1+O(2^{-k})], \quad k \to \infty. \tag{5.12}$$

Using our result for $a(k)$ in (5.5) and solving the linear recurrence yields

$$b(k) = \prod_{\ell=1}^{k} \frac{8a(\ell-1)}{2^{2^{\ell+1}}(2^\ell-1)}. \tag{5.13}$$

Using the (exact) expression for $a(k)$ in (5.10) in (5.13), and noting that

$$\prod_{\ell=1}^{k} \frac{8\cdot 2^\ell}{2^{2^{\ell+1}}(2^\ell-1)} 2^{-(\ell+1/2)2^\ell} 2^{2^\ell} = 8^k 2^{1-(2k+1)2^k} \prod_{\ell=1}^{k} \left(\frac{1}{1-2^{-\ell}}\right),$$

we obtain

$$b(k) = 2^{-(2k+1)2^k} 2^{3k+1} \prod_{\ell=1}^{k} \left(\frac{1}{1-2^{-\ell}} \left(\prod_{j=1}^{\ell-1} (1-2^{-j-1})^{-2^{-j}} \right)^{2^{\ell-1}} \right). \tag{5.14}$$

We write the double product in (5.14) as

$$\prod_{\ell=1}^{k} C_0^{2^{\ell-1}} \frac{1}{1-2^{-\ell}} \left(\prod_{j=\ell}^{\infty} (1-2^{-j-1})^{2^{-j}} \right)^{2^{\ell-1}} \sim C_0^{2^k-1} C_1, \qquad k \to \infty, \tag{5.15}$$

where C_0 is given in (5.11) and

$$C_1 = \prod_{\ell=1}^{\infty} \left(\frac{1}{1-2^{-\ell}} \prod_{j=\ell}^{\infty} (1-2^{-j-1})^{2^{\ell-j-1}} \right).$$

In view of (5.15) we have

$$b(k) \sim 2^{-(2k+1)2^k} 2^{3k+1} C_0^{2^k-1} C_1, \quad k \to \infty, \tag{5.16}$$

and then $h^k_{2^{k+1}-2} = (2^{k+1}-2)!b(k)$ has the expansion

$$h^k_{2^{k+1}-2} \sim \frac{C_1}{C_0}\sqrt{\pi} 2^{3k/2} \left(\frac{2C_0}{e^2}\right)^{2^k}, \quad k \to \infty. \tag{5.17}$$

The constant C_1 may be simplified by noting that

$$\begin{aligned} C_1 &= \prod_{\ell=1}^{\infty} \frac{1}{1-2^{-\ell}} \left[\prod_{m=1}^{\infty} (1-2^{-m-\ell})^{2^{-m}}\right] \\ &= \left[\prod_{\ell=1}^{\infty} \frac{1}{1-2^{-\ell}}\right] \left(\prod_{N=2}^{\infty} \left[\prod_{m=1}^{N-1} (1-2^{-N})^{2^{-m}}\right]\right) \\ &= \left[\prod_{N=2}^{\infty} (1-2^{-N})^{1-2^{1-N}}\right] \left[\prod_{\ell=1}^{\infty} \frac{1}{1-2^{-\ell}}\right] \\ &= 2\prod_{N=2}^{\infty} (1-2^{-N})^{-2^{1-N}} \\ &= 2C_0. \end{aligned}$$

Hence, $h^k_n \sim 2\sqrt{\pi} 2^{3k/2} (2C_0 e^{-2})^{2^k}$ for $k \to \infty$ if $n = 2^{k+1}-2$.

Next we solve the recurrence (2.2) for $M = O(1)$ and $n \to \infty$, thus proving Theorem 2.1(iv). We change variables from (n,k) to (M,k) with $n = 2^{k+1} - M$ and

$$h^k_n = G(M,k) = G(2^{k+1}-n,k). \tag{5.18}$$

We replace k by $k-1$ in (2.2) and note that

$$\begin{aligned} h^k_{n+1} &= G(2^{k+1}-n-1,k) = G(M-1,k), \\ h^{k-1}_i &= G(2^k - i, k-1) = G\left(\frac{M}{2} + \frac{n}{2} - i, k-1\right). \end{aligned}$$

Thus (2.2) becomes, in terms of G and M,

$$G(M-1,k) = \sum_{i=0}^{n} \binom{n}{i} 2^{-n} G\left(\frac{M}{2} + \frac{n}{2} - i, k-1\right) G\left(\frac{M}{2} - \frac{n}{2} + i, k-1\right). \tag{5.19}$$

But $G = 0$ for $M \le 0$ (i.e., $n \ge 2^{k+1}$) so that the sum in (5.19) may be truncated over the range $(n-M)/2 < i < (n+M)/2$. Then setting $i = \ell + (n-M)/2$ we have

$$G(M-1,k) = \sum_{\ell=1}^{M-1} \binom{n}{\ell + (n-M)/2} 2^{-n} G(\ell, k-1) G(M-\ell, k-1). \tag{5.20}$$

The expression in (5.20) is still exact. For $n, k \to \infty$ with $M = O(1)$, we have

$$\binom{n}{i} 2^{-n} = \sqrt{\frac{2}{\pi n}} [1 + O(n^{-1})] = \frac{2^{-k/2}}{\sqrt{\pi}} [1 + O(2^{-k})],$$

so in this range we replace (5.20) by the asymptotic relation

$$G(M-1,k) \sim \frac{2^{-k/2}}{\sqrt{\pi}} \sum_{\ell=1}^{M-1} G(\ell,k-1)G(M-\ell,k-1). \tag{5.21}$$

Setting $M=1$, replacing $\sim$ by $=$, and solving (5.21) for $G(1,k)$ yields

$$G(1,k) \sim 2\sqrt{\pi}2^{k/2}A_0^{2^k}, \tag{5.22}$$

where A_0 is an undetermined constant. Setting $M=2$ in (5.21) and using (5.22) leads to

$$\frac{G(2,k)}{G(2,k-1)} \sim \frac{2}{\sqrt{\pi}}2^{-k/2}G(1,k-1) \sim 2\sqrt{2}A_0^{2^{k-1}}, \tag{5.23}$$

so that

$$G(2,k) \sim 2\sqrt{\pi}B_0 2^{3k/2}A_0^{2^k}, \tag{5.24}$$

where B_0 is another constant. We have scaled the factor $2\sqrt{\pi}$ out of the constant for convenience. Proceeding inductively we find that for general M

$$G(M,k) \sim 2\sqrt{\pi}B_0^{M-1}2^{(M-1/2)k}A_0^{2^k}\mathcal{G}(M), \quad k\to\infty. \tag{5.25}$$

Using (5.25) in (5.21) we find that

$$\mathcal{G}(M-1) = 4\cdot 2^{-M}\sum_{\ell=1}^{M-1}\mathcal{G}(\ell)\mathcal{G}(M-\ell) \tag{5.26}$$

with $\mathcal{G}(2)=1$. Thus, $\mathcal{G}(M)=1/(M-1)!$, which yields

$$G(M,k) \sim \frac{2\sqrt{\pi}}{(M-1)!}B_0^{M-1}2^{(M-1/2)k}A_0^{2^k}. \tag{5.27}$$

It remains only to determine A_0 and B_0. But our exact analysis for $M=1$ and $M=2$ shows, in view of (5.10) and (5.17) (with $C_1/C_0=2$), that

$$A_0 = \frac{2C_0}{e^2}, \quad B_0 = 1. \tag{5.28}$$

We have thus derived the result in (2.22) of Theorem 2.1(iv). We conclude by noting that the range $M=O(1)$ corresponds to the "left-most" tail of the distribution, and that for $n\to\infty$ the condition $M=O(1)$ can be satisfied only when n is close to a power of 2.

5.2. The central-regime analysis. We next analyze the scale $k,n\to\infty$ with $k-\log_2 n = O(1)$. We thus set $\xi = n2^{-k}$ and

$$h_n^k = F(\xi;n) = F(n2^{-k};n). \tag{5.29}$$

We consider the range $0<\xi<2$ and note that as $\xi\to 2^-$ we are approaching the scale $M=O(1)$, which we just analyzed. We have $k-\log_2 n = -\log_2\xi$ so that $k-\log_2 n\to\infty$ corresponds to $\xi\to 0^+$. For any fixed $\xi>0$ we are still in the left

tail as the mass is concentrated where $k - \log_2 n \sim \sqrt{2\log_2 n}$, which corresponds to $\xi \approx 2^{-\sqrt{2\log_2 n}}$, which is small.

We comment that the ξ-scale also arises in related models. We have previously shown in [19] that for tries, b-tries, and PATRICIA trees, the limit $n, k \to \infty$ with ξ fixed is important to the asymptotic analysis. We note that

$$h_{n+1}^k = F((n+1)2^{-k}; n+1) = F\left(\xi + \frac{\xi}{n}; n+1\right)$$

and

$$h_i^{k-1} = F(i2^{-k+1}; i) = F\left(\frac{2i}{n}\xi; i\right).$$

Using the above in (2.2), after replacing k by $k-1$, we obtain

$$F\left(\xi + \frac{\xi}{n}; n+1\right) = \sum_{i=0}^{n} \binom{n}{i} 2^{-n} F\left(\frac{2i}{n}\xi; i\right) F\left(2\left(1 - \frac{i}{n}\right)\xi; n-i\right). \tag{5.30}$$

We analyze (5.30) by a WKB-type expansion (cf. [10]). That is, we seek an asymptotic solution of (5.30) in the form

$$F(\xi; n) = e^{-n\Phi(\xi)}\left[A(\xi) + \frac{1}{n}A^{(1)}(\xi) + O(n^{-2})\right]. \tag{5.31}$$

The above may be viewed as a generalized saddle-point expansion. For simpler models, such as tries and b-tries, we can obtain exact expressions for the corresponding distributions. These usually involve Cauchy integrals that can be asymptotically evaluated by the saddle-point method. This then leads to an expansion of the form (5.31) for fixed ξ. Note that for b-tries ξ takes on values in the range $[0, b]$ and for PATRICIA trees the range is $[0, 1]$. For more difficult models, which cannot be or have not been solved exactly, we must try to obtain the asymptotic expansion directly from the equations, such as (2.2). This generally requires making an *ansatz*, such as (5.31).

Setting $x = i/n$ and $y = \sqrt{n}(x - 1/2) = \sqrt{n}(i/n - 1/2)$ and using Stirling's formula, we have

$$\binom{n}{i} 2^{-n} \sim \frac{e^{nf_0(x)}}{\sqrt{2\pi n x(1-x)}}, \qquad 0 < x < 1, \tag{5.32}$$

where $f_0(x) = -\log 2 - x\log x - (1-x)\log(1-x)$. For y fixed (5.32) simplifies to the Gaussian

$$\binom{n}{i} 2^{-n} = \sqrt{\frac{2}{\pi n}} e^{-2y^2}[1 + O(n^{-1})]. \tag{5.33}$$

We use (5.31) in (5.30) and expand for large n, which yields

$$e^{-n\Phi(\xi)} \cdot e^{-(\xi\Phi)'(\xi)}[A(\xi) + O(n^{-1})] = \sum_{i=0}^{n} \frac{e^{nf_0(i/n)}}{\sqrt{2\pi n}} \sqrt{\frac{n^2}{i(n-i)}}$$
$$\times \exp\left(-n\left[\frac{i}{n}\Phi\left(\frac{2i}{n}\xi\right) + \left(1 - \frac{i}{n}\right)\Phi\left(2\left(1 - \frac{i}{n}\right)\xi\right)\right]\right) A\left(\frac{2i}{n}\xi\right) A\left(2\left(1 - \frac{i}{n}\right)\xi\right).$$

Note that near the endpoints in the sum in (5.30), i and $n-i$ may not be large so that the expansion (5.31) does not apply. However, the major contribution to the sum comes from the range $i = n/2 + O(\sqrt{n})$. We approximate the sum in (5.34) by an integral via Euler–Maclaurin to get

(5.34)
$$e^{-n\Phi(\xi)} \cdot e^{-(\xi\Phi)'(\xi)}[A(\xi) + O(n^{-1})] \sim \frac{\sqrt{n}}{\sqrt{2\pi}} \int_0^1 \frac{e^{nf_0(x)}}{\sqrt{x(1-x)}} A(2x\xi)A(2(1-x)\xi)e^{-n[x\Phi(2\xi x)+(1-x)\Phi(2\xi(1-x))]}dx.$$

By symmetry, the major contribution must come from the midpoint $x = 1/2$. Setting

$$g(t) = t\Phi(2\xi t) + (1-t)\Phi(2\xi(1-t))$$

we have $g(1/2) = \Phi(\xi)$, $g'(1/2) = 0$, and

$$g''\left(\frac{1}{2}\right) = 8\xi\Phi'(\xi) + 4\xi^2\Phi''(\xi).$$

Setting $x = 1/2 + y/\sqrt{n}$ and expanding the integral by Laplace's method yields, to leading order,

$$e^{-n\Phi(\xi)}e^{-(\xi\Phi)'(\xi)}A(\xi) = \frac{1}{\sqrt{2\pi}} \frac{2\sqrt{2\pi}}{\sqrt{4 + g''(1/2)}} e^{-n\Phi(\xi)}A^2(\xi).$$

Thus the exponential factors cancel and we have

$$A(\xi) = [1 + 2\xi\Phi'(\xi) + \xi^2\Phi''(\xi)]^{1/2}e^{-\xi\Phi'(\xi)-\Phi(\xi)}. \tag{5.35}$$

We have thus expressed $A(\cdot)$ in terms of $\Phi(\cdot)$, though the latter remains undetermined. By refining the approximation (5.33) to the "kernel" in (5.30) and obtaining higher order terms in the Laplace expansion of the integral, we can obtain relations between the terms $A^{(j)}$ in the series in (5.31). Using these we can express $A^{(j+1)}$ in terms of $\Phi, A, A^{(1)}, \ldots, A^{(j)}$, but we can never determine Φ. For tries and b-tries the corresponding Φ can be determined using a standard saddle-point expansion. For PATRICIA trees we could only study Φ numerically and asymptotically, which we proceed to do for the DST model.

Next we use asymptotic matching to determine the behaviors of $\Phi(\xi)$ as $\xi \to 2^-$ and $\xi \to 0^+$. We first require that the expansion on the M-scale matches that on the ξ-scale. This amounts to assuming that both are valid on some intermediate scale where $M = 2^{k+1} - n \to \infty$ but $\xi = n2^{-k} = 2n/(n+M) \to 2$. We write this condition symbolically as

$$G(M,k)|_{M\to\infty} \sim F(\xi;n)|_{\xi\to 2}. \tag{5.36}$$

To compare the two sides of (5.36) we expand G as $M \to \infty$, which simply amounts to approximating $(M-1)!$ by Stirling's formula in (5.27). Using (5.28) then yields

$$G(M,k) \sim \sqrt{2M}\left(\frac{2C_0}{e^2}\right)^{n/\xi} e^M \left(\frac{2^k}{M}\right)^M 2^{-k/2}. \tag{5.37}$$

Noting that $n/\xi = 2^k$ and $M = n(2/\xi - 1)$ we rewrite (5.37) in terms of n and ξ. According to (5.36) this should be the expansion of $Ae^{-n\Phi}$ as $\xi \to 2$. We then find that the matching condition is satisfied provided that

$$\Phi(\xi) = \frac{1}{2}\log\left(\frac{e^2}{2C_0}\right) + \frac{1}{2}(2-\xi)\log(2-\xi) + (\xi-2)\left[\frac{1}{4}\log\left(\frac{2C_0}{e^2}\right) + \frac{1}{2}\right] + o(2-\xi) \tag{5.38}$$

and for $\xi \to 2^-$

$$A(\xi) \sim \sqrt{2(2-\xi)}. \tag{5.39}$$

Thus the matching condition yields the behavior of both Φ and A. We show that (5.38) and (5.39) are consistent with (5.35). From (5.38) we get $\Phi(\xi) \sim \Phi(2) = -1 - \log(\sqrt{2C_0})$, $\Phi'(\xi) \sim -\frac{1}{2}\log(2-\xi)$, and $\Phi''(\xi) \sim 1/[2(2-\xi)]$. Using these results in (5.35) yields $A(\xi) \sim \sqrt{4\Phi''(\xi)}e^{-2\Phi'(\xi)}e^{-\Phi(2)} \sim \sqrt{\frac{2}{2-\xi}}(2-\xi)$, which recovers (5.39). To obtain the above we also used the second term in the expansion of $\Phi'(\xi)$ as $\xi \to 2$.

Next we match the WKB expansion to the "left-most" right tail, which we derived in section 4. We let $\xi \to 0$ with $k/\log_2 n = \nu \to 1^+$. Furthermore we choose an intermediate limit where $n \to \infty$, $\xi \to 0$ so that $n\Phi(\xi) \to 0$ and we can approximate

$$1 - h_n^k \sim 1 - Ae^{-n\Phi} \sim 1 - e^{-n\Phi} \sim n\Phi.$$

The expansion that applies for fixed $k/\log_2 n = \nu > 1$ is given by $1 - h_n^k \sim 2^{-k^2/2}2^{-k/2}\tilde{F}_j(n)$, where $\tilde{F}_j$ is given by (4.37). Expanding (4.37) for $k/\log_2 n = \nu \to 1^+$ amounts to doing nothing, since Q_* is periodic and the rest of (4.37) is in the simplest possible form. We have already observed that $2^{-k^2/2}2^{-k/2}\times$ (4.37) can be written as $n\times$ [function of $(k - \log_2 n)$] = $n\times$ [function of ξ]. Thus the matching condition is satisfied if

$$n\Phi(\xi)|_{\xi\to 0^+} \sim \frac{2^{-9/8}}{\sqrt{\log 2}}\frac{n}{(-\log_2\xi)^2}Q_*\left(\log_2\xi - \log_2(-\log_2\xi) + \frac{1}{2}\right) \times \exp\left(-\frac{\log^2\xi}{2\log 2} + (\log_2\xi)\log(-\log_2\xi) + \left(\frac{1}{2} - \frac{1}{\log 2}\right)\log\xi - \frac{[\log(-\log_2\xi)]^2}{2\log 2}\right). \tag{5.40}$$

The above is equivalent to (2.19) and we have used the periodicity of Q_*.

Expression (5.40) represents the leading term for Φ and we may also view it as representing the first six terms in the expansion of $\log\Phi$. These have respective orders of magnitude $O(\log^2\xi)$, $O(|\log\xi|\log|\log\xi|)$, $O(|\log\xi|)$, $O(\log^2|\log\xi|)$, $O(\log|\log\xi|)$, and $O(1)$. The $O(1)$ term involves the periodic function Q_*. The analysis in section 4 suggests that there is an error term in (5.40) of the form $[1 + O(1/|\log\xi|)]$. It may thus be desirable to compute higher order terms for Φ, which would involve higher order matchings (cf. section 3). Finally we note that Φ and all its derivatives vanish as $\xi \to 0^+$. This completes the analysis of the left tail. To summarize, we have treated the M and ξ scales, determined the leading term completely for the former, and obtained partial results for the latter.

Appendix A. We derive the series representation (2.30) for the periodic function $Q_*(z)$. By using the series representation (2.21) for $\vartheta_1(\cdot)$, comparing it to the infinite product form of $\vartheta_1(\cdot)$, and also using $Q(z) = e^{-\pi iz}\csc(\pi z)e^{p(z)}$ we obtain

$$Q(z) = \frac{2^{-z^2/2}2^{-z/2}}{\sin(\pi z)}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{m/2}2^{mz}. \tag{A.1}$$

This yields a third representation for $Q(z)$, which supplements the two in (4.46). We evaluate

$$\begin{aligned} Q_*(z) &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-s^2/2}Q\left(z+\frac{is}{\sqrt{\log 2}}\right)ds \\ &= \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} e^{-y^2/2}e^{-i\sqrt{\log 2}zy}2^{z^2/2}Q\left(\frac{iy}{\sqrt{\log 2}}\right)dy, \end{aligned} \tag{A.2}$$

where we have shifted the contour by setting $s = i\sqrt{\log 2}z + y$. Using (A.1) in (A.2) and exchanging the orders of integration and summation, we obtain

$$\begin{aligned} Q_*(z) =& \frac{1}{\sqrt{2\pi}}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]2^{z^2/2}\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{m/2} \\ &\times \int_{-\infty}^{\infty}\Re\left[\frac{e^{-iy\sqrt{\log 2}/2}e^{-i\sqrt{\log 2}zy}2^{miy/\sqrt{\log 2}}}{i\sinh(\pi y/\sqrt{\log 2})}\right]dy, \end{aligned} \tag{A.3}$$

where $\Re$ denotes the real part. From tables of integrals we have

$$\int_{-\infty}^{\infty}\frac{\sin(Ax)}{\sinh(Bx)}dx = \frac{\pi}{B}\tanh\left(\frac{\pi A}{2B}\right). \tag{A.4}$$

Applying (A.4) to (A.3) with $B = \pi/\sqrt{\log 2}$ and $A = \left(m - z - \frac{1}{2}\right)\sqrt{\log 2}$, we get

$$\begin{aligned} Q_*(z) =& \frac{\sqrt{\log 2}}{\sqrt{2\pi}}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]2^{z^2/2} \\ &\times \sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{m/2}\tanh\left(\frac{\log 2}{2}\left(m-z-\frac{1}{2}\right)\right). \end{aligned} \tag{A.5}$$

By changing $m \to 1-m$, we see that $Q_*(z) = Q_*(-z)$. We can thus write $Q_*(z) = \frac{1}{2}[Q_*(z) + Q_*(-z)]$. Then we use

$$\begin{aligned} &\tanh\left[\frac{\log 2}{2}\left(m-z-\frac{1}{2}\right)\right] + \tanh\left[\frac{\log 2}{2}\left(m+z-\frac{1}{2}\right)\right] \\ &\quad = \frac{2(2^{m-1/2}-2^{-m+1/2})}{2^{m-1/2}+2^{-m+1/2}+2^z+2^{-z}} \end{aligned}$$

with which (A.5) becomes the same as (2.30).

Appendix B. We obtain a series representation for $\Phi_0(\xi)$, defined in (3.4). We use the series form of $Q(z)$ found in (A.1) and exchange the order of summation and

integration. This yields

$$\Phi_0(\xi) = \pi\,\xi\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}2^{-m^2/2}2^{m/2}(-1)^{m+1} \times \frac{1}{2\pi i}\int_{\frac{5}{2}-i\infty}^{\frac{5}{2}+i\infty}\frac{2^{-s}2^{ms}2^{\Delta s}}{\Gamma(3-s)\sin(\pi s)}ds, \tag{B.1}$$

where $\Delta = -\log_2\xi$. In (3.4) the contour of integration is any vertical contour, but in (B.1) we restrict to $\Re(s) > 2$. Using $\Gamma(3-s)\sin(\pi s) = \pi/\Gamma(s-2)$ (cf. [1]) we see that the integral in (B.1) has simple poles at $s = 2-N$, $N \geq 0$ and hence

$$\frac{1}{2\pi i}\int_{\frac{5}{2}-i\infty}^{\frac{5}{2}+i\infty}\Gamma(s-2)2^{(m+\Delta-1)s}ds = \sum_{N=0}^{\infty}\frac{(-1)^N}{N!}2^{(2-N)(m+\Delta-1)} = 2^{2m+2\Delta-2}\exp(-2^{-m-\Delta+1}). \tag{B.2}$$

Using (B.2) in (B.1) and noting that $2^{2\Delta} = \xi^{-2}$ yields

$$\Phi_0(\xi) = \frac{1}{4\xi}\left[\prod_{\ell=1}^{\infty}(1-2^{-\ell})^{-2}\right]\sum_{m=-\infty}^{\infty}(-1)^{m+1}2^{-m^2/2}2^{5m/2}\exp(-2\xi 2^{-m}), \tag{B.3}$$

which establishes (3.5). The choice $\Re(s) > 2$ was somewhat arbitrary. However, if we choose *any* vertical contour, then the value of the integral in (B.2) will differ from the right side of (B.2) by the residues at a finite number of poles. Each such residue will be proportional to 2^{-Nm}, but this term will vanish after we evaluate the sum over m, in view of (3.6) and the comments below it. Thus, the final result for Φ_0 is independent of the contour chosen.

REFERENCES

[1] M. Abramowitz and I. Stegun, eds., *Handbook of Mathematical Functions*, John Wiley, New York, 1962.

[2] D. Aldous and P. Shields, *A diffusion limit for a class of random-growing binary trees*, Probab. Theory Related Fields, 79 (1988), pp. 509–542.

[3] G. Andrews, R. Askey, and R. Roy, *Special Functions*, Cambridge University Press, Cambridge, UK, 1999.

[4] C. Bender and S. Orszag, *Advanced Mathematical Methods for Scientists and Engineers*, McGraw-Hill, New York, 1978.

[5] L. Devroye, *A note on the probabilistic analysis of Patricia tries*, Random Structures Algorithms, 3 (1992), pp. 203–214.

[6] L. Devroye, *A study of trie-like structures under the density model*, Ann. Appl. Probab., 2 (1992), pp. 402–434.

[7] P. Flajolet and R. Sedgewick, *Digital search trees revisited*, SIAM J. Comput., 15 (1986), pp. 748–767.

[8] P. Flajolet and B. Richmond, *Generalized digital trees and their difference-differential equations*, Random Structures Algorithms, 3 (1992), pp. 305–320.

[9] P. Flajolet, X. Gourdon, and P. Dumas, *Mellin transforms and asymptotics: Harmonic sums*, Theoret. Comput. Sci., 144 (1995), pp. 3–58.

[10] N. Froman and P. Froman, *JWKB Approximation*, North-Holland, Amsterdam, 1965.

[11] E. Gilbert and T. Kadota, *The Lempel-Ziv algorithm and message complexity*, IEEE Trans. Inform. Theory, 38 (1992), pp. 1839–1842.

[12] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, Cambridge University Press, Cambridge, UK, 1997.

[13] P. Jacquet and W. Szpankowski, *Asymptotic behavior of the Lempel-Ziv parsing scheme and digital search trees*, Theoret. Comput. Sci., 144 (1995), pp. 161–197.
[14] P. Jacquet and W. Szpankowski, *Analytic depoissonization and its applications*, Theoret. Comput. Sci., 201 (1998), pp. 1–62.
[15] P. Jacquet, W. Szpankowski, and J. Tang, *Average profile of the Lempel-Ziv parsing scheme for a Markovian source*, Algorithmica, to appear.
[16] P. Kirschenhofer, H. Prodinger, and W. Szpankowski, *Digital search trees again revisited: The internal path length perspective*, SIAM J. Comput., 23 (1994), pp. 598–616.
[17] D. Knuth, *The Art of Computer Programming. Sorting and Searching*, 2nd ed., Addison-Wesley, Reading, MA, 1998.
[18] C. Knessl and W. Szpankowski, *Quicksort algorithm again revisited*, Discrete Math. Theor. Comput. Sci., 3 (1999), pp. 43–64.
[19] C. Knessl and W. Szpankowski, *Heights in generalized tries and PATRICIA tries*, in Proceedings of LATIN'2000, Punta del Este, Uruguay, 2000, Lecture Notes in Comput. Sci. 1776, Springer-Verlag, New York, 2000, pp. 298–307.
[20] A. Konheim and D.J. Newman, *A note on growing binary trees*, Discrete Math., 4 (1973), pp. 57–63.
[21] P. Lagerstrom, *Matched Asymptotic Expansions: Ideas and Techniques*, Springer-Verlag, New York, 1988.
[22] G. Louchard, *Exact and asymptotic distributions in digital and binary search trees*, RAIRO Theoretical Inform. Applications, 21 (1987), pp. 479–495.
[23] G. Louchard and W. Szpankowski, *Average profile and limiting distribution for a phrase size in the Lempel-Ziv parsing algorithm*, IEEE Trans. Inform. Theory, 41 (1995), pp. 478–488.
[24] H. Mahmoud, *Evolution of Random Search Trees*, John Wiley, New York, 1992.
[25] A. Odlyzko, *Asymptotic enumeration*, in Handbook of Combinatorics, Vol. II, R. Graham, M. Götschel, and L. Lovász, eds., Elsevier Science, 1995, pp. 1063–1229.
[26] B. Pittel, *Asymptotic growth of a class of random trees*, Ann. Probab., 13 (1985), pp. 414–427.
[27] A. Rényi, *On random subsets of a finite set*, Mathematica, 3 (1961), pp. 355–362.
[28] W. Szpankowski, *The evaluation of an alternative sum with applications to the analysis of some data structures*, Inform. Process. Lett., 28 (1988), pp. 13–19.
[29] W. Szpankowski, *A characterization of digital search trees from the successful search viewpoint*, Theoret. Comput. Sci., 85 (1991), pp. 117–134.
[30] W. Szpankowski, *On the height of digital trees and related problems*, Algorithmica, 6 (1991), pp. 256–277.
[31] J. Ziv and A. Lempel, *A universal algorithm for sequential data compression*, IEEE Trans. Inform. Theory, 23 (1977), pp. 337–343.
[32] J. Ziv and A. Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Trans. Inform. Theory, 24 (1978), pp. 530–536.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 965–977

SIMPLE CONFLUENTLY PERSISTENT CATENABLE LISTS*

HAIM KAPLAN†, CHRIS OKASAKI‡, AND ROBERT E. TARJAN§

Abstract. We consider the problem of maintaining persistent lists subject to concatenation and to insertions and deletions at both ends. Updates to a persistent data structure are nondestructive—each operation produces a new list incorporating the change, while keeping intact the list or lists to which it applies. Although general techniques exist for making data structures persistent, these techniques fail for structures that are subject to operations, such as catenation, that combine two or more versions. In this paper we develop a simple implementation of persistent double-ended queues (deques) with catenation that supports all deque operations in constant amortized time. Our implementation is functional if we allow memoization.

Key words. functional programming, data structures, persistent data structures, stack, queue, stack-ended queue (steque), double-ended queue (deque), memoization

AMS subject classifications. 68P05, 68Q25

PII. S0097539798339430

1. Introduction.

Over the last fifteen years, there has been considerable development of *persistent* data structures, those in which not only the current version but also older ones are available for access (*partial persistence*) or updating (*full persistence*). In particular, Driscoll et al. [5] developed efficient general methods to make pointer-based data structures partially or fully persistent, and Dietz [3] developed an efficient general method to make array-based structures fully persistent.

These general methods support updates that apply to a single version of a structure at a time, but they do not accommodate operations that combine two different versions of a structure, such as set union or list catenation. Driscoll, Sleator, and Tarjan [4] coined the term *confluently persistent* for fully persistent structures that support such combining operations. An alternative way to obtain persistence is to use purely functional programming. We take here an extremely strict view of pure functionality: we disallow lazy evaluation, memoization, and other such techniques. For list-based data structure design, purely functional programming amounts to using only the LISP functions CONS, CAR, CDR. Purely functional data structures are automatically persistent, and indeed confluently persistent.

A simple but important problem in data structure design that makes the issue of confluent persistence concrete is that of implementing persistent double-ended queues (deques) with catenation. A series of papers [2, 4] culminated in the work of Kaplan and Tarjan [11, 10], who developed a confluently persistent implementation of deques with catenation that has a worst-case constant time and space bound for any deque

*Received by the editors June 1, 1998; accepted for publication (in revised form) January 19, 2000; published electronically August 24, 2000.
http://www.siam.org/journals/sicomp/30-3/33943.html

†Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel (haimk@math.tau.ac.il).

‡Department of Computer Science, Columbia University, New York, NY 10027 (cokasaki@cs.columbia.edu). This author's research at Carnegie Mellon University was supported by the Advanced Research Projects Agency CSTO under the title "The Fox Project: Advanced Languages for Systems Software," ARPA Order C533, issued by ESC/ENS under contract F19628-95-C-0050.

§Department of Computer Science, Princeton University, Princeton, NJ 08544 and InterTrust Technologies Corporation, Sunnyvale, CA 94086 (ret@cs.princeton.edu). This author's research at Princeton University was partially supported by NSF grant CCR-9626862.

operation, including catenation. The Kaplan–Tarjan data structure and its precursors obtain confluent persistence by being purely functional.

If all one cares about is persistence, purely functional programming is unnecessarily restrictive. In particular, Okasaki [14, 15, 16] observed that the use of lazy evaluation in combination with memoization can lead to efficient functional (but not purely functional in our sense) data structures that are confluently persistent. In order to analyze such structures, Okasaki developed a novel kind of debit-based amortization. Using these techniques and weakening the time bound from worst-case to amortized, he was able to considerably simplify the Kaplan–Tarjan data structure, in particular to eliminate its complicated skeleton that encodes a tree extension of a redundant digital numbering system.

In this paper we explore the problem of further simplifying the Kaplan–Tarjan result. We obtain a confluently persistent implementation of deques with catenation that has a constant amortized time bound per operation. Our structure is substantially simpler than the original Kaplan–Tarjan structure, and even simpler than Okasaki's catenable deques: whereas Okasaki requires efficient persistent deques without catenation as building blocks, our structure is entirely self-contained. Furthermore our analysis uses a standard credit-based approach. We give two alternative, but closely related implementations of our method. The first uses memoization. The second, which saves a small constant factor in time and space, uses an extension of memoization in which any expression can replace an equivalent expression.

The remainder of the paper consists of five sections. In section 2, we introduce terminology and concepts. In section 3, we illustrate our approach by developing a persistent implementation of deques without catenation. In section 4, we extend our approach to handle stacks with catenation. In section 5, we develop our solution for deques with catenation. We conclude in section 6 with some remarks and open problems. An extended abstract of this work appeared in [9].

2. Preliminaries. The objects of our study are lists. As in [11, 10] we allow the following operations on lists:

MAKELIST(x):	return a new list containing the single element x.
PUSH(x, L):	return a new list formed by adding element x to the front of list L.
POP(L):	return a pair whose first component is the first element on list L and whose second component is a list containing the second through last elements of L.
INJECT(L, x):	return a new list formed by adding element x to the back of list L.
EJECT(L):	return a pair whose first component is a list containing all but the last element of L and whose second component is the last element of L.
CATENATE(L, R):	return a new list formed by catenating L and R, with L first.

We seek implementations of these operations (or specific subsets of them) on persistent lists: any operation is allowed on any previously constructed list or lists at any time. For discussions of various forms of persistence, see [5]. A *stack* is a list on which only PUSH and POP are allowed. A *queue* is a list on which only INJECT and POP are allowed. A *steque (stack-ended queue)* is a list on which only PUSH, POP, and INJECT are allowed. Finally, a *deque (double-ended queue)* is a list on which

all four operations PUSH, POP, INJECT, and EJECT are allowed. For any of these four structures, we may or may not allow catenation. If catenation is allowed, PUSH and INJECT become redundant, since they are special cases of catenation, but it is sometimes convenient to treat them as separate operations because they are easier to implement than general catenation.

We say a data structure is *purely functional* if it can be built and manipulated using the LISP functions CAR, CONS, CDR. That is, the structure consists of a set of immutable nodes, each either an atom or a node containing two pointers to other nodes, with no cycles of pointers. The nodes we use to build our structures actually contain a fixed number of fields; reducing our structures to two fields per node by adding additional nodes is straightforward. Various nodes in our structure represent lists.

To obtain our results, we extend pure functionality by allowing memoization, in which a function is evaluated only once on a node; the second time the same function is evaluated on the same node, the value is simply retrieved from the previous computation. In all our constructions, there are only a constant number of memoized functions (one or two). We can implement memoization by having a node point to the results of applying each memoized function to it. Initially each such pointer is undefined. The first function evaluation fills in the appropriate pointer to indicate the result. Subsequent evaluations merely follow the pointer to the result, which takes $O(1)$ time.

We also consider the use of a more substantial extension of pure functionality, in which we allow the operation of replacing a node in a structure by another node representing the same list. Such a replacement can be performed in an imperative setting by replacing all the fields in the node, for instance, in LISP by using REPLACA and REPLACD. Replacement can be viewed as a generalization of memoization. In our structures, any node is replaced at most twice, which means that all our structures can be implemented in a write-once memory. (It is easy to convert an algorithm that overwrites any field only a fixed constant number of times into a write-once algorithm, with only a constant-factor loss of efficiency.) The use of overwriting instead of memoization saves a small constant factor in running time and storage space and slightly simplifies the amortized analysis.

To perform amortized analysis, we use a standard potential-based framework. We assign to each configuration of the data structure (the totality of nodes currently existing) a *potential*. We define the amortized cost of an operation to be its actual cost plus the net increase in potential caused by performing the operation. In our applications, the potential of an empty structure is zero and the potential is always nonnegative. It follows that, for any sequence of operations starting with an empty structure, the total actual cost of the operations is bounded above by the sum of their amortized costs. See the survey paper [17] for a more complete discussion of amortized analysis.

3. Noncatenable deques. In this section we describe an implementation of persistent noncatenable deques with a constant amortized time bound per operation. The structure is based on the analogous Kaplan–Tarjan structure [11, 10] but is much simpler. The result presented here illustrates our technique for doing amortized analysis of a persistent data structure. At the end of the section we comment on the relation between the structure proposed here and previously existing solutions.

3.1. Representation. Here and in subsequent sections we say a data structure is *over* a set A if it stores elements from A. Our representation is recursive. It

is built from bounded-size deques called *buffers*, each containing at most three elements. Buffers are of two kinds: *prefixes* and *suffixes*. A nonempty deque d over A is represented by an ordered triple consisting of a prefix over A, denoted by $pr(d)$; a (possibly empty) *child deque* of ordered *pairs* over A, denoted by $c(d)$; and a suffix over A, denoted by $sf(d)$. Each pair consists of two elements from A. The child deque $c(d)$, if nonempty, is represented in the same way. We define the set of *descendants* $\{c^i(d)\}$ of a deque d in the standard way—namely, $c^0(d) = d$ and $c^{i+1}(d) = c(c^i(d))$, provided $c^i(d)$ and $c(c^i(d))$ exist.

The order of elements in a deque is defined recursively to be the one consistent with the order of each triple, each buffer, each pair, and each child deque. Thus, the order of elements in a deque d is first the elements of $pr(d)$, then the elements of each pair in $c(d)$, and finally the elements of $sf(d)$.

In general the representation of a deque is not unique—the same sequence of elements may be represented by triples that differ in the sizes of their prefixes and suffixes, as well as in the contents and representations of their descendant deques. Whenever we refer to a deque d we actually mean a particular representation of d, one that will be clear from the context.

The pointer representation for this representation is the obvious one: a node representing a deque d contains pointers to $pr(d)$, $c(d)$, and $sf(d)$. Note that the pointer structure of d is essentially a linked list of its descendants, since $c^i(d)$ contains a pointer to $c^{i+1}(d)$, for each i.

3.2. Operations. Implementing the deque operations is straightforward, except for maintaining the size bounds on buffers. Specifically, a PUSH on a deque is easy unless its prefix is of size three, a POP on a deque is easy unless its prefix is empty, and symmetric statements hold for INJECT and EJECT. We deal with buffer overflow and underflow in a proactive fashion, first fixing the buffer so that the operation to be performed cannot violate its size bounds, and then actually doing the operation. The details are as follows.

We define a buffer to be *green* if it contains one or two elements, and *red* if it contains zero or three. We define two memoized functions on a deque: GP, which constructs a representation of the same list but with a green prefix; and GS, which constructs a representation of the same list with a green suffix. We only apply GP (GS, respectively) to a list whose prefix (suffix) is red and can be made green. Specifically, for GP, if the prefix is empty, the child deque must be nonempty, and symmetrically for GS. Below we give implementations of PUSH, POP, and GP; the implementations for INJECT, EJECT, and GS are symmetric. We denote a deque with prefix p, child deque c, and suffix s by $[p, c, s]$. As mentioned in section 2, we can implement the memoization of GP and GS by having each node point to the nodes resulting from applying GP and GS to it; initially, such pointers are undefined.

PUSH(x, d): If $|pr(d)| = 3$, let $e = \text{GP}(d)$; otherwise, let $e = d$. Push x onto $pr(e)$ to form p' and return $[p', c(e), sf(e)]$.

POP(d): If $pr(d)$ is empty and $c(d)$ is not, let $e = \text{GP}(d)$; otherwise, let $e = d$. If $pr(e)$ is nonempty, let $(x, p) = \text{POP}(pr(e))$ and return the pair $(x, [p, c(e), sf(e)])$. Otherwise ($c(e)$ must be empty), let $(x, s) = \text{POP}(sf(e))$ and return the pair $(x, [\emptyset, \emptyset, s])$.

GP(d): If $|pr(d)| = 3$, let x, y, z be the three elements in $pr(d)$. Let p be a prefix containing only x, and let $c' = \text{PUSH}((y, z), c(d))$. Return $[p, c', sf(d)]$. Otherwise ($pr(d)$ is empty and $c(d)$ is not), let $((x, y), c') = \text{POP}(c(d))$ and let p' be a prefix containing x followed by y. Return $[p', c', sf(d)]$.

3.3. Analysis. The amortized analysis of this method relies on the memoization of GP and GS. We call a node representing a deque *secondary* if it is returned by a call to GP or GS and *primary* otherwise. If a secondary node y is constructed by a call GP(x) (GS(x), respectively), the only way to access y later is via another call GP(x) (GS(x), respectively): no secondary node is returned as the result of a PUSH, POP, INJECT, or EJECT operation. This means that GP and GS are called only on primary nodes.

We divide the nodes representing deques into three states: such a node is rr if both its buffers are red, gr if exactly one of its buffers is red, and gg if both its buffers are green. We subdivide the rr and gr states: an rr node is $rr0$ if neither GP nor GS has been applied to it, $rr1$ if exactly one of GP and GS has been applied to it, and $rr2$ if both GP and GS have been applied to it; a gr node is $gr0$ if neither GP nor GS has been applied to it, and $gr1$ otherwise. By the discussion above, every secondary node is $gr0$ or gg. We define $\#rr0$, $\#rr1$, and $\#gr0$ to be the numbers of *primary* nodes in states $rr0$, $rr1$, and $gr0$, respectively. We define the potential of a collection of nodes representing deques to be $4\#rr0 + 2\#rr1 + \#gr0$.

A call to PUSH is either terminal or results in a call to GP, which in turn calls PUSH. Similarly, a call to POP is either terminal or results in a call to GP, which in turn calls POP. We charge the $O(1)$ time spent in a call to GP (exclusive of the inner call to PUSH or POP) to the PUSH or POP that calls GP. A call to PUSH results in a sequence of recursive calls to PUSH (via calls to GP), of which the bottommost is *terminal* and the rest are *nonterminal*. A nonterminal PUSH has one of the following effects: it converts a primary $rr0$ node to $rr1$ and creates a new primary $gr0$ node (the result of the push) and a new secondary $gr0$ node (the result of the call to GP); it converts a primary $rr1$ node to $rr2$ and creates a new primary $gr0$ node and a new secondary $gr0$ node; or, it converts a primary $gr0$ node to $gr1$ and creates a new primary gg node and a new secondary gg node. In each case the total potential drops by one, paying for the time needed for the PUSH (excluding the recursive call). A terminal PUSH takes $O(1)$ time, creates $O(1)$ new nodes, and increases the potential by $O(1)$. We conclude that PUSH takes $O(1)$ amortized time. Analogous arguments apply to POP, INJECT, and EJECT, giving us the following theorem.

THEOREM 3.1. *Each of the operations* PUSH, POP, INJECT, *and* EJECT *defined above takes* $O(1)$ *amortized time.*

3.4. Implementation using overwriting. With the memoized implementation described above, a primary rr node can give rise to two secondary gr nodes representing the same list; a primary gr node can give rise to a secondary gg node representing the same list. These redundant representations exist simultanously. A gr representation, however, dominates an rr representation for performing deque operations, and a gg representation dominates a gr representation. This allows us to improve the efficiency of the implementation by using overwriting in place of memoization: when GP is called on a node, it overwrites the contents of the node with the results of the GP computation, and similarly for GS. Then only one representation of a list exists at any time, and it evolves from rr to gr to gg (via one of two alternative paths, depending on whether GP or GS is called first). Each node now needs only three fields (for prefix, child deque, and suffix) instead of five (two extra for GP and GS).

Not only does the use of overwriting save a constant factor in running time and storage space, but it also simplifies the amortized analysis, as follows. We define $\#rr$ and $\#gr$ to be the number of nodes in states rr and gr, respectively. (There

are now no secondary nodes.) We define the potential of a collection of nodes to be $3\#rr + \#gr$. A nonterminal PUSH has one of the following effects: it converts an rr node to gr and creates a new gr node, or converts a gr node to gg and creates a new gg node. In either case it reduces the potential by one, paying for the $O(1)$ time required by the PUSH (excluding the recursive call). A terminal PUSH takes $O(1)$ time and can increase the potential by $O(1)$. We conclude that PUSH takes $O(1)$ amortized time. Similar arguments apply to POP, INJECT, and EJECT.

3.5. Related work. The structure just described is based on the Kaplan–Tarjan structure of [10, section 4], but simplifies it in three ways. First, the skeleton of our structure (the sequence of descendants) is a stack; in the Kaplan–Tarjan structure, this skeleton must be partitioned into a stack of stacks in order to support worst-case constant-time operations (via a redundant binary counting mechanism). Second, the recursive changes to the structure to make its nodes green are one-sided, instead of two-sided: in the original structure, the stack-of-stacks mechanism requires coordination to keep both sides of the structure in related states. Third, the maximum buffer size is reduced, from five to three. In the special case of a steque, the maximum size of the suffix can be further reduced to two. In the special case of a queue, both the prefix and the suffix can be reduced to maximum size two.

There is an alternative, much older approach that uses incremental recopying to obtain persistent deques with worst-case constant-time operations. See [7] for a discussion of this approach. The incremental recopying approach yields an arguably simpler structure than the one presented here, but our structure generalizes to allow catenation, which no one knows how to implement efficiently using incremental recopying. Also, our structure can be extended to support access, insertion, and deletion d positions away from the end of a list in $O(\log d)$ amortized time, by applying the ideas in [12].

4. Catenable steques. In this section we show how to extend our ideas to support catenation. Specifically, we describe a data structure for catenable steques that achieves an $O(1)$ amortized time bound for PUSH, POP, INJECT, and CATENATE. The data structure is based on the same recursive decomposition of lists as that in section 5 of [10]. The pointer structure that we need here is much simpler than that in [10], and the analysis is amortized, following the framework outlined in section 2 and used in section 3.

4.1. Representation. Our structure is similar to the structure of section 3, but with slightly different definitions of the component parts. As in section 3, we use buffers of two kinds: prefixes and suffixes. Each prefix contains two to six elements and each suffix contains one to three elements. A nonempty steque d over A is represented either by a suffix $sf(d)$ only, or by an ordered triple consisting of a prefix $pr(d)$ over A, a child steque $c(d)$ of pairs over A, and a suffix $sf(d)$ over A. In contrast to section 3, a *pair over* A is defined to be an ordered pair containing a prefix and a possibly empty steque of pairs over A. Observe that this definition adds an additional kind of recursion (pairs store steques) to the structure of section 3. This extra kind of recursion is what allows efficient catenation.

The order of elements in a steque is the one consistent with the order of each triple, each buffer, each pair, each steque within a pair, and each child steque. As in section 3, there can be many different representations of a steque containing a given list of elements; when speaking of a steque, we mean a particular representation of it.

The pointer structure for this representation is straightforward. Each triple is

represented by a node containing three pointers: to a prefix, a child steque, and a suffix. Each pair is represented by a node containing two pointers: to a prefix and a steque.

4.2. Operations. The implementation of the steque operations is much like the implementation of the noncatenable deque operations presented in section 3.2. We call a prefix *red* if it contains either two or six elements, and *green* otherwise. We call a suffix *red* if it contains three elements, and *green* otherwise. The prefix in a suffix-only steque is considered to have the same color as the suffix. We define two memoized functions, GP and GS, which produce green-prefix and green-suffix representations of a steque, respectively. Each is called only when the corresponding buffer is red and can be made green. We define PUSH, POP, and INJECT to call GP or GS when necessary to obtain a green buffer. In the definitions below, we represent a steque with prefix p, child steque c, and suffix s by $[p, c, s]$.

PUSH(x, d):

Case 1. Steque d is represented by a triple. If $|pr(d)| = 6$, then let $e = \text{GP}(d)$; otherwise, let $e = d$. Let $p = \text{PUSH}(x, pr(e))$ and return $[p, c(e), sf(e)]$.

Case 2. Steque d is represented by a suffix only. If $|sf(d)| = 3$, create a prefix p containing x and the first two elements of $sf(d)$, create a suffix s containing the last element of $sf(d)$, and return $[p, \emptyset, s]$. Otherwise, create a suffix s by pushing x onto $sf(d)$ and return $[\emptyset, \emptyset, s]$.

INJECT(d, x):

Case 1. Steque d is represented by a triple. If $|sf(d)| = 3$, let $e = \text{GS}(d)$; otherwise, let $e = d$. Let $s = \text{INJECT}(sf(e), x)$ and return $[pr(e), c(e), s]$.

Case 2: Steque d is represented by a suffix only. If $|sf(d)| = 3$, create a suffix s containing x, and return $[sf(d), \emptyset, s]$. Otherwise, create a suffix s by injecting x into $sf(d)$ and return $[\emptyset, \emptyset, s]$.

CATENATE(d_1, d_2):

Case 1. d_1 and d_2 are represented by triples. First, catenate the buffers $sf(d_1)$ and $pr(d_2)$ to obtain p. Now, calculate c' as follows: If $|p| \leq 5$, then let $c' = \text{INJECT}(c(d_1), (p, c(d_2)))$. Otherwise, $6 \leq |p| \leq 9$. Create two new prefixes p' and p'', with p' containing the first four elements of p and p'' containing the remaining elements. Let $c' = \text{INJECT}(\text{INJECT}(c(d_1), (p', \emptyset)), (p'', c(d_2)))$. In either case, return $[pr(d_1), c', sf(d_2)]$.

Case 2. d_1 or d_2 is represented by a suffix only. Push or inject the elements of the suffix-only steque one by one into the other steque.

Note that both PUSH and CATENATE produce valid steques even when their second arguments are steques with prefixes of length one. Although such steques are not normally allowed, they may exist transiently during a POP. Every such steque is immediately passed to PUSH or CATENATE, and then discarded, however. In order to define the POP, GP, and GS operations, we define a NAIVE-POP operation that simply pops its steque argument without making sure that the result is a valid steque.

NAIVE-POP(d): If d is represented by a triple, let $(x, p) = \text{POP}(pr(d))$ and return the pair $(x, [p, c(d), sf(d)])$. If d consists of a suffix only, let $(x, s) = \text{POP}(sf(d))$ and return the pair $(x, \emptyset)$ if $|d| = 1$ or $(x, [\emptyset, \emptyset, s])$ if $|d| > 1$.

POP(d):

Case 1. Steque d is represented by a suffix only or $|pr(d)| > 2$. Return NAIVE-POP(d).

Case 2. Steque d is represented by a triple, $|pr(d)| = 2$, and $c(d) = \emptyset$. Let x be the first element on pr(d) and y the second. If $|sf(d)| < 3$, push y onto $sf(d)$ to form s and return $(x, [\emptyset, \emptyset, s])$. Otherwise ($|sf(d)| = 3$), form p from y and the first two elements on $sf(d)$, form s from the last element on $sf(d)$, and return $(x, [p, \emptyset, s])$.

Case 3. Steque d is represented by a triple, $|pr(d)| = 2$, and $c(d) \neq \emptyset$. Return NAIVE-POP(GP(d)).

GP(d): If $|pr(d)| = 6$, then create two new prefixes p and p' by splitting $pr(d)$ equally in two. Let $c' =$ PUSH$((p', \emptyset), c(d))$. Return $[p, c', sf(d)]$. Otherwise ($|pr(d)| = 2$ and $c(d) \neq \emptyset$), proceed as follows. Inspect the first pair (p, d') in $c(d)$. If $|p| \geq 4$ or d' is not empty, let $((p, d'), c') =$ NAIVE-POP$(c(d))$; otherwise, let $((p, d'), c') =$ POP$(c(d))$. Now inspect p.

Case 1. p contains at least four elements. Pop the first two elements from p to form p'' and inject these two elements into $pr(d)$ to obtain p'. Let $c'' =$ PUSH$((p'', d'), c')$. Return $[p', c'', sf(d)]$.

Case 2. p contains at most three elements. Push the two elements in $pr(d)$ onto p to obtain p'. Let $c'' =$ CATENATE(d', c') if d' is nonempty, or $c'' = c'$ if d' is empty. Return $[p', c'', sf(d)]$.

GS(d): (Steque d is represented by a triple with $|sf(d)| = 3$) Let p contain the first two elements of $sf(d)$ and s the last element on $sf(d)$. Let $c' =$ INJECT$(c(d), (p, \emptyset))$. Return $[pr(d), c', s]$.

4.3. Analysis. The analysis of this method is similar to the analysis in section 3.3. We define primary and secondary nodes, node states, and the potential function exactly as in section 3.3: the potential function defined there is $4\#rr0+2\#rr1+\#gr0$, where $\#rr0$, $\#rr1$, and $\#gr0$ are the numbers of primary nodes in states $rr0$, $rr1$, and $gr0$, respectively.

As in section 3.3, we charge the $O(1)$ cost of a call to GP or GS (excluding the cost of any recursive call to PUSH, POP, or INJECT) to the PUSH, POP, or INJECT that calls GP or GS. The amortized costs of PUSH and INJECT are $O(1)$ by an argument identical to that used to analyze PUSH in section 3.3. Operation CATENATE calls PUSH and INJECT a constant number of times and creates a single new node, so its amortized cost is also $O(1)$.

To analyze POP, assume that a call to POP recurs to depth k (via intervening calls to GP). By an argument analogous to that for PUSH, each of the first $k-1$ calls pays for itself by decreasing the potential by one. The terminal call to POP can result in a call to either PUSH or CATENATE, each of which has $O(1)$ amortized cost. It follows that the overall amortized cost of POP is $O(1)$, giving us the following theorem.

THEOREM 4.1. *Each of the operations* PUSH, POP, INJECT, *and* CATENATE *defined above takes* $O(1)$ *amortized time.*

We can improve the time and space efficiency of the steque data structure by constant factors by using overwriting in place of memoization, exactly as described in section 3.4. If we do this, we can also simplify the amortized analysis, again exactly as described in section 3.4.

4.4. Related work. The structure presented in this section is analogous to the Kaplan–Tarjan structure of [10, section 5] and the structure of [8, section 7], but simplifies them as follows. First, the buffers are of constant-bounded size, whereas the structure of [10, section 5] uses noncatenable steques as buffers, and the structure of [8, section 7] uses noncatenable stacks as buffers. These buffers in turn must be represented as in section 3 of this paper or by using one of the other methods

mentioned there. In contrast, the structure of the present section is entirely self-contained. Second, the skeleton of the present structure is just a stack, instead of a stack of stacks as in [10] and [8]. Third, the changes used to make buffers green are applied in a one-sided, need-driven way; in [10] and [8], repairs must be made simultaneously to both sides of the structure in carefully chosen locations.

Okasaki [14] has devised a different and somewhat simpler implementation of confluently persistent catenable steques that also achieves an $O(1)$ amortized bound per operation. His solution obtains its efficiency by (implicitly) using a form of path reversal [18] in addition to lazy evaluation and memoization. Our structure extends to the double-ended case, as we shall see in the next section; whether Okasaki's structure extends to this case is an open problem.

5. Catenable deques. In this section we show how to extend our ideas to support all five list operations. Specifically, we describe a data structure for catenable deques that achieves an $O(1)$ amortized time bound for PUSH, POP, INJECT, EJECT, and CATENATE. Our structure is based upon an analogous structure of Okasaki [16], but simplified to use constant-size buffers.

5.1. Representation. We use three kinds of buffers: *prefixes*, *middles*, and *suffixes*. A nonempty deque d over A is represented either by a suffix $sf(d)$ or by a 5-tuple that consists of a prefix $pr(d)$, a left deque of triples $ld(d)$, a middle $md(d)$, a right deque of triples $rd(d)$, and a suffix $sf(d)$. A *triple* consists of a *first middle buffer*, a deque of triples, and a *last middle buffer*. One of the two middle buffers in a triple must be nonempty, and in a triple that contains a nonempty deque both middles must be nonempty. All buffers and triples are over A. A prefix or suffix in a 5-tuple contains three to six elements, a suffix in a suffix-only representation contains one to eight elements, a middle in a 5-tuple contains exactly two elements, and a nonempty middle buffer in a triple contains two or three elements.

The order of elements in a deque is the one consistent with the order of each 5-tuple, each buffer, each triple, and each recursive deque. The pointer structure is again straightforward, with the nodes representing 5-tuples or triples containing one pointer for each field.

5.2. Operations. We call a prefix or suffix in a 5-tuple *red* if it contains either three or six elements and *green* otherwise. We call a suffix in a suffix-only representation *red* if it contains eight elements and *green* otherwise. The prefix of a suffix-only deque is considered to have the same color as the suffix. We introduce two memoizing functions GP and GS as in sections 3.2 and 4.2, which produce green-prefix and green-suffix representations of a deque, respectively, and which are called only when the corresponding buffer is red but can be made green. Below we give the implementations of PUSH, POP, GP, and CATENATE; the implementations of INJECT, EJECT, and GS are symmetric to those of PUSH, POP, and GP, respectively. We denote a deque with prefix p, left deque l, middle m, right deque r, and suffix s by $[p, l, m, r, s]$.

PUSH(x, d):

Case 1. Deque d is represented by a 5-tuple. If $|pr(d)| = 6$, then let $e = \text{GP}(d)$; otherwise, let $e = d$. Return $[\text{PUSH}(x, pr(e)), ld(e), md(e), rd(e), sf(e)]$.

Case 2. Deque d is represented by a suffix only. If $sf(d) < 8$, return a suffix-only deque with suffix $\text{PUSH}(x, sf(d))$. Otherwise, push x onto $sf(d)$ to form s, with nine elements. Create a new prefix p with the first four, a middle with the next two, and a suffix s with the last three. Return $[p, \emptyset, m, \emptyset, s]$.

As in section 4.2, the implementation of POP uses NAIVE-POP.

POP(d):

Case 1. Deque d is represented by a suffix only or $|pr(d)| > 3$. Return NAIVE-POP(d).

Case 2. $|pr(d)| = 3$ and $ld(d) \cup rd(d) \neq \emptyset$. Return NAIVE-POP(GP($d$)).

Case 3. $|pr(d)| = 3$ and $ld(d) \cup rd(d) = \emptyset$. Let x be the first element on $pr(d)$. If $|sf(d)| = 3$, create a new suffix s containing all the elements in $pr(d)$, $md(d)$, and $sf(d)$ except x, and return the pair consisting of x and the deque represented by s only. Otherwise, form p from $pr(d)$ by popping x and injecting the first element on $md(d)$, form m' from $md(d)$ by popping the first element and injecting the first element on $sf(d)$, form s from $sf(d)$ by popping the first element, and return $(x, [p, \emptyset, m', \emptyset, s])$.

GP(d): If $|pr(d)| = 6$, create two new prefixes p and p', with p containing the first four elements of $|pr(d)|$ and p' the last two; return $[p, \text{PUSH}((p', \emptyset, \emptyset), ld(d)), md(d), rd(d), sf(d)]$. Otherwise ($|pr(d)| = 3$ and $ld(d) \cup rd(d) \neq \emptyset$); proceed as follows.

Case 1. $ld(d) \neq \emptyset$. Inspect the first triple t on $ld(d)$. If either the first nonempty middle buffer in t contains three elements or t contains a nonempty deque, let $(t, l) =$ NAIVE-POP($ld(d)$); otherwise let $(t, l) =$ POP($ld(d)$). Let $t = (x, d', y)$ and assume that x is nonempty if t consists of only one nonempty middle buffer. Apply the appropriate one of the following two subcases.

Case 1.1. $|x| = 3$. Form x' from x and p from $pr(d)$ by popping the first element from x and injecting it into $pr(d)$. Return $[p, \text{PUSH}((x', d', y), l), md(d), rd(d), sf(d)]$.

Case 1.2. $|x| = 2$. Inject both elements in x into pr(d) to form p. If d' and y are empty, return $[p, l, md(d), rd(d), sf(d)]$. Otherwise ($d'$ and y are nonempty) let $l' =$ CATENATE(d', PUSH($(y, \emptyset, \emptyset), l)$) and return $[p, l', md(d), rd(d), sf(d)]$.

Case 2. $ld(d) = \emptyset$ and $rd(d) \neq \emptyset$. Inspect the first triple t in $rd(d)$. If either the first nonempty middle buffer in t contains three elements or t contains a nonempty deque, let $(t, r) =$ NAIVE-POP($rd(d)$); otherwise, let $(t, r) =$ POP($rd(d)$). Let $t = (x, d', y)$ and assume that x is nonempty if t consists of only one nonempty middle buffer. Of the following two subcases, apply the appropriate one.

Case 2.1. $|x| = 3$. Form p, m', and x' from $pr(d)$, m, and x by popping an element from m and injecting it into $pr(d)$ to form p, popping an element from m and injecting the first element from x to form m', and popping the first element from x to form x'. Return $[p, \emptyset, m', \text{PUSH}((x', d', y), r), sf(d)]$.

Case 2.2. $|x| = 2$. Inject the two elements in $md(d)$ into $pr(d)$ to form p. Let $r' = r$ if d' and y are empty or $r' =$ CATENATE(d', PUSH($(y, \emptyset, \emptyset), r)$) otherwise. Return $[p, \emptyset, x, r', sf(d)]$.

CATENATE(d_1, d_2):

Case 1. Both d_1 and d_2 are represented by 5-tuples. Let y be the first element in $pr(d_2)$, and let x be the last element in $sf(d_1)$. Create a new middle m containing x followed by y. Partition the elements in $sf(d_1) - \{x\}$ into at most two buffers s_1' and s_1'', each containing two or three elements in order, with s_1'' possibly empty. Let $ld_1' =$ INJECT($ld(d_1), (md(d_1), rd(d_1), s_1')$). If $s_1'' \neq \emptyset$, then let $ld_1'' =$ INJECT($ld_1', (s_1'', \emptyset, \emptyset)$); otherwise, let $ld_1'' = ld_1'$. Similarly, partition the elements in $pr(d_1) - \{y\}$ into at most two prefixes p_2' and p_2'', each containing two or three elements in order, with p_2' possibly empty. Let $rd_2' =$ PUSH($(p_2'', ld(d_2), md(d_2)), rd(d_2)$). If $p_2' \neq \emptyset$ let $rd_2'' =$ PUSH($(p_2', \emptyset, \emptyset), rd_2'$); otherwise, let $rd_2'' = rd_2'$. Return $[pr(d_1), ld_1'', m, rd_2'', sf(d_2)]$.

Case 2. d_1 or d_2 is represented by a suffix only. Push or inject the elements of the suffix-only deque one by one into the other deque.

5.3. Analysis. To analyze this structure, we use the same definitions and the same potential function as in sections 3.3 and 4.3. The amortized costs of PUSH, INJECT, CATENATE, and POP are $O(1)$ by an argument analogous to that in section 4.3. The amortized cost of EJECT is $O(1)$ by an argument symmetric to that for POP. Thus we obtain the following theorem.

THEOREM 5.1. *Each of the operations* PUSH, POP, INJECT, EJECT, *and* CATENATE *defined above takes* $O(1)$ *amortized time.*

Just as in sections 3.4 and 4.3, we can improve the time and space constant factors and simplify the analysis by using overwriting in place of memoization. Overwriting is the preferred implementation, unless one is using a functional programming language that supports memoization but does not easily allow overwriting.

5.4. Related work. The structure presented in this section is analogous to the structures of [16, Chapter 11] and [8, section 9] but simplifies them as follows. First, the buffers are of constant size, whereas in [16] and [8] they are noncatenable deques. Second, the skeleton of the present structure is a binary tree, instead of a tree extension of a redundant digital numbering system, as in [8]. Also, our amortized analysis uses the standard potential function method of [17] rather than the more complicated debit mechanism used in [16]. Another related structure is that of [10, section 5], which represents purely functional, real-time deques as pairs of triples rather than 5-tuples, but otherwise is similar to (but simpler than) the structure of [8, section 9]. It is straightforward to modify the structure presented here to use pairs of triples rather than 5-tuples.

6. Further results and open questions. If the universe A of elements over which deques are constructed has a total order, we can extend the structures described here to support an additional heap order based on the order on A. Specifically, we can support the additional operation of finding the minimum element in a deque (but not deleting it) while preserving a constant amortized time bound for every operation, including finding the minimum. We merely have to store with each buffer, each deque, and each pair or triple the minimum element in it. For related work, see [1, 2, 6, 13].

We can also support a *flip* operation on deques. A flip operation reverses the linear order of the elements in the deque: the ith from the front becomes the ith from the back, and vice versa. For the noncatenable deques of section 3, we implement flip by maintaining a *reversal bit* that is flipped by a flip operation. If the reversal bit is set, a push becomes an inject, a pop becomes an eject, an inject becomes a push, and an eject becomes a pop. To support catenation as well as flip we use reversal bits at all levels. We must also symmetrize the definition in section 5 to allow a deque to be represented by a prefix only and extend the various operations to handle this possibility. The interpretation of reversal bits is cumulative. That is, if d is a deque and x is a deque inside of d, x is regarded as being reversed if an odd number of reversal bits are set to 1 along the path of actual pointers in the structure from the node for d to the node for x. Before performing catenation, if the reversal bit of either or both of the two deques is 1, we push such bits down by flipping such a bit of a deque x to 0, flipping the bits of all the deques to which x points, and swapping the appropriate buffers and deques. (The prefix and suffix exchange roles, as do the left deque and right deque; the order of elements in the prefix and suffix is reversed as well.) We do such push-downs of reversal bits by assembling new deques, not by overwriting the old ones.

We have devised an alternative implementation of catenable deques in which the sizes of the prefixes and suffixes are between 3 and 5 instead of 3 and 6. We do this by

memoizing the POP and EJECT operations and avoiding creating a new structure with a green prefix (suffix, respectively) representing the original deque when performing POP (EJECT, respectively). Using a more complicated potential function than the ones used in earlier sections, we can show that such an implementation runs in $O(1)$ amortized time per operation.

One direction for future research is to find a way to simplify our structures further. Specifically, consider the following alternative representation of catenable deques, which uses a single recursive subdeque rather than two such subdeques. A nonempty deque d over A is represented by a triple that consists of a prefix $pr(d)$, a (possibly empty) child deque of triples $c(d)$, and a suffix $sf(d)$. A *triple* consists of a nonempty *prefix*, a deque of triples, and a nonempty *suffix*, or just of a nonempty prefix or suffix. All buffers and triples are over A. The operations PUSH, POP, INJECT, and EJECT have implementations similar to their implementations in section 5. The major difference is in the implementation of CATENATE, which for this structure requires a call to POP. Specifically, let d_1 and d_2 be two deques to be catenated. CATENATE pops $c(d_1)$ to obtain a triple (p, d', s) and a new deque c, injects $(s, c, sf(d_1))$ into d' to obtain d'', and then pushes $(p, d'', pr(d_2))$ onto $c(d_2)$ to obtain c'. The final result has prefix $pr(d_1)$, child deque c', and suffix $sf(d_2)$. It is an open question whether this algorithm runs in constant amortized time per operation for any constant upper and lower bounds on the buffer sizes.

Another research direction is to design a confluently persistent representation of sorted lists such that accesses or updates d positions from either end take $O(\log d)$ time, and catenation takes $O(1)$ time. The best structure so far developed for this problem has a doubly logarithmic catenation time [12]; it is purely functional, and the time bounds are worst-case.

Acknowledgments. We thank Michael Goldwasser for a detailed reading of this paper and Jason Hartline for discussions that led to our implementations using memoization.

REFERENCES

[1] A. L. Buchsbaum, R. Sundar, and R. E. Tarjan, *Data-structural bootstrapping, linear path compression, and catenable heap-ordered double-ended queues*, SIAM J. Comput., 24 (1995), pp. 1190–1206.

[2] A. L. Buchsbaum and R. E. Tarjan, *Confluently persistent deques via data structural bootstrapping*, J. Algorithms, 18 (1995), pp. 513–547.

[3] P. F. Dietz, *Fully persistent arrays*, in Proceedings of the 1989 Workshop on Algorithms and Data Structures (WADS'89), Lecture Notes in Comput. Sci. 382, Springer-Verlag, New York, 1995, pp. 67–74.

[4] J. Driscoll, D. Sleator, and R. Tarjan, *Fully persistent lists with catenation*, J. ACM, 41 (1994), pp. 943–959.

[5] J. R. Driscoll, N. Sarnak, D. Sleator, and R. Tarjan, *Making data structures persistent*, J. Comput. System Sci., 38 (1989), pp. 86–124.

[6] H. Gajewska and R. E. Tarjan, *Deques with heap order*, Inform. Process. Lett., 12 (1986), pp. 197–200.

[7] R. Hood, *The Efficient Implementation of Very-High-Level Programming Language Constructs*, Ph.D. thesis, TR 82-503, Department of Computer Science, Cornell University, Ithaca, NY, 1982.

[8] H. Kaplan, *Purely Functional Lists*, Ph.D. thesis, Department of Computer Science, Princeton University, Princeton, NJ, 1997.

[9] H. Kaplan, C. Okasaki, and R. E. Tarjan, *Simple confluently persistent catenable lists* (extended abstract), in Proceedings of the Scandinavian Workshop on Algorithm Theory (SWAT), Stockholm, Sweden, Springer-Verlag, New York, 1998, pp. 119–130.

[10] H. Kaplan and R. E. Tarjan, *Purely functional, real-time deques with catenation*, J. ACM, 46 (1999), pp. 577–603.

[11] H. Kaplan and R. E. Tarjan, *Persistent lists with catenation via recursive slow-down* (preliminary version), in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, ACM Press, New York, 1995, pp. 93–102.

[12] H. Kaplan and R. E. Tarjan, *Purely functional representations of catenable sorted lists*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, ACM Press, New York, 1996, pp. 202–211.

[13] S. R. Kosaraju, *An optimal RAM implementation of catenable min double-ended queues*, in Proceedings of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, SIAM, Philadelphia, 1994, pp. 195–203.

[14] C. Okasaki, *Amortization, lazy evaluation, and persistence: Lists with catenation via lazy linking*, in Proceedings of the 36th Symposium on Foundations of Computer Science, Milwaukee, WI, IEEE, Piscataway, NJ, 1995, pp. 646–654.

[15] C. Okasaki, *Simple and efficient purely functional queues and deques*, J. Funct. Progamming, 5 (1995), pp. 583–592.

[16] C. Okasaki, *Purely functional data structures*, Ph.D. thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1996.

[17] R. E. Tarjan, *Amortized computational complexity*, SIAM J. Alg. Discrete Methods, 6 (1985), pp. 306–318.

[18] R. E. Tarjan and J. V. Leeuwen, *Worst case analysis of set union algorithms*, J. ACM, 31 (1984), pp. 245–281.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 978–989

APPROXIMATING THE STRETCH FACTOR OF EUCLIDEAN GRAPHS*

GIRI NARASIMHAN† AND MICHIEL SMID‡

Abstract. There are several results available in the literature dealing with efficient construction of t-spanners for a given set S of n points in $\mathbb{R}^d$. t-spanners are Euclidean graphs in which distances between vertices in G are at most t times the Euclidean distances between them; in other words, distances in G are "stretched" by a factor of at most t. We consider the interesting dual problem: given a Euclidean graph G whose vertex set corresponds to the set S, compute the stretch factor of G, i.e., the maximum ratio between distances in G and the corresponding Euclidean distances. It can trivially be solved by solving the all-pairs-shortest-path problem. However, if an approximation to the stretch factor is sufficient, then we show it can be efficiently computed by making only $O(n)$ approximate shortest path queries in the graph G. We apply this surprising result to obtain efficient algorithms for approximating the stretch factor of Euclidean graphs such as paths, cycles, trees, planar graphs, and general graphs. The main idea behind the algorithm is to use Callahan and Kosaraju's well-separated pair decomposition.

Key words. computational geometry, spanners, approximate shortest paths, well-separated pairs

AMS subject classification. 68U05

PII. S0097539799361671

1. Introduction.

Let S be a set of n points in $\mathbb{R}^d$, where $d \geq 1$ is a small constant, and let G be an undirected connected graph having the points of S as its vertices. The length of any edge (p, q) of G is defined as the Euclidean distance $|pq|$ between the two vertices p and q. Such graphs are called *Euclidean graphs*. The length of a path in G is defined as the sum of the lengths of all edges on this path. For any two vertices p and q of G, we denote by $|pq|_G$ the distance in G between them, i.e., the length of a shortest path connecting p and q.

Let $t > 1$ be a real number. We say that G is a *t-spanner* for S if for each pair of points $p, q \in S$, we have $|pq|_G \leq t \cdot |pq|$, i.e., there exists a path in G between p and q of length at most t times the Euclidean distance between these two points.

The smallest t such that G is a t-spanner for S is called the *stretch factor* of G (also referred to as *dilation* [21] or *distortion* [19] in the literature). We will denote the stretch factor by t^*. Note that

$$t^* = \max \left\{ \frac{|pq|_G}{|pq|} : p, q \in S, p \neq q \right\}.$$

Most of the earlier research considered the problem of constructing or analyzing geometric t-spanners for a given set of points. In this paper, we consider the interesting dual problem, stated as follows.

*Received by the editors September 27, 1999; accepted for publication (in revised form) February 25, 2000; published electronically August 24, 2000.
http://www.siam.org/journals/sicomp/30-3/36167.html

†Department of Mathematical Sciences, The University of Memphis, Memphis, TN 38152 (giri@msci.memphis.edu). The research of this author was supported by NSF grant CCR-940-9752 and a grant from Cadence Design Systems.

‡Department of Computer Science, University of Magdeburg, D-39106 Magdeburg, Germany (michiel@isg.cs.uni-magdeburg.de).

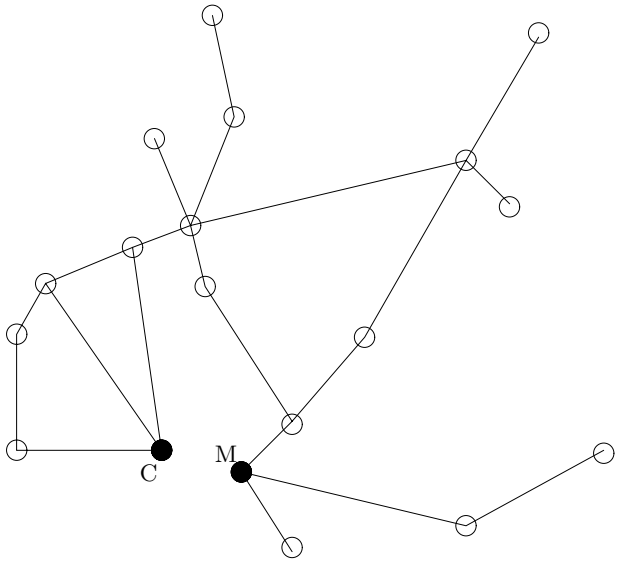


FIG. 1. *A section of the Scandinavian rail network; C denotes Copenhagen and M denotes Malmö.*

PROBLEM. *Given a set S of n points in $\mathbb{R}^d$, and a connected Euclidean graph with vertices from S, design an efficient algorithm to compute (exactly or approximately) its stretch factor.*

Spanners have applications in network design, robotics, distributed algorithms, and many other areas and have been the subject of considerable research [1, 4, 8, 11, 14, 18, 24]. More recently, spanners have received a lot of attention by researchers with the discovery of new applications for them in the design of approximation algorithms for geometric optimization problems such as the Euclidean traveling salesperson problem [3, 23].

If the graph represents, say, a network of highways, then the stretch factor is a measure of the maximum percentage increase in driving distance for using the network of highways over the direct "as-the-crow-flies" distance. Thus the stretch factor of a network is an important parameter to be considered when evaluating and analyzing networks. Furthermore, determining the two vertices in the network for which this increase is maximized helps to identify the "weakest" part of the network in terms of distances.

Figure 1 shows a section of the Scandinavian rail network. It is clear that the stretch factor in this network is determined by Copenhagen (marked C) and Malmö (marked M). A link between these two cities would drastically reduce the stretch factor.[1]

Let $G = (S, E)$ be a Euclidean graph and let $n := |S|$ and $m := |E|$. Clearly, the time complexity of solving the *all-pairs-shortest-path* problem for G is an upper bound on the time complexity of computing the stretch factor of G. Hence, running Dijkstra's algorithm—implemented with Fibonacci heaps—from each vertex of G gives the stretch factor of G in $O(n^2 \log n + nm)$ time (cf. [10]). For some classes of graphs, better running times can be obtained. For example, if G is a planar Euclidean graph, Frederickson [15] has shown that the distances in G between all pairs of vertices can

[1] A 16-kilometer bridge across the Øresund connecting the two cities is currently being built.

be computed in $O(n^2)$ total time. Therefore, the stretch factor of a planar Euclidean graph can be computed in $O(n^2)$ time.

We are not aware of any algorithms that compute the stretch factor in subquadratic time for any class of connected Euclidean graphs. (Exceptions are trivial classes of graphs, such as complete graphs, which have stretch factor one.) For example, we do not even know if the stretch factor of a Euclidean path can be computed in $o(n^2)$ time. This leads to the question whether there are faster algorithms that *approximate* stretch factors.

Let G be a connected Euclidean graph with stretch factor t^*, and let $c_1 \geq 1$, $c_2 \geq 1$, and $t \geq 1$ be real numbers. We say that t is a (c_1, c_2)-*approximate stretch factor* of G if

$$\frac{1}{c_1}\, t \leq t^* \leq c_2 t.$$

1.1. Our results. The results of this paper are as follows.

1. Using the *well-separated pair decomposition* of Callahan and Kosaraju [7], we reduce the problem of approximating the stretch factor of any Euclidean graph G to a sequence of $O(n)$ approximate shortest path queries in G.
2. We prove that, in the algebraic computation tree model, any algorithm that takes as input any connected Euclidean graph G with n vertices and computes an approximation to the stretch factor of G takes $\Omega(n \log n)$ time in the worst case.
3. For any real constant $\epsilon > 0$, we can compute in $O(n \log n)$ time a $(1, 1 + \epsilon)$-approximate stretch factor of any Euclidean path, cycle, or tree with n vertices. By the previous result, this is optimal in the algebraic computation tree model.
4. For any real constant $\epsilon > 0$, we can compute in $O(n\sqrt{n}\,)$ time a $(1, 1 + \epsilon)$-approximate stretch factor of any planar Euclidean graph with n vertices.
5. For any integer constant $\beta \geq 1$ and real constant $\epsilon > 0$, we can compute in $O(mn^{1/\beta} \log^2 n)$ expected time a $(2\beta(1+\epsilon), 1+\epsilon)$-approximate stretch factor of any Euclidean graph with n vertices and m edges.

In our first algorithm (Algorithm $\mathcal{A}$), the stretch factor is approximated by making farthest pair queries on $O(n)$ pairs of *sets of points*. Except for graphs such as paths, cycles, trees, and planar graphs, it is not clear how such queries can be solved efficiently. Our second algorithm (Algorithm $\mathcal{B}$) is much simpler; it makes shortest path queries for $O(n)$ specific pairs of *points*. The time complexity of Algorithm $\mathcal{B}$ is consequently improved over the corresponding one for Algorithm $\mathcal{A}$.

It is interesting that $O(n)$ approximate shortest path queries are sufficient to approximate the stretch factor. It is also interesting to note the pairs of points on which these $O(n)$ shortest path queries are made. In Algorithm $\mathcal{B}$, these linear number of queries depend *only* on the positions of the vertices; they do *not* depend on the edges of the graph G. Finally, our algorithms also determine two vertices for which the stretch factor is approximately maximized.

1.2. Related work. As mentioned already, our reduction uses the well-separated pair decomposition of [7], thus adding to the list of applications of this powerful method. For other applications of this decomposition, see [4, 6, 7].

Some related research in the general direction of approximating stretch factors include papers by Dobkin, Friedman, and Supowit [12] and Keil and Gutwin [17], which showed that the Delaunay triangulation has a stretch factor bounded by a

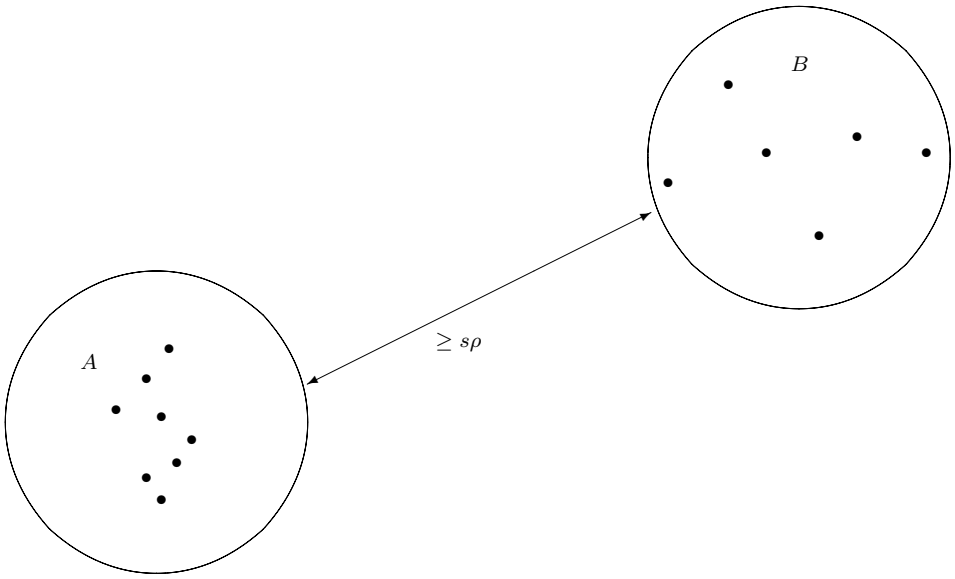


FIG. 2. *Two planar point sets A and B that are well separated with respect to s. Both circles have radius ρ; their distance is at least $s\rho$.*

small constant, and a paper by Eppstein [13], which showed that a certain class of Euclidean graphs (called beta-skeletons) can have arbitrarily large stretch factors. Note that these papers analyze the largest possible stretch factor of *any* Delaunay triangulation or beta-skeleton. For example, Keil and Gutwin showed that for any finite set of points in the plane, the stretch factor of its Delaunay triangulation is bounded from above by $\frac{2\pi}{3\cos\pi/6}$. Clearly, for some sets of points, the stretch factor can be much smaller. The current paper represents the first attempt at devising algorithms to efficiently approximate the stretch factor of a given Euclidean graph.

2. Well-separated pairs. Our algorithms use the *well-separated pair decomposition* devised by Callahan and Kosaraju [7]. We briefly review this decomposition and some of its relevant properties.

DEFINITION 1. *Let $s > 0$ be a real number, and let A and B be two finite sets of points in $\mathbb{R}^d$. We say that A and B are* well separated with respect to s *if there are two disjoint d-dimensional balls C_A and C_B, having the same radius, such that* (i) *C_A contains all points of A,* (ii) *C_B contains all points of B, and* (iii) *the distance between C_A and C_B is at least equal to s times the radius of C_A.*

See Figure 2 for an illustration. In this paper, s will always be a constant, called the *separation constant*. The following lemma follows easily from Definition 1.

LEMMA 2. *Let A and B be two finite sets of points that are well separated with respect to s, let a and p be points of A, and let b and q be points of B. Then*

1. $|ab| \leq (1 + 4/s)|pq|$,
2. $|pa| \leq (2/s)|pq|$.

DEFINITION 3 (see [7]). *Let S be a set of n points in $\mathbb{R}^d$ and $s > 0$ a real number.* A well-separated pair decomposition *(WSPD) for S (with respect to s) is a sequence*

General Algorithm $\mathcal{A}$
The algorithm takes as input a Euclidean graph G on a set S of points in $\mathbb{R}^d$ and a real constant $\epsilon > 0$.
Step 1: Using separation constant $s = 4/\epsilon$, compute a WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}$$

for the set S.
Step 2: For each i, $1 \leq i \leq k$, compute two points a_i and b_i, where $a_i \in A_i$ and $b_i \in B_i$, such that

$$|a_i b_i|_G = \max\{|pq|_G : p \in A_i, q \in B_i\},$$

and compute $t_i := |a_i b_i|_G / |a_i b_i|$.
Step 3: Report the value of t, defined as $t := \max(t_1, t_2, \ldots, t_k)$. Also report points a_i and b_i for which $t = t_i$.

FIG. 3. *The first general algorithm for approximating the stretch factor of a Euclidean graph.*

of pairs of nonempty subsets of S,

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\},$$

such that

1. *$A_i \cap B_i = \emptyset$ for all $i = 1, 2, \ldots, k$,*
2. *for any two distinct points p and q of S, there is exactly one pair $\{A_i, B_i\}$ in the sequence, such that*
   (a) *$p \in A_i$ and $q \in B_i$, or*
   (b) *$p \in B_i$ and $q \in A_i$,*
3. *A_i and B_i are well separated with respect to s for all $i = 1, 2, \ldots, k$.*

The integer k is called the size *of the WSPD.*

Callahan and Kosaraju showed how such a WSPD of size $k = O(n)$ can be computed using a binary tree, called the *fair split tree*.

THEOREM 4 (see [7]). *Let S be a set of n points in $\mathbb{R}^d$ and $s > 0$ a separation constant. In $O(n \log n + \alpha_{ds} n)$ time, we can compute a WSPD for S of size at most $\alpha_{ds} n$. The constant in the Big-Oh bound does not depend on s. Moreover, for a large separation constant s, the value of α_{ds} is proportional to $2^d d^{d/2} s^d$.*

3. The first general algorithm. Let S be a set of n points in $\mathbb{R}^d$, and let G be a connected Euclidean graph having the points of S as its vertices. Recall that the stretch factor t^* of G is equal to

$$t^* = \max\left\{ \frac{|pq|_G}{|pq|} : p, q \in S, p \neq q \right\}.$$

Consider a WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}$$

for S. It follows from Lemma 2 that all Euclidean distances between a point of A_i and a point of B_i are roughly equal. Hence, if we compute for each i, $1 \leq i \leq k$, a point $a_i \in A_i$ and a point $b_i \in B_i$ whose distance $|a_i b_i|_G$ in G is maximum, then the largest value of $|a_i b_i|_G / |a_i b_i|$ should be a good approximation to the stretch factor t^* of G. This observation leads to our first general algorithm, which we denote by $\mathcal{A}$. See Figure 3. The following lemma proves the correctness of Algorithm $\mathcal{A}$.

LEMMA 5. *The value of t reported by Algorithm $\mathcal{A}$ is a $(1, 1+\epsilon)$-approximate stretch factor of G.*

Proof. Let t^* be the stretch factor of the graph G. We have to show that $t \leq t^* \leq (1+\epsilon)t$. Since t can be written as $|pq|_G/|pq|$ for some points p and q in S, $p \neq q$, it is clear that $t \leq t^*$.

To prove the second inequality, let x and y be two points of S such that $t^* = |xy|_G/|xy|$. Let i be the (unique) index such that (i) $x \in A_i$ and $y \in B_i$ or (ii) $x \in B_i$ and $y \in A_i$. Assume without loss of generality (w.l.o.g.) that (i) holds.

Consider the points $a_i \in A_i$ and $b_i \in B_i$ that were chosen in Step 2 of the algorithm. Clearly, $|xy|_G \leq |a_ib_i|_G$. By Lemma 2, we have $|a_ib_i| \leq (1+4/s)|xy| = (1+\epsilon)|xy|$. This gives

$$t^* = \frac{|xy|_G}{|xy|} \leq \frac{|a_ib_i|_G}{|xy|} \leq (1+\epsilon)\frac{|a_ib_i|_G}{|a_ib_i|} = (1+\epsilon)t_i \leq (1+\epsilon)t.$$

This completes the proof. □

4. An improved algorithm. The main problem with the general Algorithm $\mathcal{A}$ presented in the previous section is that Step 2 is hard to implement efficiently. In a preliminary version of this paper [20], we showed that Algorithm $\mathcal{A}$ can be used to compute a $(1, 1+\epsilon)$-approximation to the stretch factor of Euclidean paths, cycles, and trees in $O(n \log n)$, $O(n \log n)$, and $O(n \log^2 n)$ time, respectively. Using results from Arikati et al. [2], we were able to design an $O(n^{5/3}\mathit{polylog}(n))$-time algorithm for computing a $(2, 1+\epsilon)$-approximation to the stretch factor of planar Euclidean graphs.

In this section, we give a much simpler approximation algorithm. Recall that in Algorithm $\mathcal{A}$ we compute for each pair $\{A_i, B_i\}$ in a WSPD for S, a point $a_i \in A_i$ and a point $b_i \in B_i$ for which $|a_ib_i|_G$ is maximum, and we use $|a_ib_i|_G/|a_ib_i|$ as a candidate for the approximate stretch factor. Below, we prove that we can take *arbitrary* points $a_i \in A_i$ and $b_i \in B_i$ and use $|a_ib_i|_G/|a_ib_i|$, or an approximation to this quantity, as a candidate. Note that this is counterintuitive because the distances in the graph G between points of A_i and points of B_i can vary greatly.

Hence, the problem of approximating the stretch factor of a Euclidean graph can be reduced to the problem of making $O(n)$ (approximate) shortest path query computations. Shortest path query computations can, in general, be implemented more efficiently than the *farthest pair* (between sets of vertices) computations that were required when implementing Algorithm $\mathcal{A}$.

This improved algorithm, which we denote by $\mathcal{B}$, will be given in section 4.1 below. In section 5, we show that Algorithm $\mathcal{B}$ achieves, in subquadratic time, comparable approximation ratios for various classes of Euclidean graphs, as compared to Algorithm $\mathcal{A}$.

4.1. The reduction. Let p and q be two distinct vertices of a connected Euclidean graph G, and let $c \geq 1$ be a real number. We say that the real number $L(p, q)$ is a *c-approximation* to the length of a shortest path in G between p and q if

$$|pq|_G \leq L(p, q) \leq c \cdot |pq|_G.$$

Let $\mathcal{G}$ be a class of connected Euclidean graphs. We assume that we are given an algorithm ASP_c that takes as input (i) any graph G from the class $\mathcal{G}$ and (ii) any sequence of pairs of vertices of G; the algorithm ASP_c computes a c-approximation to $|ab|_G$ for each pair (a, b) in this sequence.

Algorithm $\mathcal{B}$ is given in Figure 4. The following theorem bounds the performance ratio of the output of Algorithm $\mathcal{B}$.

Improved Algorithm $\mathcal{B}$
The algorithm takes as input a Euclidean graph G from the class $\mathcal{G}$, on a set S of points in $\mathbb{R}^d$, and a real constant $\epsilon > 0$.
Step 1: Using separation constant $s = 4(1+\epsilon)/\epsilon$, compute a WSPD

$$\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}$$

for S. For each i, $1 \le i \le k$, take an arbitrary point $a_i \in A_i$, and an arbitrary point $b_i \in B_i$.
Step 2: Use algorithm ASP_c to compute, for each i, $1 \le i \le k$, a c-approximation $L(a_i, b_i)$ to the length $|a_i b_i|_G$ of a shortest path in G between a_i and b_i. For each i, $1 \le i \le k$, let

$$t_i := \frac{L(a_i, b_i)}{|a_i b_i|}.$$

Step 3: Report the value of t, defined as $t := \max(t_1, t_2, \ldots, t_k)$. Also report points a_i and b_i for which $t = t_i$.

FIG. 4. *The improved algorithm for approximating the stretch factor of a Euclidean graph.*

THEOREM 6. *Let G be a Euclidean graph from the class $\mathcal{G}$ on a set S of points in $\mathbb{R}^d$, let t^* be the stretch factor of G, and let t be the value that is reported by Algorithm $\mathcal{B}$. Then*

$$\frac{1}{c}\, t \le t^* \le (1+\epsilon)^2 t,$$

i.e., t is a $(c, (1+\epsilon)^2)$-approximate stretch factor of G.

Proof. Let i be the index such that $t = t_i = L(a_i, b_i)/|a_i b_i|$. Then

$$t \le c\, \frac{|a_i b_i|_G}{|a_i b_i|} \le c t^*.$$

In the rest of the proof, we will prove the second inequality. To be more precise, we will show that for all points $p, q \in S$, $p \ne q$,

$$\frac{|pq|_G}{|pq|} \le (1+\epsilon)^2 t.$$

This will prove that $t^* \le (1+\epsilon)^2 t$.

The proof is by induction on the rank of the distance $|pq|$ in the sorted sequence of $\binom{n}{2}$ distances in S. To start the induction, assume that p, q is a closest pair in S. Let i be the index such that (i) $p \in A_i$ and $q \in B_i$ or (ii) $p \in B_i$ and $q \in A_i$. Assume w.l.o.g. that (i) holds. Since $s > 2$, it follows from Lemma 2 that $A_i = \{p\}$ and $B_i = \{q\}$. Hence, in Step 2 of the algorithm, we have computed the value $t_i = L(p, q)/|pq|$. It follows that

$$\frac{|pq|_G}{|pq|} \le \frac{L(p, q)}{|pq|} = t_i \le t < (1+\epsilon)^2 t.$$

Now assume that p, q is not a closest pair in S and, moreover, assume that $|xy|_G/|xy| \le (1+\epsilon)^2 t$ for all pairs x, y of points of S such that $x \ne y$ and $|xy| < |pq|$. Let i be the index such that (i) $p \in A_i$ and $q \in B_i$ or (ii) $p \in B_i$ and $q \in A_i$. Again, assume w.l.o.g. that (i) holds. Consider the point $a_i \in A_i$ and the point $b_i \in B_i$ that were chosen in Step 2 of the algorithm. By the triangle inequality, we have

$$|pq|_G \le |pa_i|_G + |a_i b_i|_G + |b_i q|_G. \tag{4.1}$$

We distinguish three cases, depending on whether $|pa_i|_G$ and $|b_iq|_G$ are smaller or larger than $(\epsilon/2)|a_ib_i|_G$, respectively.

Case 1. $|pa_i|_G > (\epsilon/2)|a_ib_i|_G$ and $|b_iq|_G > (\epsilon/2)|a_ib_i|_G$.

First note that $p \neq a_i$, because $|pa_i|_G > 0$. We may assume w.l.o.g. that $|b_iq|_G \leq |pa_i|_G$. It follows from (4.1) that

$$|pq|_G < \frac{2(1+\epsilon)}{\epsilon}\,|pa_i|_G.$$

Since $s = 4(1+\epsilon)/\epsilon$, Lemma 2 implies that $|pa_i| \leq \frac{\epsilon}{2(1+\epsilon)}|pq|$. Therefore,

$$\frac{|pq|_G}{|pq|} < \frac{2(1+\epsilon)}{\epsilon}\,\frac{|pa_i|_G}{|pq|} \leq \frac{|pa_i|_G}{|pa_i|}.$$

Since $|pa_i| < |pq|$, the induction hypothesis implies that

$$\frac{|pq|_G}{|pq|} < \frac{|pa_i|_G}{|pa_i|} \leq (1+\epsilon)^2 t.$$

Case 2. $|pa_i|_G \leq (\epsilon/2)|a_ib_i|_G$.

In this case, (4.1) implies that

$$|pq|_G \leq (1+\epsilon/2)|a_ib_i|_G + |b_iq|_G.$$

Moreover, Lemma 2 implies that $|b_iq| \leq \frac{\epsilon}{2(1+\epsilon)}|pq|$ and $|a_ib_i| < (1+\epsilon)|pq|$.

First assume that $b_i \neq q$. Then

$$\begin{aligned}\frac{|pq|_G}{|pq|} &\leq (1+\epsilon/2)\,\frac{|a_ib_i|_G}{|pq|} + \frac{|b_iq|_G}{|pq|}\\ &< (1+\epsilon/2)(1+\epsilon)\,\frac{|a_ib_i|_G}{|a_ib_i|} + \frac{\epsilon}{2(1+\epsilon)}\,\frac{|b_iq|_G}{|b_iq|}\\ &\leq (1+\epsilon/2)(1+\epsilon)\,\frac{L(a_i,b_i)}{|a_ib_i|} + \frac{\epsilon}{2(1+\epsilon)}\,\frac{|b_iq|_G}{|b_iq|}.\end{aligned}$$

Since $|b_iq| < |pq|$, the induction hypothesis implies that $|b_iq|_G/|b_iq| \leq (1+\epsilon)^2 t$. Hence,

$$\frac{|pq|_G}{|pq|} < (1+\epsilon/2)(1+\epsilon)\,t_i + \frac{\epsilon(1+\epsilon)}{2}\,t \leq (1+\epsilon)^2 t,$$

where the last inequality follows from the fact that $t_i \leq t$.

If $b_i = q$, then basically the same calculation shows that

$$\frac{|pq|_G}{|pq|} < (1+\epsilon/2)(1+\epsilon)\,t_i < (1+\epsilon)^2 t.$$

Case 3. $|b_iq|_G \leq (\epsilon/2)|a_ib_i|_G$.

This case is symmetric to Case 2. □

In the following theorem, we summarize the result of this section. We denote by $T(n,m,k)$ the worst running time of algorithm ASP_c, when given (i) a graph $G \in \mathcal{G}$ having n vertices and m edges and (ii) a sequence of c-approximate shortest path queries, consisting of k pairs of vertices of G.

THEOREM 7. *Let S be a set of n points in $\mathbb{R}^d$, let G be a Euclidean graph from the class $\mathcal{G}$, having the points of S as its vertices, and let ϵ be a real constant such that $0 < \epsilon \leq 3$. We can compute a $(c, 1+\epsilon)$-approximate stretch factor of G in time*

$$O(n \log n) + T(n, m, \beta_{d\epsilon} n).$$

Here, $\beta_{d\epsilon}$ is a constant which is proportional to $24^d d^{d/2} (1/\epsilon)^d$ if $\epsilon \downarrow 0$.

Proof. Run Algorithm $\mathcal{B}$ with ϵ replaced by $\epsilon/3$. Let t be the value that is computed by this algorithm. By Theorem 6, we have $t \leq ct^*$ and

$$t^* \leq (1 + \epsilon/3)^2 t \leq (1+\epsilon)t.$$

The bound on the running time follows immediately from the algorithm and Theorem 4. □

5. Applications of Algorithm $\mathcal{B}$.

5.1. A lower bound. Before we start with the applications, we prove a lower bound for approximating the stretch factor in the algebraic computation tree model. (See Ben-Or [5] or Preparata and Shamos [22] for a description of this model.)

THEOREM 8. *Any algebraic computation tree algorithm that takes as input* (i) *a Euclidean path or cycle on a set of n points in $\mathbb{R}^d$ and* (ii) *real numbers $c_1 \geq 1$ and $c_2 \geq 1$, and that computes a (c_1, c_2)-approximate stretch factor of this graph, has worst-case running time $\Omega(n \log n)$.*

Proof. We give the lower bound proof for the case when the graph is a path. The lower bound proof for the cycle is similar.

Let $\mathcal{C}$ be any algorithm that satisfies the hypothesis. We will show that $\mathcal{C}$ can be used to solve the *element-uniqueness problem*, which is known to have an $\Omega(n \log n)$ lower bound in the algebraic computation tree model. (See [5, 22].)

Let $x_1, x_2, \ldots, x_n$ be a sequence of n real numbers. We consider these numbers as points on the x_1-axis in $\mathbb{R}^d$. Let M be the maximal element in the input sequence. Define the path P by

$$P := (x_1, M+1, x_2, M+2, x_3, M+3, \ldots, x_{n-1}, M+n-1, x_n).$$

Note that each edge of P has a nonzero length. We choose arbitrary real numbers $c_1 \geq 1$ and $c_2 \geq 1$ and run Algorithm $\mathcal{C}$ on the path P. Let t be the (c_1, c_2)-approximate stretch factor of P that is computed. Then it is easy to see that t is finite if and only if the input numbers $x_1, x_2, \ldots, x_n$ are pairwise distinct.

Since the reduction takes $O(n)$ time, it follows that Algorithm $\mathcal{C}$ has a worst-case running time of $\Omega(n \log n)$. □

5.2. Paths, cycles, and trees. Let $\mathcal{G}$ be the class of Euclidean paths, cycles, or trees. For any graph G in this class, we can, after an $O(n)$-time preprocessing, answer exact shortest path queries in $O(1)$ time, if G is a path or cycle, and in $O(\log n)$ time, if G is a tree. (If we allow nonalgebraic operations, then we can even answer shortest path queries in a tree in $O(1)$ time; see [16].) Hence, we can apply Theorem 7, with $c = 1$ and $T(n, m, k) = O(n + k)$, if G is a path or cycle, and $T(n, m, k) = O(n + k \log n)$, if G is a tree, and get the following result.

THEOREM 9. *Let S be a set of n points in $\mathbb{R}^d$; let G be a Euclidean path, cycle, or tree, having the points of S as its vertices; and let ϵ be a real constant such that $0 < \epsilon \leq 3$. In $O(n \log n)$ time, we can compute a $(1, 1+\epsilon)$-approximate stretch factor of G.*

It follows from Theorem 8 that the above result is optimal in the algebraic computation tree model.

5.3. Planar graphs. For the next application, let $\mathcal{G}$ be the class of planar connected Euclidean graphs. Let G be a graph in this class, on a set of n points in $\mathbb{R}^d$. Arikati et al. [2] have shown that we can build a data structure, in $O(n\sqrt{n}\,)$ time, that allows us to solve exact shortest path queries in $O(\sqrt{n}\,)$ time per query. Hence, we can apply Theorem 7 with $c = 1$ and $T(n, m, k) = O(n\sqrt{n} + k\sqrt{n}\,)$. This gives the following theorem.

THEOREM 10. *Let S be a set of n points in $\mathbb{R}^d$, let G be a planar connected Euclidean graph having the points of S as its vertices, and let ϵ be a real constant such that $0 < \epsilon \leq 3$. In $O(n\sqrt{n}\,)$ time, we can compute a $(1, 1+\epsilon)$-approximate stretch factor of G.*

5.4. General graphs. In our final application, we let $\mathcal{G}$ be the general class of connected Euclidean graphs. Let $G \in \mathcal{G}$ be any graph with n vertices and m edges. Note that $m \geq n - 1$. Cohen [9] has shown that for any integer $\beta \geq 1$ and any constant ϵ such that $0 < \epsilon \leq 1/2$, any sequence of $(2\beta(1+\epsilon))$-approximate shortest path queries can be answered in *expected* time

$$O((m+k)n^{1/\beta}\beta \log^2 n),$$

where k is the number of queries. Applying Theorem 7 gives the following result.

THEOREM 11. *Let S be a set of n points in $\mathbb{R}^d$, let G be a connected Euclidean graph having the points of S as its vertices and having m edges, let $\beta \geq 1$ be an integer constant, and let ϵ be a real constant such that $0 < \epsilon \leq 1/2$. In*

$$O(mn^{1/\beta} \log^2 n)$$

expected time, we can compute a $(2\beta(1+\epsilon), 1+\epsilon)$-approximate stretch factor of G.

By choosing different values for the integer constant β, Theorem 11 gives an interesting trade-off between the running time and the approximation factor. For example, by choosing β large enough, the running time in Theorem 11 is almost linear in m, but then the approximation of the stretch factor is very weak (although it is still bounded by a constant).

Theorem 11 implies the following result for *sparse* graphs, i.e., graphs having $O(n)$ edges.

COROLLARY 12. *Let S be a set of n points in $\mathbb{R}^d$, let G be a sparse connected Euclidean graph having the points of S as its vertices, let $\beta \geq 1$ be an integer constant, and let ϵ be a real constant such that $0 < \epsilon \leq 1/2$. In*

$$O(n^{1+1/\beta} \log^2 n)$$

expected time, we can compute a $(2\beta(1+\epsilon), 1+\epsilon)$-approximate stretch factor of G.

6. Conclusions. In this paper we showed how to efficiently compute a close approximation to the stretch factors of Euclidean graphs. We showed that the problem can be reduced either to a sequence of farthest pair queries on $O(n)$ pairs of sets of points or to a sequence of (approximate) shortest path queries for $O(n)$ specific pairs of points. It would be interesting to know whether $o(n)$ shortest path queries are sufficient for determining stretch factors approximately.

Except for trivial classes such as complete graphs, it is not known how to compute the exact stretch factor of *any* class of Euclidean graphs in less time than that required to solve the all-pairs-shortest-path problem. Hence, it is not evenknown if

the exact stretch factor of simple Euclidean graphs, such as paths, can be computed in subquadratic time.

Stretch factors can be thought of as a quantitative measure to compare distances in two different metrics. In this paper, we demonstrated techniques to compute stretch factors in order to compare a "graph metric" with the Euclidean metric. It would be interesting to study stretch factors as a measure to compare two non-Euclidean metrics. Our techniques cannot be used then, since no equivalent of the well-separated pair decomposition is known for non-Euclidean metrics.

REFERENCES

[1] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares, *On sparse spanners of weighted graphs*, Discrete Comput. Geom., 9 (1993), pp. 81–100.

[2] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. Smid, and C. D. Zaroliagis, *Planar spanners and approximate shortest path queries among obstacles in the plane*, in Algorithms—ESA '96, Fourth Annual European Symposium, Lecture Notes in Comput. Sci. 1136, Springer-Verlag, New York, 1996, pp. 514–528.

[3] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn, *A polynomial-time approximation scheme for weighted planar graph TSP*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, 1998, pp. 33–41.

[4] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid, *Euclidean spanners: Short, thin, and lanky*, in Proceedings of the 27th Annual ACM Symposium on the Theory of Computing, San Francisco, SIAM, Philadelphia, 1995, pp. 489–498.

[5] M. Ben-Or, *Lower bounds for algebraic computation trees*, in Proceedings of the 15th Annual ACM Symposium on the Theory of Computing, 1983, pp. 80–86.

[6] P. B. Callahan and S. R. Kosaraju, *Faster algorithms for some geometric graph problems in higher dimensions*, in Proceedings of the 4th Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, SIAM, Philadelphia, 1993, pp. 291–300.

[7] P. B. Callahan and S. R. Kosaraju, *A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields*, J. ACM, 42 (1995), pp. 67–90.

[8] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New sparseness results on graph spanners*, Internat. J. Comput. Geom. Appl., 5 (1995), pp. 125–144.

[9] E. Cohen, *Fast algorithms for constructing t-spanners and paths with stretch t*, SIAM J. Comput., 28 (1998), pp. 210–236.

[10] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[11] G. Das and G. Narasimhan, *A fast algorithm for constructing sparse Euclidean spanners*, Internat. J. Comput. Geom. Appl., 7 (1997), pp. 297–315.

[12] D. P. Dobkin, S. J. Friedman, and K. J. Supowit, *Delaunay graphs are almost as good as complete graphs*, Discrete Comput. Geom., 5 (1990), pp. 399–407.

[13] D. Eppstein, *Beta-Skeletons Have Unbounded Dilation*, Technical Report 96-15, Department of Information and Computer Science, University of California, Irvine, CA, 1996.

[14] D. Eppstein, *Spanning trees and spanners*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science, Amsterdam, 1999, pp. 425–461.

[15] G. N. Frederickson, *Fast algorithms for shortest paths in planar graphs, with applications*, SIAM J. Comput., 16 (1987), pp. 1004–1022.

[16] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors*, SIAM J. Comput., 13 (1984), pp. 338–355.

[17] J. M. Keil and C. A. Gutwin, *Classes of graphs which approximate the complete Euclidean graph*, Discrete Comput. Geom., 7 (1992), pp. 13–28.

[18] C. Levcopoulos, G. Narasimhan, and M. Smid, *Efficient algorithms for constructing fault-tolerant geometric spanners*, in Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, 1998, pp. 186–195.

[19] N. Linial, E. London, and Y. Rabinovich, *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15 (1995), pp. 215–245.

[20] G. Narasimhan and M. Smid, *Approximating the stretch factor of Euclidean paths, cycles and trees*, Report 9, Department of Computer Science, University of Magdeburg, Magdeburg, Germany, 1999.

[21] D. Peleg and A. Schäffer, *Graph spanners*, J. Graph Theory, 13 (1989), pp. 99–116.
[22] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, Berlin, 1988.
[23] S. Rao and W. D. Smith, *Approximating geometrical graphs via "spanners" and "banyans,"* in Proceedings of the 30th Annual ACM Symposium on the Theory of Computing, 1998, pp. 540–550.
[24] M. Smid, *Closest-point problems in computational geometry*, in Handbook of Computational Geometry, J.-R. Sack and J. Urrutia, eds., Elsevier Science, Amsterdam, 1999, pp. 877–935.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 990–1009

THE FORMULA ISOMORPHISM PROBLEM*

MANINDRA AGRAWAL† AND THOMAS THIERAUF‡

Abstract. We investigate the computational complexity of the *formula isomorphism* problem (FI): on input of two boolean formulas F and G decide whether there exists a permutation of the variables of G such that F and G become equivalent. FI is contained in $\Sigma_2\mathbf{P}$, the second level of the polynomial hierarchy.

Our main result is a one-round interactive proof for the complementary *formula nonisomorphism* problem (FNI), where the verifier has access to an **NP**-oracle. To obtain this, we use a result from learning theory by Bshouty et al. that boolean formulas can be learned probabilistically with equivalence queries and access to an **NP**-oracle. As a consequence, FI cannot be $\Sigma_2\mathbf{P}$-complete unless the polynomial hierarchy collapses.

Further properties of FI are shown: FI has and- and or-functions, the counting version, #FI, can be computed in polynomial time relative to FI, and FI is self-reducible.

Key words. boolean isomorphism problems, complexity theory, interactive proofs, learning theory

AMS subject classifications. 06E30, 68Q17, 68Q32, 94C10

PII. S0097539798343647

1. Introduction.

An interesting computational issue is to decide the equivalence of two given programs with respect to some computational model. While the problem is undecidable for computational models such as Turing machines, LOOP-programs, or context-free languages (see [Hu79]), it is **coNP**-complete for LOOP(1)-programs (no nested loops), circuits, branching programs, and boolean formulas. Moreover, it can be efficiently solved for one-time-only branching programs by a randomized algorithm [BCW80]. For deterministic finite automata it can be efficiently solved deterministically (see [Hu79]).

We consider the following generalization of the equivalence problem: determine (for some fixed model) whether there exists a *bijective transformation* of the inputs of one program that makes the two given programs equivalent. Since we consider such a transformation of the input intuitively as an *easy* operation, this captures the idea of a notion of *almost equivalence*.

From a computational perspective, the *graph isomorphism* problem might be the most studied problem of this kind. Here, the bijection is required to be an *isomorphism*, i.e., a permutation of the input vertices (see [Hof82] and [KST93] for a comprehensive discussion of this problem).

From a combinatorial perspective, another such problem has been very well studied too: the *boolean congruence* problem. Mathematicians since the last century have investigated the structure of the sets of boolean functions that are congruent to each other. Here, the *congruence* is a bijective transformation in which a permutation of the variables is composed with a *negation mapping* which maps each variable either

*Received by the editors August 19, 1998; accepted for publication (in revised form) February 5, 2000; published electronically August 24, 2000.

http://www.siam.org/journals/sicomp/30-3/34364.html

†Department of Computer Science, Indian Institute of Technology, Kanpur 208016, India (manindra@iiitk.ac.in). The research of this author was done while visiting the University of Ulm, Germany, and was supported in part by an Alexander von Humboldt fellowship.

‡Abteilung Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany (thierauf@informatik.uni-ulm.de).

to itself or to its complement. (The paper by Borchert, Ranjan, and Stephan [BRS98] gives extensive background and provides a list of early references on this problem.) To motivate the name, the boolean congruence relation can be seen as a geometrical congruence: there are 2^n assignments for a boolean function f over n variables that form the nodes of an n-dimensional cube in $\mathbf{R}^n$. Those assignments, where f evaluates to one, constitute a subgraph of the cube, *the n-dimensional geometrical cube that represents f*. Two functions f and g are congruent if and only if the n-dimensional geometrical cubes that represent f and g are geometrically congruent, that is, there is a distance-preserving bijection from one subgraph to the other.

In recent years, this and similar problems have been reconsidered from the complexity perspective [BR93, BRS98, CK91]. Of course, one needs to fix a model for representing boolean functions—the most popular of these are formulas and circuits. For formulas, we can define the *formula congruence* problem (FC) as follows: given two boolean formulas over n variables, determine whether they are congruent. One can similarly define the *formula isomorphism* problem (FI) where, instead of a congruence, we ask for an isomorphism. Although congruence is a broader notion than isomorphism, FC is many-one equivalent to FI [BRS98].

It is straightforward to see that FI is **coNP**-hard but it is not known to be in **coNP**. Therefore it is at least as difficult as the boolean formula equivalence problem. As an upper bound on its complexity, it can be easily shown that FI is in the second level of the polynomial hierarchy, $\Sigma_2\mathbf{P}$. It is posed as an open problem by Borchert, Ranjan, and Stephan [BRS98] whether FI is complete for $\Sigma_2\mathbf{P}$. They conjectured that it is not. In this paper we give an affirmative answer to this question in the following sense: we show that FI is *not* complete for $\Sigma_2\mathbf{P}$ unless the polynomial hierarchy collapses.

We also give a lower bound for FI: we show that the problem to decide whether a graph has a unique optimal clique—which is not known to be in the Boolean Hierarchy—many-one reduces to it.

The FI problem shares many similarities with the graph isomorphism problem (GI). Many of the results for GI carry over to FI with similar proofs (including the noncompleteness one), although with some crucial differences.

One can also consider more general bijections than isomorphisms and congruences. We can rewrite any permutation of n variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ as a product of a permutation matrix $\boldsymbol{P}$ with $\boldsymbol{x}$ over $\mathbf{GF}(2)$. That is, an isomorphism can be written as $\boldsymbol{xP}$, and a congruence can be written as $\boldsymbol{xP} + \boldsymbol{c}$ for a vector $\boldsymbol{c} \in \{0,1\}^n$. A natural generalization of the above notions is therefore to consider linear and affine transformations $\boldsymbol{xA}$ and $\boldsymbol{xA} + \boldsymbol{c}$, respectively, where $\boldsymbol{A}$ has to be a bijection on $\{0,1\}^n$. We call two formulas *linear equivalent* or *affine equivalent* if they become equivalent after a linear or an affine transformation of the variables of one of the formulas, respectively. As in the case of isomorphism and congruence, the *formula linear equivalence* problem (FLE) and the *formula affine equivalence* problem (FAE) are many-one equivalent [BRS98]. Also, FI many-one reduces to FAE [BRS98].

Considering circuits as representation of boolean functions instead of boolean formulas, we get the corresponding problems CI, CC, CLE, and CAE. The latter is the most complex problem we considered so far: all the other problems are many-one reducible to it. The above mentioned result for FI holds in fact more generally for CAE. That is, CAE is not complete for $\Sigma_2\mathbf{P}$ unless the polynomial hierarchy collapses.

The paper is organized as follows. In section 3 we show that the formula nonisomorphism (FNI) has a one-round interactive proof, where the verifier has access to

an **NP**-oracle. From this we conclude the above mentioned noncompleteness result. In section 4 we show that FI has and- and or-functions. This will provide us with a lower bound for FI: the unique optimal clique problem (UOClique) can be many-one reduced to it. In section 5 we show that the counting version of FI can be solved in polynomial time relative to FI. This is a result that holds analogously for GI. Finally, in section 6 we show that FI is self-reducible.

2. Preliminaries. For standard notions in complexity theory we refer the reader to textbooks such as [BDG88, BDG91, Hu79]. We fix an alphabet Σ, for example, $\Sigma = \{0, 1\}$. Complexity classes we use are **P**, **NP**, or **RP**. The class of polynomial-time computable *functions* is **FP**. We also use *relativized* classes such as $\mathbf{P}^{\mathbf{NP}}$. The levels of the *polynomial hierarchy* (**PH**) are denoted by $\Sigma_k\mathbf{P}$. The *boolean hierarchy* is the closure of **NP** under boolean operations. Its kth level consists of all sets that can be expressed as the symmetric difference of at most k **NP**-sets. For any class $\boldsymbol{\mathcal{C}}$, we denote the complement class by **co** $\boldsymbol{\mathcal{C}}$.

Set A is *many-one reducible* to set B (we write $A \leq_m^p B$ for short) if there is a function $f \in \mathbf{FP}$ such that for every $x \in \Sigma^*$

$$x \in A \iff f(x) \in B.$$

We write $A \equiv_m^p B$ if both sets are many-one reducible to each other. *Truth-table reductions* generalize many-one reductions by allowing the function $f(x)$ to compute several strings of which we determine membership in B. On the outcome of these membership queries a boolean predicate is applied which decides on the membership of x in A. If the boolean predicate is simply a conjunction or disjunction, we talk of a *conjunctive* or *disjunctive truth-table reduction*, respectively.

Our main results are based on *interactive proof systems* introduced by Goldwasser, Micali, and Rackoff [GMR89] and Babai and Moran [BM88]. For completeness, we give definitions of such classes.

2.1. Complexity classes. An *interactive proof system* [GMR89] for a set L consists of a *prover* P and *verifier* V. The verifier is a randomized polynomial-time algorithm that can communicate with the prover. The prover can make arbitrary computations. After following some communication protocol, the verifier finally has to accept or reject a given input x such that

$$\begin{aligned} x \in L &\implies \exists \text{ prover } P: \ \mathbf{Prob}(V, P)(x) \text{ accepts } = 1, \\ x \notin L &\implies \forall \text{ prover } P: \ \mathbf{Prob}(V, P)(x) \text{ accepts } \leq 1/2, \end{aligned}$$

where the probability is taken over the random choices of the verifier.

IP denotes the class of sets that have an interactive proof system. $\mathbf{IP}[k]$ is the subclass of **IP** where the verifier and the prover exchange at most k messages on every input.

A concept very similar to interactive proof systems was introduced by Babai and Moran [BM88] and called *Arthur–Merlin games*. Here Arthur plays the role of the verifier and Merlin that of the prover. The only difference is in the use of the random bits: the verifier has its own random tape which is secret to the prover. In contrast, Arthur and Merlin have a random tape available to both. The difference between interactive proof systems and Arthur–Merlin games is therefore sometimes pinpointed to *private vs. public coins*.

If there are at most k message exchanges between Arthur and Merlin, the language is in the class $\mathbf{AM}[k]$. In contrast to the notation in interactive proof systems, $\mathbf{AM}$ denotes the class $\mathbf{AM}[2]$.

In a series of surprising results it was shown that, for constant $k \geq 2$, all the above classes coincide [BM88, GS89], that is,

$$\mathbf{IP}[k] = \mathbf{AM}[k] \quad = \quad \mathbf{AM}.$$

We also consider relativizations of these classes. Since the prover and Merlin have unlimited computational power, they don't need an oracle. Hence $\mathbf{IP}^A$ denotes the class of interactive proof systems where the verifier has access to oracle A. Analogously, in $\mathbf{AM}^A$ Arthur has access to oracle A. We note that the above collapse result holds in presence of an oracle as well. In particular, we have $\mathbf{IP}[k]^{\mathbf{NP}} = \mathbf{AM}^{\mathbf{NP}}$.

$\mathbf{AM}$ can be expressed in terms of more standard complexity classes: for a class of sets $\boldsymbol{\mathcal{C}}$, define $\mathbf{BP} \cdot \boldsymbol{\mathcal{C}}$ as the class of sets L such that there exists a set $A \in \boldsymbol{\mathcal{C}}$ and a polynomial p such that for every x

$$\begin{aligned} x \in L &\implies \mathbf{Prob}(x, y) \in A \geq 2/3, \\ x \notin L &\implies \mathbf{Prob}(x, y) \in A \leq 1/3, \end{aligned}$$

where y is chosen uniformly at random from $\Sigma^{p(|x|)}$.

It is well known that $\mathbf{AM} = \mathbf{BP} \cdot \mathbf{NP}$, because in an $\mathbf{AM}$ protocol, *Arthur* can be replaced by a bounded error probabilistic polynomial-time machine that just passes the result of the coin tosses to *Merlin* (see [GS89]). Relative to an $\mathbf{NP}$-oracle, this equation translates to $\mathbf{AM}^{\mathbf{NP}} = \mathbf{BP} \cdot \Sigma_2\mathbf{P}$.

2.2. Isomorphism problems. The graph isomorphism problem (GI) is defined as follows. Given two graphs G_0 and G_1 with n nodes $V = \{1, \ldots, n\}$, decide whether there is a permutation φ on V such that G_1 becomes equal to G_0 when the nodes of G_1 are permuted by φ. In other words, for all nodes $i, j \in V$, we have (i, j) is an edge in G_0 if and only if $(\varphi(i), \varphi(j))$ is an edge in G_1. Clearly, GI is in $\mathbf{NP}$. The complementary *graph nonisomorphism problem* is denoted by GNI and is therefore in $\mathbf{coNP}$.

Two boolean formulas F_0 and F_1 with variables $x_1, \ldots, x_n$ are *equivalent* if they have the same value for every assignment to their variables. The equivalence problem for boolean formulas is known to be $\mathbf{coNP}$-complete.

Two formulas F_0 and F_1 are *isomorphic* if there exists a permutation φ on the set of variables $\{x_1, \ldots, x_n\}$, such that F_1 becomes equivalent to F_0 when the variables of F_1 are permuted by φ, i.e., $F_0(x_1, \ldots, x_n)$ is equivalent to $F_1 \circ \varphi = F_1(\varphi(x_1), \ldots, \varphi(x_n))$. In this case, we call φ an *isomorphism between F_0 and F_1*. The formula isomorphism problem (FI) is to decide whether two given formulas are isomorphic. It follows directly from the definition that $\mathrm{FI} \in \Sigma_2\mathbf{P}$, the second level of the polynomial hierarchy. The complementary *formula nonisomorphism problem* is denoted by FNI.

$\mathsf{Iso}(F_0, F_1)$ denotes the set of isomorphisms between F_0 and F_1. An automorphism of a formula F is an isomorphism between F and F, $\mathsf{Aut}(F) = \mathsf{Iso}(F, F)$. $\mathsf{Aut}(F)$ is a group with composition $\circ$ as group operation. It is a subgroup of the permutation group (on n variables). Every formula has a trivial automorphism: the identity mapping. The *formula automorphism problem* (FA) is to decide whether a formula F has a nontrivial automorphism, i.e., whether $|\mathsf{Aut}(F)| > 1$.

A *negation mapping* on n variables is a function ν such that $\nu(x_i) \in \{x_i, \overline{x}_i\}$ for $1 \le i \le n$. In other words, a negation mapping allows us to replace a variable by its complement. Two formulas F_0 and F_1 are *congruent* if there exists a permutation φ and a negation mapping ν on $\{x_1, \ldots, x_n\}$, such that F_0 and $F_1 \circ \nu \circ \varphi$ are equivalent. The formula congruence problem (FC) is to decide whether two formulas are congruent. Congruence is a more flexible notion than isomorphism, however; FC is clearly in $\Sigma_2\mathbf{P}$. Moreover, FI $\equiv_m^p$ FC [BRS98].

Formulas F_0 and F_1 are *affine equivalent* if there exists a nonsingular $n \times n$ matrix $\boldsymbol{A}$ and a $1 \times n$ vector $\boldsymbol{c}$ over $\mathbf{GF}(2)$ such that for every $\boldsymbol{x} = (x_1, \ldots, x_n)$, $F_0(\boldsymbol{x})$ and $F_1(\boldsymbol{x}\boldsymbol{A} + \boldsymbol{c})$ are equivalent (here addition and multiplication are over $\mathbf{GF}(2)$). The formulas are *linear equivalent* if they are affine equivalent with vector $\boldsymbol{c}$ as the zero vector. We use FAE and FLE to denote the set of pairs of formulas that are respectively affine and linear equivalent.

The above definitions can be applied to circuits instead of formulas. We use CI, CC, CLE, and CAE to denote the set of circuit pairs that are isomorphic, congruent, linear equivalent, and affine equivalent, respectively. We note that the reductions shown in [BRS98] carry over to circuits. That is, we have CI $\equiv_m^p$ CC $\le_m^p$ CLE $\equiv_m^p$ CAE. Since a formula can easily be transformed to a circuit, each boolean formula problem is many-one reducible to its corresponding circuit version. It follows that CLE and CAE are the computationally hardest problems we have defined here.

3. An interactive proof for FNI. We show that there is a one-round interactive proof for FNI where the verifier has access to an **NP**-oracle. Our interactive proof is based on the one for GNI [GMR89]. However, it differs from it in one crucial aspect. In Figure 3.1 we first recall the protocol for GNI on input of two graphs (G_0, G_1), both with nodes $V = \{1, \ldots, n\}$.

V: The verifier randomly picks $i \in \{0, 1\}$, a random permutation φ on V, and permutes G_i with φ. Let H be the graph obtained that way. The verifier sends H to the prover.
P: The prover answers by sending $j \in \{0, 1\}$ to the verifier.
V: Finally, the verifier accepts if $i = j$, and rejects otherwise.

FIG. 3.1. *Interactive protocol for* GNI.

When the input graphs are *not* isomorphic, the prover can find out from which of G_0 or G_1 the graph H was obtained by the verifier, and can therefore make the verifier accept with probability one. On the other hand, when the graphs are isomorphic, then no prover can find out the graph that was chosen by the verifier to construct H. Therefore, the answer of any prover is correct with probability at most 1/2.

Unfortunately, the analogous protocol for FNI does not work. To see this, consider the above protocol on input (F_0, F_1), where

$$F_0 = x_1 \wedge (\overline{x}_1 \vee \overline{x}_2) \quad \text{and}$$
$$F_1 = \overline{x}_1 \wedge x_2.$$

Note that F_0 and F_1 are isomorphic (exchange x_1 and x_2). The verifier randomly picks $i \in \{0, 1\}$, obtains a formula G by randomly permuting F_i, and sends it to the prover. However, even though F_0 and F_1 are isomorphic, the prover can easily detect from which one G has been obtained because of the syntactic structure of G:

any permutation of F_0 will have three literals and any permutation of F_1 will have two literals.

It seems that what we need is a *normal form* for equivalent boolean formulas that can be computed by the verifier. (Recall that the verifier has access to an **NP**-oracle.) Then the verifier could map the formula G in the above protocol to its normal form H, and then the prover could not distinguish whether H is coming from F_0 or F_1, if the formulas are isomorphic.

Clearly, we cannot simply transform G into its disjunctive or conjunctive normal form, because this might lead to formulas that are exponentially longer than G. Another obvious candidate for a normal form is the *smallest* equivalent boolean formula (under some suitable total ordering). However, it requires a $\Sigma_2\mathbf{P}$-oracle to compute this [Uma98], and our verifier only has an **NP**-oracle available.

To overcome this difficulty, we compute what can be called a *randomized normal form* of a formula. It is obtained from an algorithm in learning theory by Bshouty et al. [BCG$^+$96]. We start by providing an informal description of the learning scenario and then give a reformulation in pure complexity theoretic terms.

Let F be a boolean formula given in a *black box*. A probabilistic polynomial-time machine, the *learner*, has to compute a formula that is equivalent to F, but without seeing F. The learner can use an **NP**-oracle, and, furthermore, ask *equivalence queries* of a *teacher* who knows F. That is, the learner can send a formula G to the teacher. If F and G are equivalent, the learner has succeeded in learning F and the teacher will answer "yes." Otherwise, the teacher will send a *counterexample* to the learner, namely an assignment $\boldsymbol{a}$ such that $F(\boldsymbol{a}) \neq G(\boldsymbol{a})$. The learner *succeeds in learning* F if he outputs, with high probability, a formula that is equivalent to F. The learner might sometimes fail to learn F, but only with small probability. In this case, he makes no output. The result of Bshouty et al. [BCG$^+$96] says that boolean formulas can be learned in this setting.

How can we use this for our normal form problem? The crucial observation in the learning process is that the output of the learner does *not* depend on the specific syntactic form of the input formula F: because of the black box approach, the learner has exactly the same behavior on every formula F' given as input that is equivalent to F. *Hence, we take the output of the learner as our normal form.*

Note that this is not a normal form in the classical sense, because the learner produces possibly different equivalent formulas for different random choices. However, on each random path the output remains the same on any F' as input that is equivalent to F. This will suffice for our purposes.

We want to reformulate the result in complexity theoretic terms. Our first step is to throw out the teacher. She has to decide the equivalence of formulas and to compute counterexamples. The latter can easily be done within $\mathbf{P}^{\mathbf{NP}}$ by a standard prefix-search: extend, bit by bit, a partial assignment to the variables where the two given formulas differ. Each bit can be obtained with a query to an **NP**-oracle.

In summary, we get *one* probabilistic algorithm that, with the help of an **NP**-oracle, simulates the learner *and* the teacher. The output is, with high probability, a formula equivalent to the input formula, and, with small probability, there is no output. Moreover, the output does not depend on the syntax of the input formula. More precisely, we have the following.

THEOREM 3.1 (see [BCG$^+$96]). *There is a probabilistic polynomial-time algorithm A that has access to an* **NP***-oracle such that on input of a boolean formula F, algorithm A has the following properties.*

(1) *A outputs a boolean formula that is equivalent to F with probability at least 3/4, and makes no output otherwise.*
(2) *If F' is a formula equivalent to F, then, for any choice of the random bits used by A, it makes the same output on input F and on input F'.*

Now the idea should be clear. The verifier first transforms the randomly produced formula G into the normal form via the above algorithm, and then sends it to the prover. We give the full protocol below.

THEOREM 3.2. FNI $\in \mathbf{IP}[2]^{\mathbf{NP}}$.

Proof. The IP-protocol shown in Figure 3.2 accepts FNI. The inputs are two formulas (F_0, F_1) , both in variables $x_1, \ldots, x_n$, and without loss of generality (w.l.o.g.) of the same length.

V: The verifier randomly picks $i \in \{0,1\}$ and a random permutation φ on the n variables. Let $G = F_i \circ \varphi$. Now, the verifier uses the algorithm of Theorem 3.1 on input G to obtain an equivalent boolean formula H and sends H to the prover. On those paths where the algorithm does not generate an output, the verifier directly accepts.
P: The prover answers by sending $j \in \{0,1\}$ to the verifier.
V: Finally, the verifier accepts if $i = j$, and rejects otherwise.

FIG. 3.2. *Interactive protocol for* FNI.

We show that the above protocol works correctly. If F_0 is *not* isomorphic to F_1, a prover can determine which of F_0 and F_1 formula H is isomorphic to and tell it to the verifier. Also, on the random paths where no equivalent formula is produced, the verifier accepts. Therefore, the verifier accepts with probability one.

Now consider the case when F_0 is isomorphic to F_1. Assume the verifier picks $i = 0$ (the case $i = 1$ is analogous). Then the verifier constructs $G = F_0 \circ \varphi$ for some randomly chosen permutation φ. Since F_0 is isomorphic to F_1, G is isomorphic to F_1 too. Hence there is some permutation φ' such that formula $G' = F_1 \circ \varphi'$ is equivalent to G.

The next step of the verifier is to apply the algorithm of Theorem 3.1 to G and to obtain the equivalent formula H (or no formula). Now observe that the random bits of the verifier that lead to the construction of H on input G would also lead to the construction of H on input G'. In other word, any formula H has the same probability to be sent to the prover, irrespective of whether G was obtained from F_0 or F_1. Therefore the answer of any prover will be correct with probability 1/2.

In summary, the verifier accepts with probability 1/2 on those computations where a formula H is produced and on all computations with no output. The latter occurs with probability at most 1/4. Therefore, the verifier will accept with probability at most $1/2 + 1/4 = 3/4$.

The definition of an interactive proof system requires an error bound of at most 1/2. There is a standard trick to achieve this now: execute the above protocol three times in parallel and accept only if all three executions lead to acceptance. This doesn't change the case when F_0 is not isomorphic to F_1. In the other case, the error decreases to $(3/4)^3 < 1/2$. This proves the theorem. □

COROLLARY 3.3. FNI $\in \mathbf{BP} \cdot \Sigma_2\mathbf{P}$.

For readers familiar with cryptography we remark that the idea used to prove Theorem 3.2 can be used to give a *perfect zero-knowledge* interactive proof for FI, where the verifier has access to an **NP**-oracle.

Schöning [Sch88] gives a direct proof that the graph isomorphism problem is in **AM** by using hash functions. We remark that we can extend Schöning's proof by our technique to directly obtain Corollary 3.3.

Schöning [Sch89] showed that a $\Pi_2\mathbf{P}$-complete set cannot be in $\mathbf{BP}\cdot\Sigma_2\mathbf{P}$ unless the polynomial hierarchy collapses. Combined with Corollary 3.3, we obtain our main result.

COROLLARY 3.4. *If* FI *is* $\Sigma_2\mathbf{P}$*-complete, then* $\mathbf{PH}=\Sigma_3\mathbf{P}$.

The learning result of Bshouty et al. [BCG$^+$96] holds as well for circuits instead of formulas. Therefore we can adapt the interactive proof for FNI for the circuit nonisomorphism problem (CNI). That is, we have CNI $\in \mathbf{IP}[2]^{\mathbf{NP}}$, and consequently, we have the following.

COROLLARY 3.5. *If* CI *is* $\Sigma_2\mathbf{P}$*-complete, then* $\mathbf{PH}=\Sigma_3\mathbf{P}$.

We can extend the interactive proof for FI even further: to the *affine equivalence problem for circuits* (CAE). To do so, we slightly modify the protocol in the proof of Theorem 3.2. The only difference is the kind of transformation that is used: in Theorem 3.2, the verifier randomly generates a permutation. Now, the verifier must randomly generate an *affine transformation*. To achieve this, the verifier randomly generates an n-bit vector and an $n\times n$ 0-1 matrix. To constitute an affine transformation, the matrix should be nonsingular. Our next lemma ensures that there are enough nonsingular matrices for the verifier to find one with high probability.

LEMMA 3.6. *At least* $1/4$ *of the* $n\times n$ *matrices over* $\mathbf{GF}(2)$ *are nonsingular.*

Proof. We successively choose the column vectors of an $n\times n$ matrix such that the next column vector is linearly independent of the previous ones. The first column can be chosen arbitrarily, except that it can't be zero. So there are 2^n-1 choices.

Any k linearly independent vectors in $\mathbf{GF}(2)^n$ span a vector space of size 2^k. Therefore, when we choose the $(k+1)$st column, we have 2^n-2^k choices.

In total, $\prod_{k=0}^{n-1}(2^n-2^k)$ of the 2^{n^2} $n\times n$ matrices over $\mathbf{GF}(2)$ are nonsingular. Thus, their proportion is

$$\begin{aligned}\frac{1}{2^{n^2}}\prod_{k=0}^{n-1}(2^n-2^k) &= \prod_{k=1}^{n}\left(1-\frac{1}{2^k}\right)\\ &= \frac{1}{2}\prod_{k=2}^{n}\left(1-\frac{1}{2^k}\right) \quad (\text{for } n\geq 2)\\ &\geq \frac{1}{2}\prod_{k=2}^{n}\left(1-\frac{1}{k^2}\right) \quad (\text{for } n\geq 6)\\ &= \frac{1}{2}\left(\frac{1}{2}\ \frac{n+1}{n}\right)\\ &\geq \frac{1}{4},\end{aligned}$$

where the second line from bottom follows by induction on n. For values of n smaller than 6, the above bound holds too, as can be checked directly. □

Below we give the full protocol for the affine nonequivalence problem for circuits (CANE).

THEOREM 3.7. CANE $\in \mathbf{IP}[2]^{\mathbf{NP}}$.

Proof. The interactive proof system for CANE is shown in Figure 3.3. The inputs are two circuits (C_0, C_1) , both in variables $x_1,\ldots,x_n$, and w.l.o.g. of the same length.

V: The verifier randomly picks $i \in \{0,1\}$ and an n-bit vector $\boldsymbol{r}$. Furthermore, the verifier makes up to five trials to randomly get a nonsingular $n \times n$ 0-1 matrix.
If all trials fail, the verifier stops and accepts directly. Otherwise, let $\boldsymbol{R}$ be the (nonsingular) random matrix, and let $D = C_i(\boldsymbol{xR} + \boldsymbol{r})$.
Next, the verifier produces the randomized normal form from D according to the algorithm of Theorem 3.1, obtains the equivalent circuit E, and sends E to the prover. If the algorithm fails to produce a normal form, the verifier accepts directly.
P: The prover answers by sending $j \in \{0,1\}$ to the verifier.
V: Finally, the verifier accepts if $i = j$, and rejects otherwise.

FIG. 3.3. *Interactive protocol for* CANE.

We show that the protocol works correctly. If $(C_0, C_1) \in$ CANE, the honest prover can *always* convince the verifier.

Now consider the case where $(C_0, C_1) \notin$ CANE. It suffices to show that the probability that a specific circuit E is presented to the prover is independent of whether it was produced from C_0 or from C_1. In this case, the acceptance probability of the verifier is bounded by $1/2$ for the right answer of the prover, plus $(3/4)^5 < 1/16$ for not finding a nonsingular matrix, plus $1/4$ for not getting a normal form. This sums up to $13/16$. By executing the protocol four times in parallel, we can bring the acceptance probability down to $(13/16)^4 < 1/2$.

It remains to argue that independent of whether i was chosen to be 0 or 1, we have the same chance of getting circuit E. Let $\boldsymbol{xA} + \boldsymbol{c}$ be the affine transformation so that $C_1(\boldsymbol{xA} + \boldsymbol{c})$ is equivalent to C_0. For a random affine transformation, say $\boldsymbol{xR} + \boldsymbol{r}$, applied to C_0, we get $D_0 = C_0(\boldsymbol{xR} + \boldsymbol{r})$. The equivalent circuit via C_1 is $D_1 = C_1(\boldsymbol{xAR} + \boldsymbol{cR} + \boldsymbol{r})$. Now note that $\boldsymbol{x} \mapsto \boldsymbol{xAR} + \boldsymbol{cR} + \boldsymbol{r}$ is still a random affine transformation for fixed $\boldsymbol{A}$ and $\boldsymbol{c}$. Therefore we have the same probability to get D_0 when $i = 0$ and to get D_1 when $i = 1$. Since D_0 and D_1 are equivalent, our randomized normal form algorithm has an identical output distribution on input D_0 and on input D_1. □

COROLLARY 3.8. *If* CAE *is* $\Sigma_2\mathbf{P}$*-complete, then* $\mathbf{PH} = \Sigma_3\mathbf{P}$.

4. FI has and- and or-functions. In this section we show that the disjunctive and conjunctive truth-table degrees of FI collapse to the many-degree of FI. This is a consequence of FI having and- and or-functions. As an application we show that the formula automorphism problem and the unique optimal clique problem are many-one reducible to FI, thereby providing some lower bounds on the complexity of FI.

DEFINITION 4.1. *An* and-function *for a set A is a function* $\mathsf{and} : \Sigma^* \times \Sigma^* \mapsto \Sigma^*$ *such that for any $x, y \in \Sigma^*$, we have $x \in A$* and *$y \in A$ if and only if* $\mathsf{and}(x, y) \in A$. *similarly, an* or-function or *for A fulfills $x \in A$* or *$y \in A$ if and only if* $\mathsf{or}(x, y) \in A$.

4.1. Some technical lemmas on labellings. Before we can define the and- and or-functions for FI, we need some technical lemmas which provide us with some *marking* or *labelling mechanism* for the variables of a boolean formula such that a labelled variable is a *fix point of any automorphism* of the formula. It is not clear whether there exist such labellings that are efficiently computable. However, the following weaker labelling often suffices.

DEFINITION 4.2. *Let $F = F(x_1, \ldots, x_n)$ be a boolean formula. We call variables x_i and x_j* equivalent with respect to *F if, for any assignment $\boldsymbol{a}$ that satisfies F, we have $\boldsymbol{a}(x_i) = \boldsymbol{a}(x_j)$. By $E_F(x_i)$ we denote the set of variables of F that are*

equivalent to x_i.

Consider any automorphism $\varphi \in \mathsf{Aut}(F)$. If φ maps x_i to x_k, then φ must map all variables equivalent to x_i to variables that are equivalent to x_k, i.e., $\varphi(E_F(x_i)) \subseteq E_F(x_k)$. Furthermore, any variable x_j that is mapped by φ to a variable in $E_F(x_k)$ must belong to $E_F(x_i)$. Therefore, $\varphi(E_F(x_i)) = E_F(x_k)$. Since φ is a bijection, we conclude that $E_F(x_i)$ and $E_F(x_k)$ must be of the same size.

By using this property, we can label variable x_i by taking n new variables $z_1, \ldots, z_n$, and make them equivalent to x_i as follows. Define

$$L(x_i, z_1, \ldots, z_n) = \bigwedge_{j=1}^{n} (x_i \leftrightarrow z_j), \text{ and}$$
$$F_{[i]} = F \wedge L(x_i, z_1, \ldots, z_n).$$

The new variables z_i of $F_{[i]}$ are referred to as *labelling variables*.

Any assignment that satisfies $F_{[i]}$ must assign the same value to x_i and $z_1, \ldots, z_n$. Thus, x_i has more equivalent variables (with respect to $F_{[i]}$) than any other variable $x_k \notin E_{F_{[i]}}(x_i)$. Hence any automorphism φ of $F_{[i]}$ maps $E_{F_{[i]}}(x_i)$ onto itself. We say that φ *stabilizes* $E_{F_{[i]}}(x_i)$. Moreover, define φ' to coincide with φ except for the variables in $E_{F_{[i]}}(x_i)$, where φ' is defined to be the identity. Then the resulting permutation is still an automorphism of $F_{[i]}$, with the additional property that it *pointwise stabilizes* $E_{F_{[i]}}(x_i)$.

LEMMA 4.3. *For all* $\varphi \in \mathsf{Aut}(F_{[i]})$*:*

(1) $\varphi(E_{F_{[i]}}(x_i)) = E_{F_{[i]}}(x_i)$*;*
(2) *define* φ' *to coincide with* φ *on all variables not in* $E_{F_{[i]}}(x_i)$ *and to be the identity on* $E_{F_{[i]}}(x_i)$*. Then* $\varphi' \in \mathsf{Aut}(F_{[i]})$*.*

Now let $G = G(x_1, \ldots, x_n)$ be a second formula. We label variable x_j with the same label as x_i, namely, $L(x_j, z_1, \ldots, z_n)$, and we define $G_{[j]} = G \wedge L(x_j, z_1, \ldots, z_n)$. Then any isomorphism for $(F_{[i]}, G_{[j]})$ must map all the variables equivalent to x_j in $G_{[j]}$ to the variables equivalent to x_i in $F_{[i]}$.

COROLLARY 4.4. *For all* $\varphi \in \mathsf{Iso}(F_{[i]}, G_{[j]})$*:*

(1) $\varphi(E_{G_{[j]}}(x_j)) = E_{F_{[i]}}(x_i)$*;*
(2) *define* φ' *to coincide with* φ *on all variables not in* $E_{G_{[j]}}(x_j)$ *and to map* x_j *to* x_i*, to be the identity on the labelling variables* $z_1, \ldots, z_n$*, and an arbitrary bijection on the remaining variables of* $E_{G_{[j]}}(x_j)$*. Then* $\varphi' \in \mathsf{Iso}(F_{[i]}, G_{[j]})$*.*

It follows that when two formulas $F_{[i]}$ and $G_{[j]}$ as above are isomorphic, we know that there is an isomorphism that maps x_j to x_i and keeps the new variables from the labelling process on themselves. We will therefore omit to explicitly mention the new variables in a label and will simply write $L(x_i, n)$ when we label x_i with n variables that do not yet occur in the considered formula.

A more general task is to force automorphisms to stabilize a *set of variables*. Let $\boldsymbol{x} = (x_1, \ldots, x_n)$, $\boldsymbol{y} = (y_1, \ldots, y_m)$, $F = F(\boldsymbol{x}, \boldsymbol{y})$, and suppose we want to consider only automorphisms of F that map x- to x-variables and y- to y-variables. Clearly, we could simply extend the technique from Lemma 4.3 and label variable y_i with $L(y_i, M)$ for $i = 1, \ldots, m$ with $M = \max\{n, m\} + 1$. Note that the formula we get can increase quadratically in size with respect to F. However, later on we need to apply the construction iteratively such that the resulting formula still has polynomial size. In order to guarantee this, the size of the formula we obtain should increase only linearly in size. Thus we need a new technique here. The following works: for a new

variable s and a number M define formula $S(\boldsymbol{x}, \boldsymbol{y}, s, M)$ as follows:

$$S(\boldsymbol{x}, \boldsymbol{y}, s, M) = \left(\bigvee_{i=1}^{n} x_i \to s \right) \quad \wedge \quad \left(\bigvee_{i=1}^{m} y_i \to \overline{s} \right) \quad \wedge \quad L(s, M).$$

Let $S = S(\boldsymbol{x}, \boldsymbol{y}, s, n+m)$. S has the following property: let $\boldsymbol{a}$ be a satisfying assignment of S. If $\boldsymbol{a}$ assigns a 1 to any of the x-variables, then $\boldsymbol{a}(s) = 1$ which implies that $\boldsymbol{a}(\overline{s}) = 0$, and therefore $\boldsymbol{a}$ must assign 0 to all the y-variables. Symmetrically, if $\boldsymbol{a}$ assigns a 1 to any of the y-variables, then $\boldsymbol{a}(\overline{s}) = 1$ which implies that $\boldsymbol{a}(s) = 0$, and therefore $\boldsymbol{a}$ must assign 0 to all the x-variables.

Now consider $F \wedge S$. We claim that any automorphism of $F \wedge S$ must map x- to x-variables and y- to y-variables, unless they are equivalent.

LEMMA 4.5. *Let $F = F(\boldsymbol{x}, \boldsymbol{y})$ be a formula as above such that the all-zero assignment does not satisfy F. Let $\varphi \in \mathsf{Aut}(F \wedge S)$.*

1. *φ maps x- to x-variables and y- to y-variables except, maybe, for variables x_i and y_j which are set to zero by* every *satisfying assignment of $F \wedge S$.*
2. *Define φ' to coincide with φ on all variables that are set to 1 by at least one satisfying assignment of $F \wedge S$, and to be the identity on the remaining variables. Then $\varphi' \in \mathsf{Aut}(F \wedge S)$, and it maps x- to x-variables and y- to y-variables.*

Proof. Any automorphism φ of $F \wedge S$ must stabilize s (more precisely, $E_{F \wedge S}(s)$) because of its label. Let $\boldsymbol{a}$ be an assignment that satisfies $F \wedge S$. By assumption, $\boldsymbol{a}$ is not the all-zero assignment. So let x_i be a variable such that $\boldsymbol{a}(x_i) = 1$ (the case that $\boldsymbol{a}(y_i) = 1$ for some y_i is analogous). Since $x_i \to s$, we have that $\boldsymbol{a}(s) = 1$. Since $y_j \to \overline{s}$, we have that $\boldsymbol{a}(y_j) = 0$ for $j = 1, \ldots, m$. Therefore, φ cannot map x_i to some y_j in order of $\varphi(\boldsymbol{a})$ to satisfy $F \wedge S$. We conclude that φ must map x_i to some x_j.

It follows that whenever φ maps, say, x_j to y_k, then *every* satisfying assignment of $F \wedge S$ assigns zero to both, x_j and y_k, i.e., all such variables are in $E_F(x_j)$. If we modify φ to be the identity on $E_F(x_j)$, we still have an automorphism for $F \wedge S$. The latter shows part 2 of the lemma. □

We extend the lemma to isomorphisms. Let $G = G(\boldsymbol{x}, \boldsymbol{y})$. Then any isomorphism of $(F \wedge S, G \wedge S)$ must map x- to x-variables and y- to y-variables, unless they are equivalent.

COROLLARY 4.6. *Let F and G be formulas as above such that the all-zero assignment does not satisfy F or G. If $(F, G) \in$ FI, then there is a $\varphi \in \mathsf{Iso}(F \wedge S, G \wedge S)$ that maps all the x- to x-variables and all the y- to y-variables.*

For a last generalization step, consider again $F = F(\boldsymbol{x}, \boldsymbol{y})$, where $n = m$. Now we want to allow automorphisms of F to map x- variables to y-variables in the following way: either x-variables are only mapped to x-variables or x-variables are only mapped to y-variables. We can achieve this as follows. For new variables s and t and a number M define formula $ST(\boldsymbol{x}, \boldsymbol{y}, s, t, M)$:

$$ST(\boldsymbol{x}, \boldsymbol{y}, s, t, M) = \left(\bigvee_{i=1}^{n} x_i \to s \right) \quad \wedge \quad \left(\bigvee_{i=1}^{n} y_i \to \overline{s} \right) \quad \wedge \quad (s \leftrightarrow \overline{t})$$
$$\wedge \quad L(s, M) \quad \wedge \quad L(t, M).$$

Let $ST = ST(\boldsymbol{x}, \boldsymbol{y}, s, t, 2n)$. As above for S, a satisfying assignment of ST can assign a 1 to either any of the x-variables or any of the y-variables, but not to both. The difference to S is that now an automorphism of ST can interchange s and t because they have the same label.

Now consider $F \wedge ST$. We claim that any automorphism of $F \wedge ST$

(i) either maps x- to x-variables and y- to y-variables, or

(ii) interchanges x- and y-variables,

unless they are equivalent.

LEMMA 4.7. *Let F be a satisfiable formula as above such that the all-zero assignment does not satisfy F. For all $\varphi \in \mathsf{Aut}(F \wedge ST)$,*

(i) *either $\varphi(s) = s$, and then we have that φ maps x- to x-variables and y- to y-variables except, maybe, for variables x_i and y_j which are set to zero by* every *satisfying assignment of $F \wedge S$, or*

(ii) *$\varphi(s) = t$, and then we have that φ interchanges x- and y-variables except, maybe, for variables x_i, x_j or y_i, y_j which are set to zero by* every *satisfying assignment of $F \wedge S$.*

Furthermore, we can modify φ to an automorphism of $F \wedge ST$ that keeps x- on x-variables in case (i). *Case* (ii) *is more subtle: if we have the same number of x- and y-variables that have value zero in* every *satisfying assignment of F, then we can modify φ to interchange all x- and y-variables in case* (ii).

Proof. We have to distinguish two cases according to whether an automorphism φ maps s to s or t. In the case that $\varphi(s) = s$, we can directly use the proof of Lemma 4.5. Now let $\varphi(s) = t$, and let $\boldsymbol{a}$ be a satisfying assignment of $F \wedge ST$. Let x_i be a variable such that $\boldsymbol{a}(x_i) = 1$. Because $(x_i \to s)$ is a part of formula ST, we get that $\boldsymbol{a}(s) = 1$. Since $s \leftrightarrow \overline{t}$ is part of the formula ST, we must have $\boldsymbol{a}(t) = 0$.

Now consider formula $(F \wedge ST) \circ \varphi$, which is satisfied by the assignment $a \circ \varphi$. Since φ maps s to t, we have $\boldsymbol{a} \circ \varphi(s) = 0$, and, correspondingly, $\boldsymbol{a} \circ \varphi(t) = 1$. Therefore $a \circ \varphi$ must assign zero to all the x-variables. Hence φ must interchange x- and y-variables, except, maybe, variables that have value zero in all satisfying assignments.

Let x_i be a variable that has value zero in all satisfying assignments. Then all such variables are precisely those that are equivalent to x_i, i.e., that are in $E_F(x_i)$. Note that φ maps variables in $E_F(x_i)$ again to $E_F(x_i)$. Moreover, we can change φ arbitrarily on $E_F(x_i)$ and it still remains an automorphism for $F \wedge ST$. Therefore, if there is the same number of x-variables as y-variables in $E_F(x_i)$, we can modify φ to interchange all x- with the y-variables. □

COROLLARY 4.8. *Let F and G be formulas as above such that the all-zero assignment does not satisfy F or G. If $(F, G) \in \mathrm{FI}$, then there is a $\varphi \in \mathsf{Iso}(F \wedge ST, G \wedge ST)$ that*

(i) *either maps x- to x-variables and y- to y-variables, or*

(ii) *interchanges x- and y-variables.*

We therefore omit to explicitly mention the labelling variables in a label and simply write $L(x_i, n)$ when we label x_i with n variables that do not yet occur in the considered formula.

In the above lemmas we assumed that the all-zero assignment does not satisfy a given formula. In the following lemma we show that this *no* restriction when considering instances for FI.

LEMMA 4.9. *Let (F_0, F_1) be an instance for* FI *with n variables. Define $G_i = F_i \wedge z$, for $i = 0, 1$, for a new variable z. Formulas G_0 and G_1 have the following properties:*

(a) 0^n *is* not *a satisfying assignment of G_0 and G_1, and*

(b) $(F_0, F_1) \in \mathrm{FI} \iff (G_0, G_1) \in \mathrm{FI}$.

Proof. Property (a) is obvious. To see (b), let φ be an isomorphism for (G_0, G_1), i.e., φ permutes the variables of G_1 so that it becomes equivalent to G_0. Note that any satisfying assignment for G_0 or G_1 must set z to 1. Therefore φ must map z

to itself or to a variable equivalent to z. In the latter case, we can modify φ to be the identity on $E_{G_1}(z)$ and we still have an isomorphism for (G_0, G_1). Now, φ is an isomorphism for (F_0, F_1) as well (just ignore variable z). □

4.2. The and- and or-functions for FI. In the previous section we have established the technical tools for the construction of the and- and or-functions for FI.

THEOREM 4.10. FI *has and- and or-functions.*

Proof. Let (F_0, F_1) and (G_0, G_1) be two instances for FI. The variables of F_0 and F_1 are $x_1, \ldots, x_n$, and the variables of G_0 and G_1 are $y_1, \ldots, y_m$. By Lemma 4.9 we can assume that the all-zero assignment does not satisfy any of these formulas.

And-function. In order to construct the and-function, we simply combine the formulas by or-ing together F_0 and G_0 on one side, and F_1 and G_1 on the other side. However, we have to make sure that we don't get automorphisms that map x-variables to y-variables. For this we use formula $S = S(\boldsymbol{x}, \boldsymbol{y}, s, M)$ from above with $M = n+m$. Define

$$\mathsf{and}((F_0, F_1), (G_0, G_1)) = (C_0, C_1),$$

where formulas C_0 and C_1 are defined as follows:

$$\begin{aligned} C_0 &= (F_0 \vee G_0) \ \wedge \ S, \\ C_1 &= (F_1 \vee G_1) \ \wedge \ S. \end{aligned}$$

If $(F_0, F_1) \in$ FI and $(G_0, G_1) \in$ FI, then clearly $(C_0, C_1) \in$ FI. For the reverse direction assume that $(C_0, C_1) \in$ FI. By Corollary 4.6 there is an isomorphism φ that maps x- to x-variables and y- to y-variables, i.e., φ can be written as $\varphi = \varphi_x \cup \varphi_y \cup \varphi_S$, where φ_x is a permutation on $\{x_1, \ldots, x_n\}$, φ_y on $\{y_1, \ldots, y_n\}$, and φ_S on the extra variables from formula S.

We argue that φ_x is an isomorphism for (F_0, F_1) (an analogous argument shows that φ_y is an isomorphism for (G_0, G_1)): consider the following partial assignments.

- $\boldsymbol{a}_y$ assigns 0 to all the y-variables and
- $\boldsymbol{a}_S$ assigns 1 to variable s and the labelling variables.

Then we have $G_0(\boldsymbol{a}_y) = 0$, and $G_1(\varphi_y(\boldsymbol{a}_y)) = G_1(\boldsymbol{a}_y) = 0$. Furthermore, the second item implies that, for any assignment $\boldsymbol{a}_x$ of the x-variables, formula S evaluates to one on $\boldsymbol{a} = (\boldsymbol{a}_x, \boldsymbol{a}_y, \boldsymbol{a}_S)$ and on $\varphi(\boldsymbol{a})$. Now, since by assumption $C_0(\boldsymbol{a}) = C_1(\varphi(\boldsymbol{a}))$, we conclude that $F_0(\boldsymbol{a}_x) = F_1(\varphi_x(\boldsymbol{a}_x))$. Therefore, φ_x is an isomorphism for (F_0, F_1).

Or-function. In order to construct the or-function, we need a copy of the variables $\boldsymbol{x} = (x_1, \ldots, x_n)$ and $\boldsymbol{y} = (y_1, \ldots, y_m)$ used in the formulas F_0, F_1 and G_0, G_1, respectively. Let $\boldsymbol{u} = (u_1, \ldots, u_n)$ and $\boldsymbol{v} = (v_1, \ldots, v_m)$ be new variables. Define

$$\mathsf{or}((F_0, F_1), (G_0, G_1)) = (D_0, D_1),$$

where formulas D_0 and D_1 are defined as follows:

$$\begin{aligned} D_0 &= ((F_0(\boldsymbol{x}) \vee G_0(\boldsymbol{y})) \ \vee \ (F_1(\boldsymbol{u}) \vee G_1(\boldsymbol{v}))) \ \wedge \ R, \\ D_1 &= ((F_1(\boldsymbol{x}) \vee G_0(\boldsymbol{y})) \ \vee \ (F_0(\boldsymbol{u}) \vee G_1(\boldsymbol{v}))) \ \wedge \ R, \end{aligned}$$

where

$$R = S((\boldsymbol{x}, \boldsymbol{u}), (\boldsymbol{y}, \boldsymbol{v}), s_0, M_0) \ \wedge \ ST((\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{u}, \boldsymbol{v}), s, t, M).$$

Here we set $M_0 = n + m$ and $M = 2M_0$. (The additional brackets for the variables in formulas S and ST indicate the two groups of variables that occur in the definition of these formulas.)

The rough idea of this definition is that if there is an isomorphism for either (F_0, F_1) or (G_0, G_1), then we can construct an isomorphism for (D_0, D_1) from it: just map the nonisomorphic formulas to themselves. Formula R will ensure that we don't get more isomorphisms than the ones just described. We give more details below.

Suppose first that $(F_0, F_1) \in \mathrm{FI}$ and let φ_x be an isomorphism. Let φ_u be φ_x^{-1} but on the u-variables. That is, define

$$\varphi_u(u_i) \quad = \quad u_j, \quad \text{if } \varphi_x^{-1}(x_i) = x_j.$$

Define φ as the union of φ_x and φ_u and the identity on all the other variables of D_1. Then it is straightforward to check that φ is an isomorphism of (D_0, D_1), which is therefore in FI.

Now assume that $(G_0, G_1) \in \mathrm{FI}$ via isomorphism φ_y. Then we get an isomorphism φ for (D_0, D_1) as follows. Define

$$\begin{aligned}
\varphi(x_i) &= u_i, \\
\varphi(u_i) &= x_i, \\
\varphi(v_i) &= y_j, \quad \text{if } \varphi_y(y_i) = y_j, \\
\varphi(y_i) &= v_j, \quad \text{if } \varphi_y^{-1}(y_i) = y_j, \\
\varphi(s) &= t, \\
\varphi(t) &= s, \\
\varphi(s_0) &= s_0.
\end{aligned}$$

The remaining variables that come from the labelling process are mapped according to the variables they are equivalent to.

In summary, the isomorphisms we constructed have the following properties:

(i) either they map s and t to itself, respectively, and map x- to x-variables, u- to u-variables, y- to y-variables, and v- to v-variables,

(ii) or they interchange s with t and interchange x- with u-variables and y- with v-variables.

Conversely, if there is an isomorphism for (D_0, D_1) that fulfills property (i) or (ii), then it is easy to see that we get an isomorphism for either (F_0, F_1) or (G_0, G_1) from it, respectively. It remains to show that every isomorphism for (D_0, D_1) must satisfy one of these two properties. The only exception from this might be equivalent variables.

Assume that (D_0, D_1) are isomorphic. We consider formula R. By Corollary 4.6, its first part, $S((\boldsymbol{x}, \boldsymbol{u}), (\boldsymbol{y}, \boldsymbol{v}), s_0, M_0)$, implies that there is an isomorphism φ of (D_0, D_1) that can be written as a union of permutations $\varphi_{x,u}$ on x- and u-variables, $\varphi_{y,v}$ on y- and v-variables, $\varphi_{s,t}$ on s and t, and φ_L for the remaining variables from the labelling. Combined with its second part, $ST((\boldsymbol{x}, \boldsymbol{y}), (\boldsymbol{u}, \boldsymbol{v}), s, t, M)$, we get by Corollary 4.8 that $\varphi_{x,u}$, depending on $\varphi_{s,t}$, either maps all x-variables to x-variables or interchanges x- and u-variables. The same holds analogously for $\varphi_{y,v}$. Thus we get an isomorphism that fulfills property (i) or (ii) above. □

We remark that we can extend the functions and and or to more than two arguments: combine them in a binary-tree-like fashion with the above functions for two arguments. The size of the output of our functions and and or is linear in the size of the input formulas. Therefore we get a polynomial-size output when we combine several instances as described above.

COROLLARY 4.11. *If a set L is disjunctively or conjunctively truth-table reducible to* FI*, then $L \leq_m^p$* FI.

We give two applications of Corollary 4.11. The formula automorphism problem (FA) is disjunctively reducible to FI, because a formula $F = F(x_1, \ldots, x_n)$ is in FA if and only if for some pair $i, j \in \{1, \ldots, n\}$, $i \neq j$, we have that $(F_{[i]}, F_{[j]})$ is in FI. It follows that FA is many-one reducible to FI.

COROLLARY 4.12. FA $\leq_m^p$ FI.

What else can we reduce to FI? Clearly, FI is **coNP**-hard: by true_n we denote a fixed formula over n variables that is a tautology (e.g., $\bigwedge_{i=1}^{n}(x_i \vee \bar{x}_i)$). Then, for a given formula F with n variables, F is a tautology as well if and only if $(F, \mathsf{true}_n) \in$ FI.

Unique satisfiability (USAT) [BG82] is the set of all boolean formulas that have exactly one satisfying assignment. The function

$$F(x_1, \ldots, x_n) \mapsto (F \wedge z) \ \vee \ \left(\bigwedge_{i=1}^{n} x_i \wedge \overline{z} \right)$$

reduces unsatisfiable formulas to uniquely satisfiable ones. Therefore USAT is **coNP**-hard. Since USAT can be written as the difference of two **NP**-sets, USAT is in **DP**, the second level of the boolean hierarchy. On the other hand, USAT is not known to be **NP**-hard. [1]

The function

$$F(x_1, \ldots, x_n) \mapsto \left(F, \bigwedge_{i=1}^{n} x_i \right)$$

reduces USAT to FC. Since FI $\equiv_m^p$ FC [BRS98], USAT can be reduced to FI.

A problem seemingly harder than USAT is the unique optimal clique problem (UOCLIQUE), that is, whether the largest clique of a given graph is unique. The standard reduction from SAT to CLIQUE also reduces USAT to UOCLIQUE. An upper bound for the complexity of UOCLIQUE is $\mathbf{P}^{\mathbf{NP}[\log]}$: with logarithmically many queries to an **NP**-oracle one can compute the size of the largest clique of a given graph. Then, with one more query, one can find out whether there is more than one clique of that size. Papadimitriou and Zachos [PZ83] asked whether UOCLIQUE is complete for $\mathbf{P}^{\mathbf{NP}[\log]}$. This is still an open problem. Buhrman and Thierauf [BT96] provide strong evidence that UOCLIQUE is *not* complete for $\mathbf{P}^{\mathbf{NP}[\log]}$.

UOCLIQUE can be disjunctively reduced to USAT [BT96]. To see this, let UCLIQUE be the unique CLIQUE version. That is, given a graph G and a integer k, decide whether G has a unique clique of size k. Now, observe that $G \in$ UOCLIQUE $\iff \exists k : (G, k) \in$ UCLIQUE. Recall that the generic reduction of an arbitrary **NP**-problem to SAT given by Cook is parsimonious. So in particular, a graph with a unique clique of a certain size reduces to a formula with precisely one satisfying assignment. Therefore we get a disjunctive reduction of UOCLIQUE to USAT. Combined with the reduction from USAT to FI, we have that UOCLIQUE can be disjunctively reduced to FI. By Corollary 4.11, this can be turned into a many-one reduction.

COROLLARY 4.13. UOCLIQUE $\leq_m^p$ FI.

As is the case for UOCLIQUE, it is not known whether FI is **NP**-hard. This is a challenging open question.

[1] However, USAT is **NP**-hard under *randomized* reductions [VV86].

5. The counting version of FI. Mathon [Mat79] showed that the counting version of GI can be (truth-table) reduced to the decision version. Thus GI behaves differently than the known **NP**-complete problems. This was historically the first hint that GI might *not* be **NP**-complete.

We will show an analogous result for FI. The proof follows essentially the same outline as Mathon's proof for GI, however, there is again a technical difficulty to get around. First of all, it is enough to compute the number of *automorphisms* of a graph G: if H is a graph isomorphic to G, the number of isomorphisms between G and H is the same as the number of automorphisms of G (or H). We sketch the algorithm to compute the number of automorphisms of G. See [Mat79] for more details.

> In the beginning, label all nodes of G (with pairwise different labels), so that the identity is the only automorphism of the resulting graph. Then successively take away the labels. If i is the node where the label was cancelled last, compute the orbit of i by asking queries of GI. When all labels are taken away, the number of automorphisms of G is the product of the orbit sizes constructed during this procedure.

In the above algorithm one needs to construct a graph $G_{[I]}$, where $I \subseteq \{1, \ldots, n\}$, such that any automorphism of $G_{[I]}$ *pointwise* stabilizes the nodes in I. Correspondingly, given a formula F in n variables, we need to construct a formula $F_{[I]}$ whose set of automorphisms corresponds to the *pointwise stabilizer of* I in $\mathsf{Aut}(F)$. More precisely, $F_{[I]}$ must retain exactly those automorphisms of F that map variables in $E_F(x_i)$ to themselves for all $i \in I$.

Our formula S from section 4 is of no help for this: though it can stabilize a set of variables, this will not be pointwise. Also, we cannot directly use the labels L defined in section 4 as they take m new variables to label a variable in a formula with m variables. Thus, starting with n variables, we would get $n(2^{|I|} - 1)$ new variables to carry out the marking which is exponential in n when $|I| = \Theta(n)$. This is clearly too much in general.

Here, we give a method to compute $F_{[I]}$ that works in $\mathbf{FP}^{\mathbf{NP}}$. Recall that FI is **coNP**-hard. Therefore we can in particular use this method when we have FI as an oracle.

LEMMA 5.1. *$F_{[I]}$ is computable in $\mathbf{FP}^{\mathbf{NP}}$.*

Proof. Recall from section 4 that for any automorphism φ of F, we have $|E_F(x_i)| = |E_F(\varphi(x_i))|$ for any variable x_i. The trick in Lemma 4.3 was to make $|E_F(x_i)|$ unique by appending several variables that are equivalent to x_i. Now, with an **NP**-oracle, we can actually compute the sets $E_F(x_i)$: just ask the **NP**-oracle whether F is equivalent with $F \wedge (x_i \leftrightarrow x_j)$ for all j. Then, if we want to label x_i, we take the smallest label such that x_i gets a unique number of equivalent variables. This will keep the number of new variables needed small.

We construct formula $F_{[I]}$ by successively labelling the variables in I. Let $F_0 = F$. Suppose that we have already labelled the first k variables of I and obtained the formula F_k for some $k \geq 0$. Say that x_i is the next variable to label for some $i \in I$. That is, x_i is not equivalent to any of the variables already labelled. Now, let t be the smallest number such that, with respect to F_k, we have

$$|E_{F_k}(x_i)| + t \neq |E_{F_k}(x_j)| \quad \text{for any} \quad x_j \notin E_{F_k}(x_i),$$

and define $F_{k+1} = F_k \wedge L(x_i, t)$. This ensures that any automorphism of the new formula F_{k+1} stabilizes the set $E_{F_k}(x_i)$. Thus F_{k+1} has the desired property.

The above process is carried out at most $|I| \leq n$ times. Observe also that the number t is bounded by n in each iteration. Therefore, we introduce in total at most

n^2 new variables. It follows that $F_{[I]}$ has polynomial length and can be constructed in polynomial time with the help of an **NP**-oracle. □

Now we can compute the number of isomorphisms between two boolean formulas in polynomial time relative to FI: simply copy the proof described above, which was used by Mathon [Mat79] to prove the analogous result for graph isomorphism, but with the labelling technique from Lemma 5.1.

THEOREM 5.2. $\#\mathrm{FI} \in \mathbf{FP}^{\mathrm{FI}}$.

We remark that the oracle queries in the algorithm to compute formula $F_{[I]}$ do not depend on each other; they can be made in parallel.

Consequently, #FI can be computed efficiently with parallel queries to FI.

Mathon's algorithm for computing the number of isomorphisms of two graphs is an inductive process, where at each intermediate stage one knows the number of isomorphisms of the labelled graphs that are considered. This observation led to the result that $\mathrm{GI} \in \mathbf{LWPP}$ [KST93] (see [FFK94] for definitions of **PP** and **LWPP**). Since **LWPP** is low for **PP**, that is, $\mathbf{PP}^{\mathbf{LWPP}} = \mathbf{PP}$ [FFK94], it follows that GI is low for **PP**.

Using Lemma 5.1, an analogous argument shows that $\mathrm{FI} \in \mathbf{LWPP}^{\mathbf{NP}}$. Since **LWPP** is low for **PP** *relative to any oracle*, i.e., $\mathbf{PP}^{\mathbf{LWPP}^{\mathbf{NP}}} = \mathbf{PP}^{\mathbf{NP}}$ [FFK94], we get the following result for FI.

THEOREM 5.3. $\mathbf{PP}^{\mathrm{FI}} = \mathbf{PP}^{\mathbf{NP}}$.

6. FI is self-reducible. A set A is self-reducible if, very informally, the decision problem whether a given instance x is in A can be reduced to *smaller* instances for A (under some order). Self-reducibility is a very useful property of a set. For example, if an **NP**-set is self-reducible, then the (seemingly more complex) *construction problem*, i.e., constructing a witness, can be reduced to the decision problem.

In this section we show that FI is self-reducible. We start by giving formal definitions of self-reducibility and the underlying partial order.

DEFINITION 6.1. *A partial order $\prec$ on Σ^* is* polynomially related *if $\prec$ is decidable in polynomial time and there is a polynomial p such that*

(i) *for any $x, y \in \Sigma^*$, we have $x \prec y \implies |x| \le p(|y|)$, and*

(ii) *any chain is polynomially length bounded: if $x_1 \prec x_2 \prec \cdots \prec x_k$, then $k \le p(|x_k|)$.*

DEFINITION 6.2. *A set A is* self-reducible *if there is a polynomially related partial order $\prec$ and a deterministic polynomial-time Turing machine M such that*

(i) *M^A accepts A, and*

(ii) *on input x and any oracle B, M^B queries only strings y such that $y \prec x$.*

The idea to show that FI is self-reducible is pretty simple: for two formulas F and G, we have that

$$(F, G) \in \mathrm{FI} \iff \exists i, j : \ (F_{[i]}, G_{[j]}) \in \mathrm{FI}.$$

Thus the self-reduction constructs the formulas $F_{[i]}$ and $G_{[j]}$ for all values of i and j and queries the oracle. However, we need to define a polynomially related partial order according to these queries. There are some subtle points that one has to take care of.

- For labelling the variables we can use the scheme from Lemma 5.1, which yields polynomially-sized formulas, even after several labellings. To ensure *polynomially-sized chains*, the machine must check if some variables in F and G are already labelled and, in this case, not relabel them. This is possible

since a label is easily detectable: it is of the form $x \leftrightarrow y$. We use the variable with the smallest index as a representative for a label and call it a *basic variable* of F. Note that the variables that don't have a label are also basic variables. By $\#\mathsf{basic}(F)$ we denote the *number of basic variables of* F. The other variables we refer to as *labelling variables*. The *label size* of a basic variable is the number of labelling variables that label it.

- When all the basic variables of F and G have a unique label size, they define a permutation on the variables. Then we have to check whether the permuted formula F is equivalent to G. This we accomplish via the standard self-reduction for the equivalence problem: set the first variable of both formulas to 0 and 1, respectively, and then verify that both pairs of the resulting formulas are equivalent. (Note that all tests can be done with FI as an oracle.)

  Hence our partial order has to respect the order of this latter self-reduction as well.

THEOREM 6.3. FI *is self-reducible.*

Proof. We start by defining the underlying polynomially related partial order $\prec$. By true_n we denote a fixed formula over n variables that is a tautology (e.g., $\bigwedge_{i=1}^{n}(x_i \vee \bar{x}_i)$). In the following, we drop the subscript n and simply write true to denote such a formula when the number of variables is clear from the context.

For formulas F, G, F', G', we define $(F, G) \prec (F', G')$ if the length of F is bounded by a fixed polynomial in the length of F'; the same holds for G and G', and any one of the following three conditions hold.

(i) $\#\mathsf{basic}(F) < \#\mathsf{basic}(F')$, and either $G = \mathsf{true}$ or $\#\mathsf{basic}(G) \leq \#\mathsf{basic}(G')$.
(ii) $\#\mathsf{basic}(G) < \#\mathsf{basic}(G')$, and either $F = \mathsf{true}$ or $\#\mathsf{basic}(F) \leq \#\mathsf{basic}(F')$.
(iii) $\#\mathsf{basic}(F) = \#\mathsf{basic}(F')$, $\#\mathsf{basic}(G) = \#\mathsf{basic}(G')$, and more basic variables in F and G have unique label sizes than in F' and G', respectively.

This finishes the description of the partial order. Note that it is polynomially related in the sense of definition 6.1.

We now describe the self-reducing machine, M, in detail. In particular observe that the queries of M will respect the partial order defined above. At several places, M has to test whether a boolean formula T is a tautology. This is done by checking that $(T[0], \mathsf{true})$ and $(T[1], \mathsf{true})$ belong to FI, where by $T[b]$, $b \in \{0, 1\}$, we denote the formula obtained from T by setting its first basic variable—and the labelled variables associated with it—to the value b.

Let F and G be the input formulas over variables $x_1, \ldots, x_n$. M first checks if either of F and G equals true.

- If both of them do, then they are isomorphic and so M accepts.
- If exactly one of them, say G, is equal to true, then F and G are isomorphic if F is a tautology too. This can be checked using the scheme described above. M accepts if and only if F is a tautology.
- If neither F nor G equals true, then M does the following. It computes the basic variables of F and G, and finds out if any two basic variables of F are equivalent by checking if the formula $T_{F,i,j}$ is a tautology, where

$$T_{F,i,j} = (F \wedge (x_i \leftrightarrow x_j)) \leftrightarrow F$$

  for every pair of basic variables x_i and x_j of F. If $T_{F,i,j}$ is a tautology, then x_i and x_j are equivalent in F; otherwise, they are not.
  - If there are x_i and x_j in F that are equivalent, then M accepts if and only if $(F', G) \in$ FI, where F' is obtained from F by replacing all occurrences

of x_j in F by x_i and and-ing the formula $x_i \leftrightarrow x_j$ to the resulting formula. By this transformation, F and F' are equivalent and F' has one less basic variable than F. If F has no equivalent basic variables, then this is repeated for G instead of F.

– If no two basic variables of F or G are equivalent, M computes for each basic variable its label size. It then checks whether these numbers of F and G match, i.e., when the list of label sizes for F and G are sorted, they must coincide. If not, then M rejects as there cannot be any isomorphism between F and G.
So assume that these numbers match.
  * If all basic variables of F are *uniquely* labelled, then M constructs a permutation φ of variables of F such that $\varphi(x_i) = x_j$, where the label size of x_i in F and x_j in G are the same. φ also maps labelled variables associated with x_i to those associated with x_j. Now, M permutes the variables of F using φ to obtain the formula $F \circ \varphi$, then checks whether $F \circ \varphi \leftrightarrow G$ is a tautology, and accepts in this case.
  * Finally, if there are some basic variables of F with identical label sizes, for every such variable x_i of F, and for every basic variable x_j of G that has the same label size, M queries the oracle whether $(F_{[i]}, G_{[j]}) \in \mathrm{FI}$. It accepts if and only if at least one of these pairs belong to FI.

It is straightforward to see that M respects the partial order defined above, works in polynomial time, and that M^{FI} accepts FI. □

7. Open problems and related work. The hardest **NP**-problem of which we know that reduces to FI is GI. We ask:

- Is FI **NP**-hard?

Let FN be the problem to decide whether two boolean formulas can be made equivalent via negation mappings. That is, each variable is mapped to either itself or its complement. FN is **coNP**-hard and can be many-one reduced to FI [BRS98] but is not known to be many-one equivalent to FI. Because of the restricted nature of the negation transformation (compared to congruence), we expect FN to be an easier problem than FI.

- Can one show a stronger upper bound for FN than we have shown for FI? For example, is FN in $\Sigma_2\mathbf{P} \cap \Pi_2\mathbf{P}$?

Because of the similarity of FI and GI, it might be tempting to think that $\mathrm{FI} \in \mathbf{NP}^{\mathrm{GI}}$. However, recall that FI is **coNP**-hard so that this would imply that $\mathbf{coNP} \subseteq \mathbf{NP}^{\mathrm{GI}}$ which, in turn, would imply that $\Sigma_2\mathbf{P} = \mathbf{NP} \cdot \mathbf{coNP} \subseteq \mathbf{NP} \cdot \mathbf{NP}^{\mathrm{GI}} = \mathbf{NP}^{\mathrm{GI}}$. Hence the polynomial hierarchy would collapse [Sch88]. But we don't have such an argument for the following:

- Is $\mathrm{FI} \in \mathbf{coNP}^{\mathrm{GI}}$?

As we mentioned already in the introduction, the equivalence of *one-time-only branching programs* can be decided efficiently by randomized (Las Vegas type) algorithms: the equivalence problem is in the class **coRP**. Hence, the corresponding isomorphism problem is in $\mathbf{NP} \cdot \mathbf{coRP}$. An obvious question now is whether it is **NP**-hard. Thierauf [Thi98] showed that the isomorphism problem for one-time-only branching programs is *not* **NP**-hard, unless the polynomial hierarchy collapses to the second level.

Acknowledgments. We benefited from discussions with V. Arvind, Bernd Borchert, Jin-yi Cai, Lance Fortnow, and Toni Lozano.

REFERENCES

[BM88] L. Babai and S. Moran, *Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[BDG88] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* I, EATCS Monographs on Theoretical Computer Science 11, Springer-Verlag, Berlin, 1988.

[BDG91] J. Balcázar, J. Díaz, and J. Gabarró, *Structural Complexity* II, EATCS Monographs on Theoretical Computer Science 22, Springer-Verlag, Berlin, 1990.

[BG82] A. Blass and Y. Gurevich, *On the unique satisfiability problem*, Inform. and Comput., 55 (1982), pp. 80–88.

[BCW80] M. Blum, A. Chandra, and M. Wegman, *Equivalence of free Boolean graphs can be decided probabilistically in polynomial time*, Inform. Process. Lett., 10 (1980), pp. 80–82.

[BR93] B. Borchert and D. Ranjan, *The Subfunction Relations are Σ_2^p-Complete*, Tech. Report MPI-I-93-121, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1993.

[BRS98] B. Borchert, D. Ranjan, and F. Stephan, *On the computational complexity of some classical equivalence relations on boolean functions*, Theory Comput. Syst., 31 (1998), pp. 679–693.

[BCG+96] N. Bshouty, R. Cleve, R. Gavaldà, S. Kannan, and C. Tamon, *Oracles and queries that are sufficient for exact learning*, J. Comput. System Sci., 52 (1996), pp. 421–433.

[BT96] H. Buhrman and T. Thierauf, *The complexity of generating and checking proofs of membership*, in Proceedings of the 13th Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1046, Springer-Verlag, New York, 1996, pp. 75–87.

[CK91] P. Clote and E. Kranakis, *Boolean functions, invariance groups, and parallel complexity*, SIAM J. Comput., 20 (1991), pp. 553–590.

[FFK94] S. Fenner, L. Fortnow, and S. Kurtz, *Gap-definable counting classes*, J. Comput. System Sci., 48 (1994), pp. 116–148.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[GS89] S. Goldwasser and M. Sipser, *Private coins versus public coins in interactive proof systems*, Adv. Comput. Res., 5 (1989), pp. 73–90.

[Hof82] C. Hoffmann, *Group-Theoretic Algorithms and Graph Isomorphism*, Lecture Notes in Comput. Sci. 136, Springer-Verlag, New York, 1982.

[Hu79] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computation*, Addison-Wesley, Reading, MA, 1979.

[KST93] H. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser Boston, Boston, MA, 1993.

[Mat79] R. Mathon, *A note on the graph isomorphism counting problem*, Inform. Process. Lett., 8 (1979), pp. 131–132.

[PZ83] C. Papadimitriou and D. Zachos, *Two remarks on the power of counting*, in Proceedings of the 6th GI Conference on Theoretical Computer Science, Lecture Notes in Comput. Sci. 176, Springer-Verlag, New York, 1983, pp. 269–276.

[Sch88] U. Schöning, *Graph isomorphism is in the low hierarchy*, J. Comput. System Sci., 37 (1988), pp. 312–323.

[Sch89] U. Schöning, *Probabilistic complexity classes and lowness*, J. Comput. System Sci., 39 (1989), pp. 84–100.

[Thi98] T. Thierauf, *The isomorphism problem for read-once branching programs and arithmetic circuits*, Chicago J. Theoret. Comput. Sci., http://www.cs.uchicago.edu/publications/cjtcs/articles/contents (1998).

[Uma98] C. Umans, *The minimum equivalent DNF problem and shortest implicants*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 556–563.

[VV86] L. Valiant and V. Vazirani, *NP is as easy as detecting unique solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 1010–1022

THE COMPUTATIONAL COMPLEXITY TO EVALUATE REPRESENTATIONS OF GENERAL LINEAR GROUPS*

PETER BÜRGISSER†

Abstract. We describe a fast algorithm to evaluate irreducible matrix representations of complex general linear groups GL_m with respect to a symmetry adapted basis (Gelfand–Tsetlin basis). This is complemented by a lower bound, which shows that our algorithm is optimal up to a factor m^2 with regard to nonscalar complexity. Our algorithm can be used for the fast evaluation of special functions: for instance, we obtain an $O(\ell \log \ell)$ algorithm to evaluate all associated Legendre functions of degree ℓ. As a further application we obtain an algorithm to evaluate immanants, which is faster than previous algorithms due to Hartmann and Barvinok.

Key words. representations of general linear groups, Gelfand–Tsetlin bases, special functions, fast algorithms, lower bounds, immanants, completeness

AMS subject classifications. 68Q40, 22E70, 15A15, 33C25

PII. S0097539798367892

1. Introduction.

The theory of representations of Lie groups has countless applications in mathematics and physics. In particular, it is an indispensable tool of quantum mechanics. Let $D\colon G \to \mathrm{GL}_d$ be an irreducible (finite dimensional) continuous representation of the Lie group G. How fast can we compute the representation matrix $D(g)$ for a given $g \in G$? This question contains the problem of the efficient evaluation of special functions and orthogonal polynomials, which can be interpreted as matrix entries of some suitable representation D (see Vilenkin and Klimyk [22]).

In this paper, we investigate the above question for the general linear groups $G = \mathrm{GL}_m$ of complex m by m matrices. This includes the case of the unitary groups $\mathrm{U}(m)$, which are of particular importance for physics, since all the continuous irreducible representations of $\mathrm{U}(m)$ can be obtained from the rational irreducible ones of GL_m by restriction.

It is well known that the irreducible polynomial representations of the general linear group GL_m can be labeled by decreasing sequences $\lambda \in \mathbb{N}^m$ (cf. [3]). With respect to some chosen basis, such a representation is a group morphism

$$D_\lambda : \mathrm{GL}_m \longrightarrow \mathrm{GL}_{d_\lambda},\ A \mapsto D_\lambda(A),$$

and the entries of D_λ are homogeneous polynomials of degree $|\lambda| := \sum_i \lambda_i$ in the entries of A. The matrix $D_\lambda(A)$ is usually called an *invariant matrix* when the entries of A are interpreted as indeterminates. Littlewood [14, 15] found an explicit construction of invariant matrices. He obtained formulas for the polynomial entries of $D_\lambda(A)$ in terms of representations of the symmetric group. Grabmeier and Kerber [10] gave a modern derivation of such formulas, and based on this, they designed an algorithm for computing invariant matrices. This algorithm in fact computes a sparse representation of the polynomial entries of the invariant matrix.

*Received by the editors July 29, 1998; accepted for publication (in revised form) April 20, 1999; published electronically August 24, 2000. An extended abstract of this work appeared in Proceedings of the 10th Conference on Formal Power Series and Algebraic Combinatorics, University of Toronto, Canada, 1998, pp. 115–126.

http://www.siam.org/journals/sicomp/30-3/36789.html

†Department of Mathematics and Computer Science, University of Paderborn, Warburger Str. 100, D-33098 Paderborn, Germany (pbuerg@math.uni-paderborn.de).

However, the example of the m by m determinant already shows that this approach cannot be efficient for evaluating $D_\lambda(A)$ at specific entries $A \in \mathrm{GL}_m$ for larger m: the determinant polynomial has $m!$ terms, but it can be computed with only $O(m^3)$ arithmetic operations using Gaussian elimination.

As unitary representations of $\mathrm{U}(m)$ are important in quantum mechanics, it is not astonishing that various explicit constructions of representations have been developed by physicists. (See for instance the books by Biedenharn and Louck [2], where the entries of invariant matrices are called boson polynomials.)

Gelfand and Tsetlin [9] derived explicit expressions for $D_\lambda(A)$ for generators A of GL_m with respect to bases adapted to the chain of subgroups

$$\mathrm{GL}_m > \mathrm{GL}_{m-1} \times \mathbb{C}^\times > \mathrm{GL}_{m-2} \times (\mathbb{C}^\times)^2 > \cdots > (\mathbb{C}^\times)^m.$$

Such symmetry adapted bases are also called *Gelfand–Tsetlin bases*. A very detailed account of this can be found in the book by Vilenkin and Klimyk [22, Chapter 18]; in section 2 we just present the most basic facts.

The main result of this paper (Theorem 4.1) is an efficient algorithm to evaluate the representation D_λ with respect to a Gelfand–Tsetlin basis. It has a nonscalar cost roughly proportional to $m^2 d_\lambda$. Besides making optimal use of the symmetry (Schur's lemma), our algorithm uses several auxiliary algorithms: an efficient transformation of matrices with the block structure of a Jordan block to a direct sum of Jordan blocks, as well as the fast multiplication of Toeplitz matrices with vectors, based on the fast Fourier transform. These auxiliary algorithms are described in section 3. We remark that our algorithm was inspired by Clausen's fast Fourier transform [7] for the symmetric group, as well as Maslen's extension to compact Lie groups (see the survey [16]). Their techniques also rely heavily on symmetry adapted bases.

In section 5 we complement our algorithmic result by proving that d_λ nonscalar operations are indeed necessary for the computation of $D_\lambda(A)v$. This is easily obtained by combining Burnside's theorem with the dimension bound of algebraic complexity. It shows that our algorithm is optimal up to a factor of m^2 with respect to nonscalar complexity.

Our algorithm provides already in the special case of GL_2 new results. We obtain a fast rational $O(\ell \log \ell)$ algorithm for computing all the associated Legendre functions $P_\ell^\mu(\cos\theta)$, $|\mu| \le \ell$, of degree ℓ from $\cos\theta$ and $\sin\theta$. (See section 6.)

For computing individual entries of the invariant matrix, the cost of our algorithm may appear to be prohibitively high, mainly because the dimension d_λ can be very large. For instance, for $\lambda = (m, 0, \ldots, 0) \in \mathbb{N}^m$ we have $d_\lambda = \binom{2m-1}{m}$, which is exponential in m. Is this inherent to the problem, or are there faster algorithms running with a number of steps polynomially bounded in m?

For approaching this question, we do not focus on individual entries of the invariant matrix, but we study related functions having some invariant meaning. Let λ be a partition of m (or a Young diagram with m boxes). We consider the function sending $A \in \mathrm{GL}_m$ to the sum of the diagonal entries of the invariant matrix $D_\lambda(A)$ corresponding to the weight $(1, \ldots, 1)$. This turns out to be the so-called *immanant*,

$$\mathrm{im}_\lambda(A) = \sum_{\pi \in S_m} \chi_\lambda(\pi) \prod_{i=1}^{m} A_{i,\pi(i)},$$

of the matrix A corresponding to λ, which was introduced by Littlewood (cf. [15]). Here, χ_λ denotes the irreducible character of the symmetric group S_m belonging to λ

(cf. [3] or [12]). Note that this notion contains the permanent and determinant as special cases ($\chi_\lambda = 1$ or $\chi_\lambda = \mathrm{sgn}$).

From our algorithm for evaluating representations of GL_m we derive in section 7 an upper bound on the computational complexity of immanants which improves previous bounds due to Hartmann [11] and Barvinok [1] (see Theorem 7.2).

In a subsequent paper [4], we will complement this upper bound by intractability results. Strictly speaking, we will prove the completeness of certain families of immanants corresponding to hook or rectangular diagrams within the framework of Valiant's algebraic P-NP theory [20, 21, 5]. This means that these families of immanants cannot be evaluated by a polynomial number of arithmetic operations, unless this is possible for the family of permanents.

2. Preliminaries on representations of GL_m. We collect first some facts about the representations of the complex general linear group GL_m (see [3, 8]). Consider GL_{m-1} as the subgroup of GL_m fixing the last canonical basis vector e_m. A polynomial representation $D_\lambda \colon \mathrm{GL}_m \to \mathrm{GL}(V)$ with highest weight $\lambda \in \mathbb{N}^m$ restricted to GL_{m-1} splits according to the branching rule [3, section 5.6] into a direct sum of representations with highest weights $\mu \in \mathbb{N}^{m-1}$, which satisfy the betweenness conditions $\lambda_j \geq \mu_j \geq \lambda_{j+1}$ for $1 \leq j < m$. It is important that each representation corresponding to such μ occurs with *multiplicity* 1. Thus the decomposition of V into corresponding submodules V_μ is unique. We note that this is also the decomposition of V restricted to the subgroup $\mathrm{GL}_{m-1} \times \mathbb{C}^\times$, as the diagonal matrix $\mathrm{diag}(1, \ldots, 1, t)$ operates on V_μ by multiplication with $t^{|\lambda|-|\mu|}$. We recall that a vector $v \in V$ is said to be of weight $w \in \mathbb{N}^m$ iff $D_\lambda(\mathrm{diag}(t_1, \ldots, t_m))v = t_1^{w_1} \cdots t_m^{w_m} v$.

If we restrict the representation D_λ successively according to the chain of subgroups

$$\mathrm{GL}_m > \mathrm{GL}_{m-1} \times \mathbb{C}^\times > \mathrm{GL}_{m-2} \times (\mathbb{C}^\times)^2 > \cdots > (\mathbb{C}^\times)^m, \tag{2.1}$$

we finally end up with a decomposition of V into one-dimensional subspaces of weight vectors. This decomposition is unique, and bases of V adapted to this decomposition are called *Gelfand–Tsetlin bases*. Thus Gelfand–Tsetlin bases are uniquely determined up to a permutation of the basis elements and scaling. We remark that if V is a (finite dimensional) Hilbert space and D_λ restricted to $\mathrm{U}(m)$ is unitary, then a Gelfand–Tsetlin basis can be chosen to orthonormal.

The splitting behavior can be conveniently visualized by a layered graph $G(\lambda)$, whose nodes on level n ($1 \leq n \leq m$) are the occurring irreducible representations of V restricted to $\mathrm{GL}_n \times (\mathbb{C}^\times)^{m-n}$. These nodes can thus be uniquely described by pairs (ν, w), where $\nu \in \mathbb{N}^n$ is a partition and $w \in \mathbb{N}^{m-n}$ satisfies $|\nu| + |w| = |\lambda|$. A node on level n is connected in the graph $G(\lambda)$ with a node on level $n-1$ if the latter appears in the decomposition of the former upon restriction to $\mathrm{GL}_{n-1} \times (\mathbb{C}^\times)^{m-n+1}$ (see Figure 2.1).

The number of paths in $G(\lambda)$ between a node $x = (\nu, w)$ at level n and a node $x' = (\nu', w')$ at level $n' < n$ is just the multiplicity with which x' occurs in x when restricted to $\mathrm{GL}_{n'} \times (\mathbb{C}^\times)^{m-n'}$. We denote this multiplicity by $\mathrm{mult}(x, x')$. We call the maximum of $\mathrm{mult}(x, x')$ taken over all pairs of nodes *two levels apart* the *multiplicity* $\mathrm{mult}(\lambda)$ of the highest weight λ. (In Figure 2.1 we have $\mathrm{mult}(\lambda) = 2$.)

The vectors of a Gelfand–Tsetlin basis can be labeled by paths in $G(\lambda)$ going from the top node λ to a node at level 1. Such paths can be encoded as semistandard tableaus: for instance, in Figure 2.1 we have two vectors of weight $(1, 1, 1)$ corresponding to the tableaus $\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array}$ and $\begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array}$. The quantity $\mathrm{mult}(x, x')$ can thus be alternatively

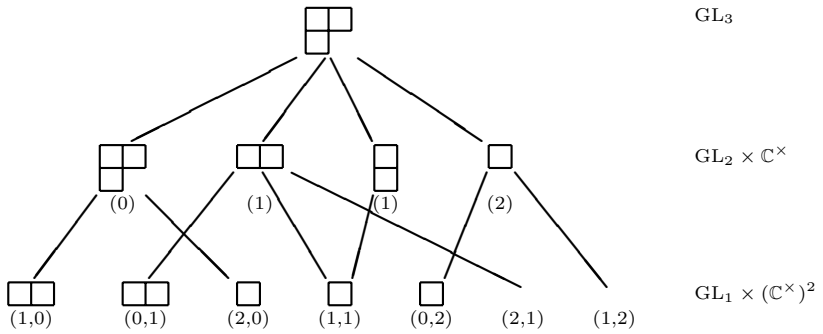


FIG. 2.1. *The graph $G(\lambda)$ for $\lambda = (2,1,0)$, $m = 3$.*

described as the number of semistandard tableaus of the skew diagram $\nu \setminus \nu'$ in which j occurs exactly w'_j times ($n' < j \le n$). These considerations also imply that the dimension d_λ equals the number of semistandard tableaus of the diagram λ. Moreover, the number of vectors of weight $(1,\dots,1)$ in a Gelfand–Tsetlin basis corresponding to λ equals the number s_λ of standard tableaus on the diagram of λ.

Remark 2.1. We have $d_\lambda \ge m$ if $\lambda = (\lambda_1,\dots,\lambda_m) \ne (m,\dots,m)$. For hook partitions $\lambda = (m-i,1,\dots,1)$ we have $\mathrm{mult}(\lambda) \le 2$.

By suitable scaling and ordering of the vectors of a Gelfand–Tsetlin basis, we can obtain a basis of V, which is adapted to the chain (2.1) of subgroups in a strong sense: the corresponding matrix representation D of GL_m satisfies the following conditions for all n.

1. The restriction $D \downarrow \mathrm{GL}_n \times (\mathbb{C}^\times)^{m-n}$ is *equal* to a direct sum of matrix representations of this subgroup.

2. Equivalent irreducible constituents of $D \downarrow \mathrm{GL}_n \times (\mathbb{C}^\times)^{m-n}$ are *equal*.

These properties are crucial for our computational purpose. For convenience, we will call such adapted bases also *Gelfand–Tsetlin bases*.

Consider now the matrix $B_{i,j}(t) \in \mathrm{GL}_m$ with entries 1 in the diagonal, entry t at position (i,j), and entries 0 elsewhere ($i \ne j$). Let $D\colon \mathrm{GL}_m \to \mathrm{GL}(V)$ be a rational representation. Then $F = F_{i,j}\colon \mathbb{C} \to \mathrm{GL}(V)$, $t \mapsto D(B_{i,j}(t))$ is a one-parameter subgroup: we have $F(s+t) = F(s)F(t)$ for $s,t \in \mathbb{C}$. Hence $F'(t) = F'(0)F(t)$, and therefore $F(t) = e^{tF'(0)}$. (Note that $F'(0)$ must be nilpotent.) Let $\epsilon_i \in \mathbb{N}^m$ denote the basis vector having components 0 except at position i, where the component equals 1.

LEMMA 2.2. *$F'_{i,j}(0)$ maps a vector of weight $w \in \mathbb{Z}^m$ into one of weight $w+\epsilon_i-\epsilon_j$.*

Proof. For a fixed vector v of weight w we may write

$$F(t)v = \sum_{s \ge 0} t^s u_s,$$

with vectors $u_s \in V$. We are going to show that u_s must be a vector of weight $w + s(\epsilon_i - \epsilon_j)$. Then we are finished, since $F'(0)v = u_1$.

We have $gB_{i,j}(t)g^{-1} = B_{i,j}(g_i t g_j^{-1})$ for a diagonal matrix $g = \mathrm{diag}(g_1,\dots,g_m)$. This implies

$$D(g)F(t) = D(g)D(B_{i,j}(t)) = D(gB_{i,j}(t)g^{-1})D(g) = F(g_i t g_j^{-1})D(g).$$

Therefore, as $D(g)v = g_1^{w_1} \cdots g_m^{w_m} v$, we obtain

$$\sum_{s\geq 0} t^s D(g)u_s = D(g)F(t)v = g_1^{w_1} \cdots g_m^{w_m} F(g_i t g_j^{-1})v = g_1^{w_1} \cdots g_m^{w_m} \cdot \sum_{s\geq 0} (g_i t g_j^{-1})^s u_s.$$

By comparing the coefficients of t, we see that u_s is indeed a vector of weight $w + s(\epsilon_i - \epsilon_j)$. □

3. Auxiliary fast linear algebra algorithms. We present some auxiliary algorithms, which we will need as subroutines in our algorithm for evaluating representations.

The first one is a variant of Gaussian elimination.

LEMMA 3.1. *Any matrix $A \in \mathrm{GL}_m$ can be factored as $A = A_N A_{N-1} \cdots A_1 \Delta$, where $N \leq 2m^2$, Δ is a diagonal matrix, and all A_i are elementary matrices of the form $B_{n-1,n}(t)$ or $B_{n,n-1}(t)$. Moreover, such a decomposition can be computed with $O(m^3)$ arithmetic operations.*

Proof. Recall that multiplying a matrix from the left by $B_{i,j}(t)$ has the effect of adding the t-fold of the jth row to the ith row. Also note the following: suppose the jth row of A equals zero. Then multiplying A from the left by $B_{i,j}(-1)B_{j,i}(1)$ has the effect of interchanging the jth row with the ith row.

By a sequence of elementary row operations affecting only *neighboring* rows, we can transform a given invertible matrix $A \in \mathrm{GL}_m$ into diagonal form Δ. Hereby, we first take care of the first column of A by working up from the bottom row, then we deal with the second column in a similar way, and so forth. We will illustrate the procedure in the case $m = 4$. Let the symbol $C_{i,j}$ denote either $B_{i,j}(-1)B_{j,i}(1)$ or $B_{i,j}(t)$ for some $t \in \mathbb{C}$. We can obtain a decomposition of the form

$$(C_{3,4}C_{2,3}C_{1,2})(C_{2,3}C_{1,2}C_{4,3})(C_{1,2}C_{3,2}C_{4,3})(C_{2,1}C_{3,2}C_{4,3})\, A = \Delta,$$

where Δ is a diagonal matrix. (The parentheses indicate the treatment of columns.) The number of occurring $C_{i,j}$ matrices equals $m(m-1)$ in the general situation. Moreover, the $C_{i,j}$ and Δ can be computed with $O(m^3)$ arithmetic operations from A. This proves the lemma. □

The next result is well known and relies on the fast Fourier transform (see, for instance, [6, Corollary 13.13]).

PROPOSITION 3.2. *Suppose $J \in \mathbb{C}^{r\times r}$ is a nilpotent Jordan block. Then e^{tJ} is a Toeplitz matrix. Thus $e^{tJ}u$ can be computed from $t \in \mathbb{C}$ and $u \in \mathbb{C}^r$ with $O(r \log r)$ arithmetic operations. For this, $O(r)$ nonscalar operations are sufficient.*

We remark that the computation of $e^{tJ}u$ is equivalent to the task of evaluating the polynomial $f(T) = \sum_{j=0}^{r-1} \frac{u_j}{j!} T^j$ and all its derivatives at $t \in \mathbb{C}$.

The first part of the following lemma shows that a matrix having the block structure of a Jordan block can be efficiently transformed to a direct sum of Jordan blocks. The second part follows then easily with Proposition 3.2.

LEMMA 3.3. *Let $M \in \mathbb{C}^{m\times m}$ be a matrix decomposed into r^2 blocks M_{ij} in $\mathbb{C}^{m_i \times m_j}$, $m = m_1 + \cdots + m_r$. Suppose that all block entries outside the lower diagonal are zero; that is, $M_{ij} = 0$ if $i \neq j+1$. Then the following is true.*

1. *There is an invertible block diagonal matrix $S = [S_{ij}] \in \mathbb{C}^{m\times m}$ (S_{ij} in $\mathbb{C}^{m_i \times m_j}$, $S_{ij} = 0$ for $i \neq j$) and a permutation matrix P such that $PSMS^{-1}P^{-1}$ is a direct sum of nilpotent Jordan blocks of size at most r.*
2. *The product $e^{tM}u$ can be computed with $O(\sum_{\rho=1}^{r} m_\rho^2 + m \log r)$ arithmetic operations from $t \in \mathbb{C}$ and $u \in \mathbb{C}^m$.*

Proof. For showing the first part, it is convenient to take a coordinate-free point of view. Let $V = V_1 \oplus \cdots \oplus V_r$ be a decomposition of vector spaces together with linear maps $\varphi_\rho \colon V_\rho \to V_{\rho+1}$ for $1 \le \rho < r$. Let $\varphi \colon V \to V$ be the linear map satisfying $\varphi(v) = \varphi_\rho(v)$ if $v \in V_\rho$, $\rho < r$, and $\varphi(v) = 0$ if $v \in V_r$. (Note that φ and φ_ρ are coordinate-free versions of M and $M_{\rho+1,\rho}$, respectively.) By a φ-chain of length t we understand an ordered set of vectors $\{v_1, \ldots, v_t\}$ such that $\varphi(v_\tau) = v_{\tau+1}$ for $1 \le \tau < t$, and $\varphi(v_t) = 0$. (φ-chains correspond to nilpotent Jordan blocks.) The first claim of the lemma amounts to showing the existence of a basis E_ρ for each V_ρ such that the basis $E_1 \cup \cdots \cup E_r$ of V is a disjoint union of φ-chains of length at most r.

For $1 \le \rho \le \sigma < r$ let $V_{\rho,\sigma}$ denote the kernel of the composition

$$\varphi_\sigma \circ \cdots \circ \varphi_{\rho+1} \circ \varphi_\rho \colon V_\rho \to V_{\sigma+1}$$

and set $V_{\rho,r} = V_\rho$ for $1 \le \rho \le r$. Note that $V_{\rho,\sigma} \subseteq V_{\rho,\sigma+1}$ and $\varphi_\rho^{-1}(V_{\rho+1,\sigma}) = V_{\rho,\sigma}$. In particular, $\varphi_\rho(V_{\rho,\sigma}) \subseteq V_{\rho+1,\sigma}$.

Choose subsets $E_{1,\sigma} \subseteq V_{1,\sigma}$ such that $E_{1,1} \cup \ldots, E_{1,\sigma}$ is a basis of $V_{1,\sigma}$ for all $1 \le \sigma \le r$. (This means that $E_{1,1} \cup \cdots \cup E_{1,r}$ is a basis of V_1 adapted to the flag $V_{1,1} \subseteq \cdots \subseteq V_{1,r}$ of subspaces.)

By induction on $\rho = 2, \ldots, r$ we are going to construct finite subsets $E_{\rho,\sigma} \subseteq V_{\rho,\sigma}$ satisfying the following conditions for all $\rho \le \sigma \le r$:

(i)$_\rho$ $E_{\rho,\rho} \cup \cdots \cup E_{\rho,\sigma}$ is a basis of $V_{\rho,\sigma}$,

(ii)$_\rho$ $\varphi_{\rho-1}(E_{\rho-1,\sigma}) \subseteq E_{\rho,\sigma}$.

Assume we have already constructed the $E_{\rho-1,\sigma}$ satisfying (i)$_{\rho-1}$ and (ii)$_{\rho-1}$. We claim that the subset $\varphi_{\rho-1}(E_{\rho-1,\sigma}) \subseteq V_{\rho,\sigma}$ is linearly independent modulo $V_{\rho,\sigma-1}$, provided $\rho \le \sigma$. Indeed, if we had a nontrivial linear combination

$$\sum_{v \in E_{\rho-1,\sigma}} \lambda_v \varphi_{\rho-1}(v) \in V_{\rho,\sigma-1},$$

then we would have

$$\sum_{v \in E_{\rho-1,\sigma}} \lambda_v v \in \varphi_{\rho-1}^{-1}(V_{\rho,\sigma-1}) = V_{\rho-1,\sigma-1}.$$

By our inductive assumption (i)$_{\rho-1}$, the set $E_{\rho-1,\rho-1} \cup \cdots \cup E_{\rho-1,\sigma-1}$ is a basis of $V_{\rho-1,\sigma-1}$ and $E_{\rho-1,\rho-1} \cup \cdots \cup E_{\rho-1,\sigma}$ is linearly independent. This is a contradiction! We may now choose subsets $E_{\rho,\sigma}$ containing $\varphi_{\rho-1}(E_{\rho-1,\sigma})$ such that $E_{\rho,\sigma}$ is linearly independent modulo $V_{\rho,\sigma-1}$. Then the conditions (i)$_\rho$ and (ii)$_\rho$ are satisfied.

We have now constructed a basis $E_\rho := E_{\rho,\rho} \cup \cdots \cup E_{\rho,r}$ for each of the spaces V_ρ. We write the basis $E := E_1 \cup \cdots \cup E_r$ as the disjoint union of the subsets

$$F := \bigcup_{\rho<\sigma} E_{\rho,\sigma} \quad \text{and} \quad G := \bigcup_{\rho} E_{\rho,\rho}.$$

By our construction, φ induces an injective map from F to $F \cup G$, and we have $\varphi(G) = \{0\}$. This abstract property easily implies that E is a disjoint union of φ-chains. It is obvious that the length of these chains cannot be bigger than r.

We now provide the proof of the second part of the lemma. Let $PSMS^{-1}P^{-1} = J = J_1 \oplus \cdots \oplus J_s$ be as in the statement of the first part. The J_σ are nilpotent Jordan blocks of size $r_\sigma \le r$. We have

$$e^{tM} = S^{-1}P^{-1}e^{tJ}PS, \; e^{tJ} = \bigoplus_{\sigma=1}^{s} e^{tJ_\sigma}.$$

We can compute $u' = Su$ from a given $u \in \mathbb{C}^m$ with $O(\sum_{\rho=1}^r m_\rho^2)$ arithmetic operations, since S is a block diagonal matrix. The vectors $u'_\sigma \in \mathbb{C}^{r_\sigma}$ satisfying $\oplus_{\sigma=1}^s u'_\sigma = Pu'$ are obtained without further arithmetic operations. By Proposition 3.2, we can compute each of the products $e^{tJ_\sigma}u'_\sigma$ with $O(r_\sigma \log r_\sigma)$ arithmetic operations. Thus we get $u'' := e^{tJ} \oplus_\sigma u'_\sigma$ from the u'_σ with $O(\sum_{\sigma=1}^s r_\sigma \log r_\sigma) \le O(m \log r)$ operations. Summarizing, we have computed $e^{tM}u$ from t and u using $O(\sum_{\rho=1}^r m_\rho^2 + m \log r)$ arithmetic operations. □

4. An algorithm for evaluating representations. The main result of this article is expressed in the following theorem. We call $\lambda \in \mathbb{N}^m$ constant iff its components are all equal.

THEOREM 4.1. *Let D_λ be the matrix representation of GL_m with respect to a Gelfand–Tsetlin basis with highest weight $\lambda \in \mathbb{N}^m$. We suppose that λ is not constant. Then the map*

$$\mathrm{GL}_m \times \mathbb{C}^{d_\lambda} \longrightarrow \mathbb{C}^{d_\lambda},\ (A, v) \mapsto D_\lambda(A)v$$

can be computed with $O(m^2(\mathrm{mult}(\lambda) + \log|\lambda|)\, d_\lambda)$ arithmetic operations. The nonscalar complexity is bounded by $O(m^2 d_\lambda + m\lambda_1)$.

Remark 4.2.

1. We assume exact arithmetic of complex numbers.

2. In the above upper bound the cost for *constructing* the algorithm (preconditioning) is not taken into account. However, the explicit formulas in Vilenkin and Klimyk [22, Chapter 18] for invariant matrices evaluated at special generators of GL_m suggest that this can also be done very efficiently.

3. We need the assumption that λ is not constant. Otherwise, the determinant could be evaluated with $O(m^2 \log m)$ operations (take $\lambda = (1, \ldots, 1)$).

COROLLARY 4.3. *The invariant matrix with respect to a nonconstant $\lambda \in \mathbb{N}^m$ and a Gelfand–Tsetlin basis can be evaluated at a matrix $A \in \mathrm{GL}_m$ with $O(m^2(\mathrm{mult}(\lambda) + \log|\lambda|)\, d_\lambda^2)$ arithmetic operations.*

Proof (of Theorem 4.1). We first factor the given matrix $A \in \mathrm{GL}_m$ according to Lemma 3.1 as $A = A_N A_{N-1} \cdots A_1 \Delta$. Note that the cost for doing this is dominated by $O(m^2 d_\lambda)$, since $d_\lambda \ge m$ for a nonconstant $\lambda \in \mathbb{N}^m$ by Remark 2.1. We therefore have $D_\lambda(A) = D_\lambda(A_N)D_\lambda(A_{N-1}) \cdots D_\lambda(A_1)D_\lambda(\Delta)$. For given $v \in \mathbb{C}^{d_\lambda}$ we first compute $v_0 = D_\lambda(\Delta)v$ and then, successively, $v_i = D_\lambda(A_i)v_{i-1}$ for $1 \le i \le N$. Obviously, $v_t = D_\lambda(A)v$.

As a Gelfand–Tsetlin basis consists of weight vectors, and the matrix $D_\lambda(\Delta)$ is diagonal with entries $t_1^{w_1} \cdots t_m^{w_m}$, where $w_i \le \lambda_1$. Thus $D_\lambda(\Delta)v$ can be certainly computed with $O(m\, d_\lambda \log \lambda_1) \le O(m\, d_\lambda \log|\lambda|)$ arithmetic operations. (Note that $2 \log w_i$ multiplications are sufficient to obtain $t_i^{w_i}$.)

It remains to show that we can compute each of the products $D_\lambda(A_i)v_i$ with $O((\mathrm{mult}(\lambda) + \log|\lambda|)d_\lambda)$ arithmetic operations. We assume that $A_i = B_{n-1,n}(t)$, the case of $A_i = B_{n,n-1}(t)$ being analogous. Let us write $V = \mathbb{C}^{d_\lambda}$ and interpret V as a GL_m-module via D_λ. Recall the graph $G(\lambda)$ introduced in section 2, which describes the splitting behavior of V. We have

$$V \downarrow \mathrm{GL}_n \times (\mathbb{C}^\times)^{m-n} = \bigoplus_x \bigoplus_{j=1}^{f(x)} V_{x,j}, \tag{4.1}$$

where the first sum is over all nodes x of $G(\lambda)$ at level n, $f(x)$ equals $\mathrm{mult}(\lambda, x)$, and $V_{x,j}$ is an irreducible $\mathrm{GL}_n \times (\mathbb{C}^\times)^{m-n}$-module of type x. Because of the symmetry

adaptation, the decomposition (4.1) is compatible with our Gelfand–Tsetlin basis; that is, subsets of this basis form a basis of each $V_{x,j}$. As A_i is contained in GL_n, the matrix $D_\lambda(A_i)$ decomposes according to (4.1) into

$$D_\lambda(A_i) = \oplus_x \oplus_j D_{x,j}(A_i).$$

If we write $v_i = \oplus_x \oplus_j v_{x,j}$ according to (4.1) (this decomposition can be done for free), we have $D_\lambda(A_i)v_i = \oplus_x \oplus_j D_{x,j}(A_i)v_{x,j}$. Now it is sufficient to show that each of the products $D_{x,j}(A_i)v_{x,j}$ can be computed with $O((\mathrm{mult}(\nu_x) + \log|\nu_x|)\, d_{\nu_x})$ arithmetic operations, where $x = (\nu_x, w)$, $\nu_x \in \mathbb{N}^n$, $d_{\nu_x} = \dim V_{x,j}$. Indeed, then $D_\lambda(A_i)v_i$ can be computed with a number of arithmetic operations bounded by

$$\begin{aligned}\sum_x (\mathrm{mult}(\nu_x) + \log|\nu_x|)\, f(x) d_{\nu_x} &\le (\mathrm{mult}(\lambda) + \log|\lambda|) \sum_x f(x) d_{\nu_x} \\ &= (\mathrm{mult}(\lambda) + \log|\lambda|)\, d_\lambda\end{aligned}$$

up to a constant factor. By symmetry adaptation, a subset of the original Gelfand–Tsetlin basis forms a Gelfand–Tsetlin basis of $V_{x,j}$, and $D_{x,j}$ is the corresponding matrix representation. We may therefore continue our argumentation assuming that $n = m$.

We have to prove now that we can compute the product $D_\lambda(B_{m-1,m}(t))v$ with $O((\mathrm{mult}(\lambda) + \log|\lambda|)\, d_\lambda)$ arithmetic operations. Put $F(t) := D_\lambda(B_{m-1,m}(t))$ and $\Gamma := F'(0)$. Similarly as in (4.1), we have the decomposition

$$V \downarrow \mathrm{GL}_{m-2} \times (\mathbb{C}^\times)^2 = \bigoplus_\nu \bigoplus_{a=0}^{|\lambda|-|\nu|\, f(\nu,a)} \bigoplus_{j=1} V_{\nu,a,j}, \tag{4.2}$$

where the first two sums are over all ν, a such that the pair $x := (\nu, w)$ with $w := (a, |\lambda| - |\nu| - a))$ is a node of $G(\lambda)$ at level $m-2$. The irreducible $\mathrm{GL}_{m-2} \times (\mathbb{C}^\times)^2$-module $V_{\nu,a,j}$ is of type x, and $f(\nu, a) = \mathrm{mult}(\lambda, x)$. Note that $f(\nu, a) \le \mathrm{mult}(\lambda)$.

$F(t)$ commutes with GL_{m-2}, and hence so does Γ. Therefore, Γ maps the isotypical components

$$W_\nu := \bigoplus_a \bigoplus_j V_{\nu,a,j}$$

of $V \downarrow \mathrm{GL}_{m-2}$ into itself. Note that $\dim W_\nu = g(\nu) d_\nu$, where $g(\nu) := \sum_a f(\nu, a)$. From Lemma 2.2 we know that Γ maps $\oplus_j V_{\nu,a,j}$ into $\oplus_j V_{\nu,a+1,j}$. Now we decompose Γ according to (4.2) into $d_\nu \times d_\nu$-matrices $\Gamma_{(\nu',a',j'),(\nu,a,j)}$. From the observations made just before, we see that these matrices vanish unless $\nu' = \nu$ and $a' = a + 1$. Such a matrix affords a GL_{m-2}-module morphism $V_{\nu,a,j} \to V_{\nu,a',j'}$. On the other hand, the identity matrix affords as well a GL_{m-2}-module morphism between these spaces, since our basis is adapted to this subgroup. Schur's lemma implies therefore that $\Gamma_{(\nu,a',j'),(\nu,a,j)}$ must be a multiple of the $d_\nu \times d_\nu$ identity matrix.

Let Γ^ν denote the matrix $[\Gamma_{(\nu,a',j'),(\nu,a,j)}]_{(a',j'),(a,j)}$, that is, the matrix of Γ restricted to W_ν. It is not hard to see that Γ^ν equals, after some suitable permutation of our basis of W_ν, a direct sum of d_ν *identical copies* of a matrix $M^\nu \in \mathbb{C}^{g(\nu)\times g(\nu)}$ (see Figure 4.1). This matrix M^ν has a decomposition into $(|\lambda| - |\nu| + 1)^2$ blocks $M^\nu_{a',a} \in \mathbb{C}^{f(\nu,a')\times f(\nu,a)}$, and all blocks outside the lower diagonal vanish: $M^\nu_{a',a} = 0$ unless $a' = a + 1$.

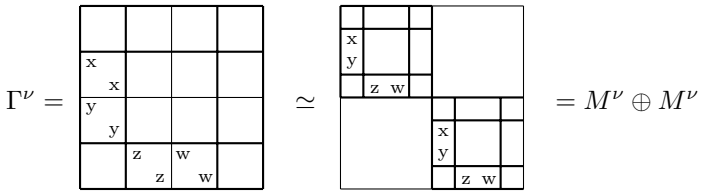


FIG. 4.1. *The matrix* Γ^ν *for* $\lambda = (2,1,0,0)$ *and* $\nu = (1,0)$. *We have* $f(\nu,0) = 1$, $f(\nu,1) = 2$, $f(\nu,2) = 1$, $g(\nu) = 4$, *and* $d_\nu = 2$.

From Lemma 3.3(2) we know that a product $e^{tM^\nu}u$ can be computed from $t \in \mathbb{C}$ and $u \in \mathbb{C}^{g(\nu)}$ with a number of arithmetic operations bounded by

$$\sum_a f(\nu,a)^2 + g(\nu)\log(|\lambda| - |\nu| + 1) \le g(\nu)\mathrm{mult}(\lambda) + g(\nu)\log(|\lambda|+1)$$

up to a constant factor. Therefore, a product $F(t)v$ can be computed with

$$O\Big(\sum_\nu d_\nu g(\nu)\big(\mathrm{mult}(\lambda) + \log(|\lambda|+1)\big)\Big)$$

arithmetic operations, which proves the claim, as $d_\lambda = \sum_\nu d_\nu g(\nu)$.

The estimation of the nonscalar complexity is similar. □

5. A lower bound. The subsequent lower bound result shows that our algorithm is optimal up to a factor of m^2 with respect to the number of nonscalar operations.

THEOREM 5.1. *Let* $\lambda \in \mathbb{N}^m$ *be such that* $|\lambda| > 1$, *and let* $v \in \mathbb{C}^{d_\lambda}$ *be nonzero. Let* D_λ *denote a matrix representation of* GL_m *with highest weight* λ. *Then any arithmetic algorithm (formally, algebraic computation tree) computing the map* $\mathrm{GL}_m \to \mathbb{C}^{d_\lambda}$, $A \mapsto D_\lambda(A)v$ *requires at least* d_λ *nonscalar operations. The evaluation of the invariant matrix* $\mathrm{GL}_m \to \mathrm{GL}_{d_\lambda}$, $A \mapsto D_\lambda(A)$ *requires at least* d_λ^2 *nonscalar operations.*

Proof. The theorem of Burnside (cf. [13]) states that the linear hull of the image of D_λ equals $\mathbb{C}^{d_\lambda \times d_\lambda}$, as D_λ is irreducible. Therefore, the entries of the invariant matrix corresponding to λ are linearly independent polynomials of degree $|\lambda| > 1$. The dimension bound in [6, (4.12)] easily implies the claims. □

6. Fast evaluation of Legendre functions. The algorithm of section 4 can be applied to evaluate many special functions and orthogonal polynomials (Legendre, Jacobi, Gegenbauer polynomials, generalized Beta functions, etc.), since all these are matrix entries of a suitable representation of GL_m (compare [22]). We illustrate this here by the simple example of the associated Legendre functions and obtain a new result.

For the following facts about the representations of GL_2 or the special unitary group $\mathrm{SU}(2)$, see [22, section 6.2]. The natural operation of GL_2 on the space V of homogeneous bivariate polynomials of degree 2ℓ in the indeterminates X, Y affords an irreducible representation of highest weight $\lambda = (2\ell, 0)$ and dimension $d_\lambda = 2\ell + 1$. All irreducible representations of GL_2 are obtained in this way for $\ell \in \frac{1}{2}\mathbb{N}$. Note that $\mathrm{mult}(\lambda) = 1$. In what follows, we will assume that $\ell \in \mathbb{N}$. The basis $(\psi_k)_{-\ell \le k \le \ell}$ given by the elements

$$\psi_k := \frac{X^{\ell-k}\,Y^{\ell+k}}{\sqrt{(\ell-k)!(\ell+k)!}}$$

is a Gelfand–Tsetlin basis having the additional property that the corresponding matrix representation $D^\ell = [D^\ell_{\mu\nu}]_{-\ell\le\mu,\nu\le\ell}$ restricted to SU(2) is unitary. Consider the special unitary matrix

$$A(\theta) := \begin{pmatrix} \cos\theta/2 & i\sin\theta/2 \\ i\sin\theta/2 & \cos\theta/2 \end{pmatrix}.$$

Theorem 4.1 yields an algorithm to compute a column of $D^\ell(A(\theta))$ from $\cos\theta/2$ and $\sin\theta/2$ with $O(\ell\log\ell)$ arithmetic operations. A closer look at this algorithm reveals that the (rational) computation may start with $\cos\theta$ and $\sin\theta$. Indeed, we have the matrix factorization

$$A(\theta) = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \begin{pmatrix} 1 & b \\ 0 & 1 \end{pmatrix} \begin{pmatrix} c & 0 \\ 0 & c^{-1} \end{pmatrix},$$

where $a = i(1-\cos\theta)/\sin\theta$, $2b = i\sin\theta$, and $c = \cos\theta/2$. Moreover, note that $c^{w_1}(c^{-1})^{w_2} = (c^2)^{w_1-\ell} = (\frac{1+\cos\theta}{2})^{w_1-\ell}$ for all weights (w_1, w_2) of V.

In turns out that the middle column of $D^\ell(A(\theta))$ just contains, up to some scaling, the associated Legendre functions P^μ_ℓ (cf. [22, section 6.3.7 (3)]: we have for $|\mu|\le\ell$

$$D^\ell_{-\mu,0}(A(\theta)) = i^{-\mu}\sqrt{\frac{(\ell-\mu)!}{(\ell+\mu)!}}\,P^\mu_\ell(\cos\theta).$$

COROLLARY 6.1. *All the associated Legendre functions $P^\mu_\ell(\cos\theta)$, $|\mu|\le\ell$, can be computed from $\cos\theta$ and $\sin\theta$ by an algorithm using only rational operations with $O(\ell\log\ell)$ arithmetic operations. The nonscalar complexity is bounded by $O(\ell)$.*

Remark 6.2. One can give a direct proof of Corollary 6.1 as follows. The associated Legendre functions P^μ_ℓ satisfy $(0\le\mu\le\ell)$

$$P^\mu_\ell(x) = (-1)^\mu(1-x^2)^{\mu/2}\frac{d^\mu P_\ell(x)}{dx^\mu},$$

where $P_\ell(x)$ denotes the Legendre polynomial of degree ℓ. (This again shows that $P^\mu_\ell(\cos\theta)$ is a polynomial in $\cos\theta$ and $\sin\theta$.) By Proposition 3.2, we can evaluate P_ℓ and all its derivatives at x with $O(\ell\log\ell)$ arithmetic operations (compare the comment following this proposition). Moreover, we have for $0\le\mu\le\ell$ that

$$P^{-\mu}_\ell(x) = (-1)^\mu\frac{\Gamma(\ell-\mu+1)}{\Gamma(\ell+\mu+1)}P^\mu_\ell(x)$$

(cf. [22, section 3.5.8 (10)]; Γ stands for the Gamma function). This finishes the alternative proof of Corollary 6.1.

7. Fast evaluation of immanants. The *immanant* of a matrix $A\in\mathbb{C}^{m\times m}$ corresponding to a partition λ of m is defined as (cf. [15])

$$\mathrm{im}_\lambda(A) = \sum_{\pi\in S_m}\chi_\lambda(\pi)\prod_{i=1}^m A_{i,\pi(i)},$$

where χ_λ denotes the irreducible character of the symmetric group S_m belonging to λ (cf. [3] or [12]). Note that this notion contains the permanent and determinant as special cases ($\chi_\lambda = 1$ or $\chi_\lambda = \mathrm{sgn}$). The reader may find some algebraic properties of immanants in Merris [17, 18].

The following lemma reduces the evaluation of immanants to the evaluation of invariant matrices. It states that immanants are in fact obtained as the sum of the diagonal entries of invariant matrices corresponding to the weight $(1,\dots,1)$. The proof of the lemma follows easily from Theorem 2.4 in [10].

LEMMA 7.1. *Let $\lambda \in \mathbb{N}^m$ be a partition of m. Suppose $D_\lambda = [D_{i,j}]$ is an irreducible matrix representation of GL_m of type λ with respect to a basis of weight vectors. Then we have for all $A \in \mathrm{GL}_m$ that*

$$\mathrm{im}_\lambda(A) = \sum_i D_{i,i}(A),$$

where the sum is over all i corresponding to basis vectors of weight $(1,\dots,1)$.

If we extend the above sum over all indices, we get the character of D_λ evaluated at A. It is interesting that this value can always be computed with a *polynomial number* of arithmetic operations in m. See Proposition 7.4 at the end of this section.

By combining Theorem 4.1 with the above lemma we get the following result.

THEOREM 7.2. *Let $\lambda \in \mathbb{N}^m$ be a nonconstant partition of m, and let s_λ denote the number of standard tableaus on the diagram of λ. One can compute $\mathrm{im}_\lambda(A)$ from $A \in \mathrm{GL}_m$ with a number of arithmetic operations bounded by*

$$O\big(m^2(\mathrm{mult}(\lambda) + \log m)\, s_\lambda d_\lambda\big).$$

The nonscalar complexity is bounded by $O(m^2 s_\lambda d_\lambda)$.

Remark 7.3. Hartmann [11] proved the upper bound $m^{6(m-s)+4}$ for the nonscalar complexity to evaluate permanents corresponding to partitions with at most s parts. Barvinok [1] showed the upper bound $O(m^3 d_\lambda^4)$ for the total complexity. Our bound improves Barvinok's, as $s_\lambda \le d_\lambda$ and $\mathrm{mult}(\lambda) \le d_\lambda$.

To compare our bound with those of Hartmann, consider hook partitions $\lambda = (k,1,\dots,1) \in \mathbb{N}^m$. For such λ one can show that

$$s_\lambda = \tbinom{m-1}{k-1},\ d_\lambda = \tfrac{m+k-1}{m}\binom{m+k-2}{m-k,\,k-1,\,k-1},$$

and hence $m^2 s_\lambda d_\lambda \le m^2 18^m$. This is considerably smaller than Hartmann's bound m^{6k} if $k \ge m/2$.

For permanents, our Theorem 7.2 yields the bound $O(m^{1.5} 4^m \log m)$, which is not too far away from the best-known upper bound $O(m 2^m)$ due to Ryser [19].

In a subsequent paper [4], we will complement this upper bound by completeness results within Valiant's algebraic P-NP theory [20, 21]. (For a comprehensive account of this theory, see [5].) In fact, we will prove the completeness of certain families of immanants corresponding to hook or rectangular diagrams.

We close by showing that the characters of GL_m can be evaluated very rapidly. Let $2 \le \omega < 3$ denote the exponent of matrix multiplication (cf. [6]).

PROPOSITION 7.4. *Let $\epsilon > 0$. Then for all m, the character $\mathrm{Tr}(D_\lambda(A))$ can be evaluated at a given matrix $A \in \mathrm{GL}_m$ with $O(\max\{\lambda_1, m\}^{\omega+\epsilon})$ arithmetic operations.*

Proof. The Schur polynomial of λ is defined as $S_\lambda = \mathrm{Tr}(D_\lambda(\mathrm{diag}(x_1,\dots,x_m)))$. Let σ_i denote the ith elementary symmetric polynomial in m variables, and set $\sigma_i = 0$ if $i < 0$ or $i > m$. Moreover, let $\mu = (\mu_1,\dots,\mu_{\lambda_1})$ be the partition conjugate to λ. Giambelli's formula states that (cf. [8, section A.1 (A.6)])

$$S_\lambda = \det[\sigma_{\mu_i+j-i}]_{1\le i,j\le\lambda_1}.$$

Let $T^m + \sum_{i=1}^{m} (-1)^i c_i(A) T^{m-i}$ be the characteristic polynomial of the matrix A with eigenvalues $x_1, \ldots, x_m$. Then we have $c_i(A) = \sigma_i(x)$ for all i. Therefore, we get from Giambelli's formula that

$$\mathrm{Tr}(D_\lambda(A)) = S_\lambda(x) = \det[c_{\mu_i + j - i}(A)]_{1 \le i,j \le \lambda_1}.$$

The algorithm is now as follows. First, compute the coefficients $c_i(A)$ of the characteristic polynomial for given A with $O(m^{\omega+\epsilon})$ arithmetic operations (cf. [6, section 16.6]). Then compute $\mathrm{Tr}(D_\lambda(A))$ by evaluating the determinant using $O(\lambda_1^{\omega+\epsilon})$ operations (cf. [6, section 16.4]). □

Acknowledgments. I thank M. Clausen and A. Shokrollahi for pointing out to me the papers by Grabmeier and Kerber, and Barvinok, respectively. I am grateful to Steve Smale for inviting me to the City University of Hong Kong, where this work was completed.

REFERENCES

[1] A. Barvinok, *Computational complexity of immanants and representations of the full linear group*, Funct. Anal. Appl., 24 (1990), pp. 144–145.
[2] L. Biedenharn and J. Louck, *Angular Momentum in Quantum Physics: Theory and Application; The Racah-Wigner Algebra in Quantum Theory*, Encyclopedia of Mathematics and its Applications 8–9, Addison-Wesley, Reading, MA, 1981.
[3] H. Boerner, *Representations of Groups*, Elsevier–North Holland, Amsterdam, 1970.
[4] P. Bürgisser, *The computational complexity of immanants*, SIAM J. Comput., 30 (2000), pp. 1023–1040.
[5] P. Bürgisser, *Completeness and Reduction in Algebraic Complexity Theory*, Algorithms Comput. Math. 7, Springer Verlag, Berlin, 2000.
[6] P. Bürgisser, M. Clausen, and M. Shokrollahi, *Algebraic Complexity Theory*, Grundlehren Math. Wiss. 315, Springer Verlag, New York, 1997.
[7] M. Clausen, *Fast generalized Fourier transforms*, Theoret. Comput. Sci., 67 (1989), pp. 55–63.
[8] W. Fulton and J. Harris, *Representation Theory*, Grad. Texts in Math. 129, Springer Verlag, New York, 1991.
[9] I. Gelfand and M. Tsetlin, *Finite dimensional representations of the group of unimodular matrices*, Dokl. Akad. Nauk SSSR, 71 (1950), pp. 825–828 (in Russian).
[10] J. Grabmeier and A. Kerber, *The evaluation of irreducible polynomial representations of the general linear groups and of the unitary groups over fields of characteristic* 0, Acta Appl. Math., 8 (1987), pp. 271–291.
[11] W. Hartmann, *On the complexity of immanants*, Linear and Multilinear Algebra, 18 (1985), pp. 127–140.
[12] G. James and A. Kerber, *The Representation Theory of the Symmetric Group*, Addison-Wesley, Reading, MA, 1981.
[13] S. Lang, *Algebra*, 2nd. ed., Addison-Wesley, Reading, MA, 1984.
[14] D. Littlewood, *The construction of invariant matrices*, Proc. London Math. Soc. (2), 43 (1937), pp. 226–240.
[15] D. Littlewood, *The Theory of Group Characters and Matrix Representations of Groups*, Oxford University Press, Oxford, UK, 1940.
[16] D. Maslen and D. Rockmore, *Generalized FFTs—a survey of some recent results*, in Groups and Computation II DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 28, AMS, Providence, RI, 1997, pp. 183–237.
[17] R. Merris, *On vanishing decomposable symmetrized tensors*, Linear and Multilinear Algebra, 5 (1977), pp. 79–86.
[18] R. Merris, *Recent advances in symmetry classes of tensors*, Linear and Multilinear Algebra, 7 (1979), pp. 317–328.
[19] H. J. Ryser, *Combinatorial Mathematics*, Carus Math. Monogr. 14, Math. Assoc. America, New York, 1963.
[20] L. Valiant, *Completeness classes in algebra*, in Proceedings of the 11th ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 249–261.

[21] L. Valiant, *Reducibility by algebraic projections*, in Logic and Algorithmic: An International Symposium held in honor of Ernst Specker, Monographies de L'Enseignement Mathématique 30, L'Enseignement Mathématique, Geneva, 1982, pp. 365–380.

[22] N. Vilenkin and A. Klimyk, *Representations of Lie groups and special functions,* I, II*, and* III, Math. Appl. 72, 74, 75, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1991, 1992, 1993.

SIAM J. COMPUT.
Vol. 30, No. 3, pp. 1023–1040

THE COMPUTATIONAL COMPLEXITY OF IMMANANTS*

PETER BÜRGISSER†

Abstract. Permanents and determinants are special cases of immanants. The latter are polynomial matrix functions defined in terms of characters of symmetric groups and corresponding to Young diagrams. Valiant has proved that the evaluation of permanents is a complete problem in both the Turing machine model (#P-completeness) as well as in his algebraic model (VNP-completeness). We show that the evaluation of immanants corresponding to hook diagrams or rectangular diagrams of polynomially growing width is both #P-complete and VNP-complete.

Key words. permanents, immanants, computational complexity, algebraic completeness

AMS subject classifications. 15A15, 68Q40, 68Q15

PII. S0097539798367880

1. Introduction.

The *permanent* $\operatorname{per}(A)$ of an n by n matrix $A = [a_{i,j}]$ is defined by

$$\operatorname{per}(A) := \sum_{\pi} \prod_{i=1}^{n} a_{i,\pi(i)},$$

where the summation is over all permutations π in S_n. Note that in contrast to the determinant, each term has a positive sign.

From the viewpoint of computational complexity, the determinant and permanent have, in spite of the similarity in their definitions, very little in common. While there are efficient polynomial time algorithms for the evaluation of the determinant, the best-known algorithm for the evaluation of the permanent of an n by n matrix needs $O(n2^n)$ arithmetic operations (Ryser [20]). A hypothesis due to Valiant in fact claims that the permanent cannot be computed with a polynomial number of arithmetic operations. This hypothesis is supported by Valiant's famous result [23] stating that the problem to evaluate the permanent of a matrix with entries in $\{0,1\}$ is #P-complete, as well as his analogous VNP-completeness result [22] in a framework of algebraic computations.

Both permanents and determinants are special cases of immanants introduced by Littlewood [16]. To define these polynomial matrix functions, we have to rely on some basic facts about the characters of the symmetric groups, which can be found for instance in the books by Boerner [2], James and Kerber [13], or Fulton and Harris [9].

It is known that the irreducible characters of the symmetric group S_n can be labeled by *partitions* of n, i.e., by decreasing sequences $\lambda = (\lambda_1, \ldots, \lambda_s)$ of natural numbers adding up to n. A partition will be identified with its (Young) *diagram* $\{(i,j) \mid 1 \le j \le \lambda_i\}$, which can be visualized as a left-justified arrangement of λ_i boxes in the ith row. (Compare Figure 3.1.) We call $|\lambda| := \sum_i \lambda_i$ the *size* and λ_1 the *width* of λ and will use the notation $\lambda \vdash n$ to express that λ is a partition or a diagram of size n. A diagram is called *rectangular* iff $\lambda_1 = \cdots = \lambda_s$.

*Received by the editors July 29, 1998; accepted for publication (in revised form) April 20, 1999; published electronically August 24, 2000. The results of this work were announced in Proceedings of the 10th Conference on Formal Power Series and Algebraic Combinatorics, University of Toronto, Canada, 1998, pp. 115–126.

http://www.siam.org/journals/sicomp/30-3/36788.html

†Department of Mathematics and Computer Science, University of Paderborn, Warburger Str. 100, D-33098 Paderborn, Germany (pbuerg@math.uni-paderborn.de).

Let $\chi_\lambda\colon S_n \to \mathbb{Z}$ denote the irreducible character of the symmetric group S_n corresponding to the diagram $\lambda \vdash n$. The *immanant* of an n by n matrix $A = [a_{i,j}]$ corresponding to λ is defined by

$$\mathrm{im}_\lambda(A) := \sum_{\pi \in S_n} \chi_\lambda(\pi) \prod_{i=1}^{n} a_{i,\pi(i)}.$$

For the "horizontal" diagrams given by $\lambda = (n)$ we have $\chi_\lambda = 1$, and the corresponding immanants specialize to the permanents. In the case of the "vertical" diagrams given by $\lambda = (1, \ldots, 1)$ we get $\chi_\lambda = \mathrm{sgn}$, and the immanants specialize to the determinants. These diagrams are special cases of those described by $(k, 1, \ldots, 1)$, which are called *hook diagrams* because of their shape. By *hook immanants* or *rectangular immanants* we will understand the immanant polynomials corresponding to diagrams of the corresponding shape. The reader may find some algebraic properties of immanants in Merris [17, 18].

Our interest for immanants stems from the fact that they constitute a natural parameterized set of polynomials, which allows us to study the change of computational complexity from easy (polynomial time computable) to difficult (complete) as the diagram λ of size n varies between the vertical and the horizontal diagram. To make this more specific, think for instance of the set of hook diagrams $(k, 1, \ldots, 1) \vdash n$ as the parameter k varies between 1 and n.

In the previous paper [3] we have developed a fast algorithm to evaluate representations of general linear groups. As a byproduct, we obtained an algorithm to evaluate the immanant im_λ at a matrix A with nonscalar cost proportional to $n^2 s_\lambda d_\lambda$, where s_λ and d_λ denote the number of standard tableaus and semistandard tableaus on the diagram of λ, respectively. This upper bound improves previous bounds due to Hartmann [11] and Barvinok [1].

In the present article we complement the upper bounds in [3] by completeness results for certain families of immanants. The results will be formulated in Valiant's algebraic P-NP theory, centering around the notion of VNP-completeness. The main features of this theory are recalled in section 2. Note that all VNP-completeness statements in this article refer to a ground field of characteristic zero.

A sequence $(\lambda^{(n)})$ of diagrams such that the size of $\lambda^{(n)}$ is polynomially bounded in n will be called a *p-sequence of diagrams*. If, additionally, the width of $\lambda^{(n)}$ is growing at least polynomially in the size of $\lambda^{(n)}$, then we call such a sequence to be of *polynomially growing width*. The corresponding families of immanant polynomials will also be called so.

We have the following conjecture.

CONJECTURE 1.1. *Any family of immanant polynomials of polynomially growing width is VNP-complete.*

The achievement of this paper is the proof of this conjecture in two special situations: for hook and rectangular immanants. The statement for hook immanants generalizes a result by Hartmann [11], while the claim for rectangular immanants answers an open problem posed by Strassen [21, Problem 14.2].

THEOREM 1.2. *Any family of hook immanants or rectangular immanants of polynomially growing width is VNP-complete.*

For sequences of diagrams of bounded width we have so far no clear idea of the complexity of the corresponding immanants. We raise the following question.

PROBLEM 1.3. *Is the family of rectangular immanants corresponding to rectangles of width* 2 *VNP-complete?*

We remark that families of hook immanants of bounded width are p-computable. This is an immediate consequence of the upper complexity bound in [3] mentioned before.

Our proofs also yield #P-completeness results for the problem to evaluate immanants at matrices A with entries in $\{0,1\}$. However, note that $\mathrm{im}_\chi(A)$ may be negative, with absolute value bounded by $n!\,n^n \le n^{2n}$ for $A \in \{0,1\}^{n\times n}$. We will therefore interpret the evaluation problem below as the modified problem to compute $\mathrm{im}_\chi(A) + n^{2n}$ from $A \in \{0,1\}^{n\times n}$.

COROLLARY 1.4. *Assume in Theorem* 1.2 *that the sequence of diagrams is polynomial time computable. Then the problem to evaluate the corresponding immanant at a given matrix with entries in* $\{0,1\}$ *is* #P*-complete.*

The paper is organized as follows. In section 2 we recall the main features of Valiant's algebraic P-NP theory. The goal of section 3 is the proof of an auxiliary result (Lemma 3.1) which is crucial for our completeness proofs. It expresses values of characters corresponding to rectangular diagrams by characters of hook diagrams. Section 4 is devoted to the proof that families of immanants are indeed p-definable.

In section 5, we provide the proof of Theorem 1.2 in several steps. The general strategy is to identify alternating sums of immanants corresponding to smaller partitions as a projection of a given immanant (Lemma 5.1), by applying the Murnaghan–Nakayama rule for the characters of the symmetric group. In combination with Lemma 3.1 we exhibit alternating sums of hook immanants as projections of rectangular immanants. This is then combined with technical results, which allows to obtain permanents or Hamilton cycle polynomials as projections of linear combinations of hook immanants.

2. Valiant's algebraic model of NP-completeness. We briefly recall the main features of Valiant's algebraic P-NP theory. For detailed expositions we refer to the survey by von zur Gathen [10] and the books by Bürgisser, Clausen, and Shokrollahi [7, Chapter 21] and Bürgisser [6].

We will adopt the following useful convention: we denote matrix functions with small letters, but the corresponding function evaluated at a matrix with distinct indeterminate entries is written in capitals. For instance, $\mathrm{per}(A)$ is the permanent of the n by n matrix A, and

$$\mathrm{PER}_n = \sum_{\pi\in S_n} \prod_{i=1}^{n} X_{i,\pi(i)}$$

denotes the permanent of an n by n matrix with indeterminate entries $X_{i,j}$. Likewise, IM_λ denotes the immanant polynomial corresponding to the diagram $\lambda \vdash n$.

Throughout the paper, the discussion will refer to a fixed field k of characteristic zero. (The reader may assume $k = \mathbb{Q}$.) By a *p-family* we understand a sequence (f_n) of multivariate polynomials $f_n \in k[X_1, \ldots, X_{v(n)}]$ such that the number of variables $v(n)$ as well as the degree $\deg f_n$ are *p-bounded* functions of n, i.e., these functions are majorized by a polynomial in n. Interesting examples are the *permanent family* $\mathrm{PER} = (\mathrm{PER}_n)$, the *determinant family* $\mathrm{DET} = (\mathrm{DET}_n)$, and the family $\mathrm{HC} = (\mathrm{HC}_n)$ of *Hamilton cycle polynomials* defined by

$$\mathrm{HC}_n = \sum_{\pi} \prod_{i=1}^{n} X_{i,\pi(i)},$$

where the sum is over all cycles $\pi \in S_n$ of length n. Note that the value of HC_n at the adjacency matrix of a digraph equals the number of its Hamilton cycles.

Let $L(f_n)$ denote the total *complexity* of $f_n \in k[X_1, \ldots, X_{v(n)}]$, that is, the minimum number of arithmetic operations $+, -, *$ to compute f_n from the variables X_i and constants in k by a straight-line program. We call a p-family *p-computable* iff $L(f_n)$ is p-bounded in n. The p-computable families constitute the complexity class VP.

A p-family (f_n) is called *p-definable* iff there exists a p-computable family (g_n), $g_n \in k[X_1, \ldots, X_{u(n)}]$, such that for all n

$$f_n(X_1, \ldots, X_{v(n)}) = \sum_{e \in \{0,1\}^{u(n)-v(n)}} g_n(X_1, \ldots, X_{v(n)}, e_{v(n)+1}, \ldots, e_{u(n)}).$$

The set of p-definable families form the complexity class VNP.

We will employ a very simple notion of reduction. A polynomial f_n is called a *projection* of a polynomial $g_m \in k[X_1, \ldots, X_u]$, written $f_n \leq g_m$, iff

$$f_n(X_1, \ldots, X_{v(n)}) = g_m(a_1, \ldots, a_u)$$

for some $a_i \in k \cup \{X_1, \ldots, X_{v(n)}\}$. That is, f_n can be derived from g_m through substitution by indeterminates and constants. A p-family (f_n) is called a *p-projection* of a family (g_m) iff there is a p-bounded function $t\colon \mathbb{N} \to \mathbb{N}$ such that f_n is a projection of $g_{t(n)}$ for all n. Finally, a p-definable family (g_m) is called *VNP-complete* iff any $(f_n) \in \mathrm{VNP}$ is a p-projection of (g_m).

In [22] Valiant proved that the p-families PER of permanents and HC of Hamilton cycles polynomials are VNP-complete (over fields k of characteristic different from two, which is a general assumption in this paper). Thus PER is not p-computable iff *Valiant's hypothesis* $\mathrm{VP} \neq \mathrm{VNP}$ is true.

One can prove that the "generating functions" corresponding to several NP-complete graph problems like cliques, graph factors, Hamilton cycles in planar graphs, etc. yield VNP-complete families as well (see [6, Chapter 3]). In fact, Valiant's hypothesis can be considered as a nonuniform algebraic counterpart of the well-known hypothesis $\mathrm{P} \neq \mathrm{NP}$ due to Cook [8]. For results relating these two hypotheses, see [4, 6]. We mention in passing that, by contrast with the classical P-NP theory, one knows specific p-definable families over finite fields, which are neither VNP-complete, nor p-computable, provided the polynomial hierarchy does not collapse (cf. [5, 6]).

For later use, we state some results going back to Valiant [24]; detailed proofs can be found in [6]. The first result shows that the complexity class VNP is closed under various natural operations.

PROPOSITION 2.1. *Let (f_n) and (g_n) be p-definable; say $f_n \in k[X_1, \ldots, X_{v(n)}]$. Then the following hold.*

(i) *Sum and product. $(f_n + g_n)$ and $(f_n \cdot g_n)$ are p-definable.*

(ii) *Substitution. $(f_n(g_1, \ldots, g_{v(n)}))$ is p-definable.*

(iii) *Coefficient. If $h_n \in k[X_{u(n)+1}, \ldots, X_{v(n)}]$ is the coefficient of some power product $X_1^{i_1} \cdots X_{u(n)}^{i_{u(n)}}$ in f_n for some $u(n) \leq v(n)$, then the family (h_n) is p-definable.*

The second result is a useful criterion for p-definability which connects the nonuniform counting complexity class #P/poly to the class VNP. Note that functions φ, which are computable in polynomial time on a Turing machine, are clearly contained in the class #P/poly. (For the definition of #P see [23]; a general definition of nonuniform complexity classes like #P/poly can be found in Karp and Lipton [15].)

PROPOSITION 2.2. *Suppose* $\varphi\colon \{0,1\}^* \to \mathbb{N}$ *is a function in the class* #P/poly. *Then the family* (f_n) *of polynomials defined by*

$$f_n = \sum_{e\in\{0,1\}^n} \varphi(e) X_1^{e_1} \cdots X_n^{e_n}$$

is p-definable.

3. Character formulas for the symmetric group. We first recall some character formulas for the symmetric group for later use. These formulas are then applied to derive Lemma 3.1, which is a crucial ingredient of our completeness proofs. More information on the characters of the symmetric groups can be found in the books by Boerner [2], James and Kerber [13], or Fulton and Harris [9].

We recall that $\lambda \vdash n$ means that $\lambda = (\lambda_1, \ldots, \lambda_s)$ is a partition of n. The irreducible character of S_n corresponding to λ is denoted by χ_λ.

To a partition λ we may assign the strictly decreasing sequence

$$\ell = [\ell_1, \ldots, \ell_s] := (\lambda_1, \ldots, \lambda_s) + (s-1, s-2, \ldots, 1, 0)$$

in $\mathbb{N}^s$ which satisfies $\sum \ell_i = n + \binom{s}{2}$. (We use square brackets to distinguish ℓ notationally from a partition λ.) It is useful to index the irreducible characters of S_n by such sequences, thus we set $\chi_\ell := \chi_\lambda$. We can extend this definition to any $\ell \in \mathbb{N}^s$ satisfying $\sum \ell_i = n + \binom{s}{2}$ by requiring the function $\ell \mapsto \chi_\ell$ to be alternating. In particular, χ_ℓ vanishes if two components of ℓ are equal. We also include the case $n = 0$ by setting $\chi_{[s-1,\ldots,0]}(1) := 1$.

Conjugacy classes of permutations in S_n are described by their *cycle format* $(\rho_1, \ldots, \rho_n)$, where ρ_i denotes the number of i-cycles. Clearly, $\sum_i i\rho_i = n$. It will be convenient to write cycle formats in frequency notation $\rho = 1^{\rho_1} \cdots n^{\rho_n}$, or shorter $\rho \models n$ in order to express that ρ is a cycle format of n. Moreover, we set $\chi_\ell(\rho) := \chi_\ell(\pi)$, where π is any permutation with cycle format ρ.

Let $u_{s,i} := Z_1^i + Z_2^i + \cdots + Z_s^i$ denote the ith elementary power sum in the indeterminates $Z_1, \ldots, Z_s$, and let

$$\Delta_s := \det(Z_i^{s-j})_{1\le i,j\le s} = \prod_{i<j}(Z_i - Z_j)$$

be the discriminant. The characters of S_n are determined by the remarkable *formula of Frobenius* (cf. [9, 4.10, p. 49]):

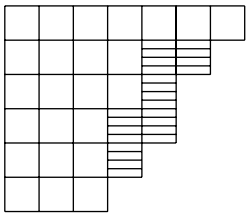
$$\Delta_s \cdot u_{s,1}^{\rho_1} \cdots u_{s,n}^{\rho_n} = \sum_\ell \chi_\ell(\rho) Z_1^{\ell_1} \cdots Z_s^{\ell_s}, \tag{3.1}$$

where the sum is over all $\ell \in \mathbb{N}^s$ satisfying $\sum \ell_i = n + \binom{s}{2}$. From this formula one easily deduces *Frobenius' recursion formula* for the characters of S_n (cf. [2, VI, section 3]): let $1 \le h \le n$ and $\rho \models n-h$. Then we have for all $\ell \in \mathbb{N}^s$ satisfying $\sum \ell_i = n + \binom{s}{2}$

$$\chi_\ell(\rho \cdot h) = \sum_i \chi_{[\ell_1,\ldots,\ell_{i-1},\ell_i-h,\ell_{i+1},\ldots,\ell_s]}(\rho), \tag{3.2}$$

where the sum is over all $1 \le i \le s$ such that $\ell_i \ge h$, and with the property that $\ell_1, \ldots, \ell_{i-1}, \ell_i - h, \ell_{i+1}, \ldots, \ell_s$ are pairwise distinct numbers.

Sometimes it is convenient to use a recursion formula related to (3.2), the so-called Murnaghan–Nakayama rule. We recall that a partition $\lambda \vdash n$ can be represented



$\lambda = (7, 6, 5, 5, 4, 3)$
$\mu = (7, 4, 4, 3, 3, 3)$
$r(\lambda, \mu) = 3$

FIG. 3.1. *A skew hook for the diagram of* λ.

by its *diagram* $\{(i, j) \mid 1 \leq j \leq \lambda_i\}$, which should be visualized as a left-justified arrangement of λ_i boxes in the ith row. A *skew hook* for λ is a connected region of boundary boxes for its diagram such that removing them leaves a diagram for another partition μ. We denote by $r(\lambda, \mu)$ the number of vertical steps in the skew hook, i.e., one less than the number of rows in the hook. For illustration see Figure 3.1.

The *Murnaghan–Nakayama* rule now reads as follows: For $\lambda \vdash n$, $1 \leq h \leq n$, and $\rho \models n - h$, we have

$$\chi_\lambda(\rho \cdot h) = \sum_\mu (-1)^{r(\lambda,\mu)} \chi_\mu(\rho), \tag{3.3}$$

where the sum is over all partitions $\mu \vdash n-h$ that can be obtained from λ by removing a skew hook containing h boxes (cf. [13, 2.4.7, p. 60] or [9, p. 59]).

As an example, let us compute the value of χ_λ on an n-cycle. The Murnaghan–Nakayama rule implies immediately that $\chi_\lambda(n^1) = (-1)^i$ if λ equals a hook partition $(n - i, 1, \ldots, 1)$, and that $\chi_\lambda(n^1) = 0$ otherwise.

For the rest of the paper, we will denote the characters corresponding to the hook partitions $(n - i, 1, \ldots, 1) \vdash n$ by $\chi_{n,i}$ and call them *hook characters*. The corresponding *hook immanant* polynomials will be denoted by $\mathrm{HI}_{n,i}$ for $0 \leq i < n$. Note that $\mathrm{HI}_{n,i}$ corresponds to a diagram of width $n - i$. For instance we have $\mathrm{HI}_{n,0} = \mathrm{PER}_n$ and $\mathrm{HI}_{n,n-1} = \mathrm{DET}_n$.

Now assume $\rho \models n$, $\rho \neq n^1$. The orthogonality relations (cf. [9, I, section 2.2]) and the above observation on the value of χ_λ on an n-cycle imply that

$$\sum_{\lambda \vdash n} \chi_\lambda(n^1)\chi_\lambda(\rho) = \sum_{i=0}^{n-1} (-1)^i \chi_{n,i}(\rho) = 0.$$

From this one easily concludes the following formula due to Merris [19]:

$$\sum_{i=0}^{n-1} (-1)^i \,\mathrm{HI}_{n,i} = n\mathrm{HC}_n. \tag{3.4}$$

We illustrate Frobenius' recursion formula (3.2) by computing the value $\chi_{(2,2,2,2)}$ for permutations in S_8 of cycle format 2^4. To the rectangular partition $(2, 2, 2, 2)$ there corresponds the sequence $\ell = [5, 4, 3, 2]$. The recursive application of (3.2) can be illustrated by the tree $T_{2,2}$ in Figure 3.2. This tree has depth 4 and its nodes carry a label $\ell \in \mathbb{N}^4$ having distinct components. The meaning of the tree $T_{2,2}$ is the following: Consider a node with label ℓ at level $1 \leq t \leq 4$, and let $\ell^{(1)}, \ldots, \ell^{(M)}$ be the labels of the sons of this node. Then we have by (3.2) that

$$\chi_\ell(2^t) = \sum_{i=1}^{M} \chi_{\ell^{(i)}}(2^{t-1}).$$

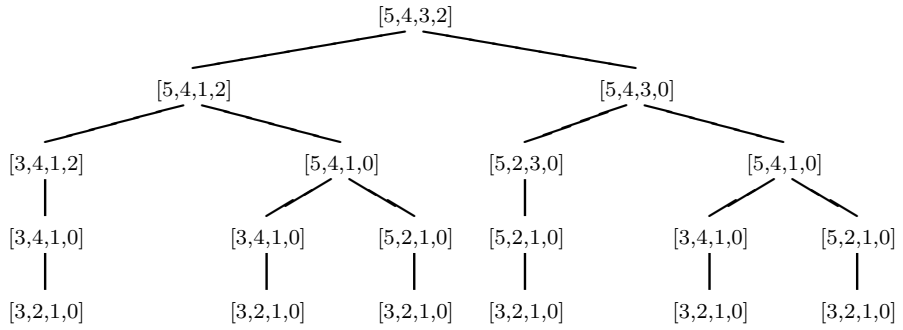


FIG. 3.2. *The tree $T_{2,2}$ illustrating the recursive application of Frobenius' recursion formula.*

From this it follows that $\chi_{[5,4,3,2]}(2^4)$ is the sum of $\chi_{[3,2,1,0]}(1)$ over all leaves of the tree. Hence $\chi_{[5,4,3,2]}(2^4) = 6\chi_{[3,2,1,0]}(1) = 6$.

To simplify notation, we will write χ_{m^s} for the character corresponding to a rectangular partition $(m, \ldots, m) \vdash sm$, and IM_{m^s} for the corresponding *rectangular immanant* polynomial.

The technical lemma below generalizes the above computational example and expresses certain values of χ_{m^s} by hook characters. The lemma will be crucial in our completeness proof in section 5.

LEMMA 3.1. *Let $s = qm + r$, $1 \le r \le m$, $q \ge 0$. Then we have for all cycle formats $\rho \models m$, $\rho \ne m^1$, that*

$$\chi_{m^s}(m^{s-1} \cdot \rho) = \gamma_{m,s} \sum_{i=0}^{r-1} (-1)^i \chi_{m,i}(\rho)$$

and

$$\chi_{m^s}(m^s) = r\gamma_{m,s} + m\beta_{m,s},$$

where

$$\gamma_{m,s} = \frac{(s-1)!}{q!^{m-r}(q+1)!^r}, \quad \beta_{m,s} = q\gamma_{m,s}.$$

Proof. As in the above example (Figure 3.2), we can describe the recursive application of Frobenius' recursion formula (3.2) for computing $\chi_{m^s}(m^s)$ by a labeled tree $T_{m,s}$. This tree is built up as follows. The root carries the label

$$\ell^{(0)} := [m+s-1, m+s-2, \ldots, m] = (m, m, \ldots, m) + (s-1, s-2, \ldots, 0).$$

To an already constructed node with label ℓ, we create a son for every $\ell - me_k$ which has nonnegative and distinct components. Here $e_k \in \{0,1\}^s$ denotes the canonical basis vector having a 1 at position $1 \le k \le s$.

We will soon prove that all leaves of $T_{m,s}$ carry the label $[s-1, \ldots, 1, 0]$. Using this already, we see that the nodes one level above the leaves carry a label of the form $L_k := [s-1, \ldots, 1, 0] + me_k$ for $1 \le k \le m$. Let α_k denote the number of these nodes. By a repeated application of the recursion formula (3.2), we obtain for all $\rho \models m$

$$\chi_{m^s}(m^{s-1} \cdot \rho) = \sum_{k=1}^{m} \alpha_k \chi_{L_k}(\rho).$$

A k-cycle transforms the sequence $(m-k+1,1,\dots,1,0,\dots,0)+(s-1,\dots,1,0)$ into L_k, and thus χ_{L_k} equals up to a sign a hook character: $\chi_{L_k}=(-1)^{k-1}\chi_{m,k-1}$. Now the point is that

$$\alpha_1=\cdots=\alpha_r,\ \alpha_{r+1}=\cdots=\alpha_m. \tag{3.5}$$

Assuming this for the moment, we may conclude for $\rho\models m$, $\rho\neq m^1$, that

$$\begin{aligned}\chi_{m^s}(m^{s-1}\cdot\rho)&=\alpha_1\sum_{i=0}^{r-1}(-1)^i\chi_{m,i}(\rho)+\alpha_{r+1}\sum_{i=r}^{m-1}(-1)^i\chi_{m,i}(\rho)\\&=(\alpha_1-\alpha_{r+1})\sum_{i=0}^{r-1}(-1)^i\chi_{m,i}(\rho),\end{aligned}$$

where we have used formula (3.4) in the last equality. We also get (recall that $\chi_{m,i}(m^1)=(-1)^i$)

$$\chi_{m^s}(m^s)=r\alpha_1+(m-r)\alpha_{r+1}=r(\alpha_1-\alpha_{r+1})+m\alpha_{r+1}.$$

So it remains to show (3.5) and to check that indeed $\alpha_1-\alpha_{r+1}=\gamma_{m,s}$ and $\alpha_{r+1}=q\gamma_{m,s}$.

To a leaf of $T_{m,s}$ there corresponds bijectively the path from the root to this leaf, which can be uniquely described by the sequence of labels $(\ell^{(0)},\dots,\ell^{(s)})$ of the nodes along this path. By assigning to $\ell^{(i)}$ the set $A_i\subseteq B:=\{0,1,\dots,m+s-1\}$ of its components, we get a sequence $(A_0,\dots,A_s)$ of subsets of B all having cardinality s, and which satisfy

$$A_{i+1}=(A_i\setminus\{a_i\})\cup\{a_i-m\} \tag{3.6}$$

with some $a_i\in A_i$ such that $a_i\geq m$ and $a_i-m\notin A_i$. It is easy to see that this correspondence is in fact a bijection between the paths of $T_{m,s}$ from the root to a leaf and the sequences $(A_0,\dots,A_s)$ of subsets of B satisfying (3.6) and such that $A_0=\{m,m+1,\dots,m+s-1\}$.

Now consider the complements $\overline{A_i}:=B\setminus A_i$. They are all of cardinality m. By induction on i one shows that the remainders modulo m of the elements of $\overline{A_i}$ are pairwise distinct. Hence we may write

$$\overline{A_i}=\{p_1^{(i)}m,p_2^{(i)}m+1,\dots,p_m^{(i)}m+m-1\} \tag{3.7}$$

with a uniquely determined vector $p^{(i)}=(p_1^{(i)},\dots,p_m^{(i)})$ contained in

$$W:=\{0,1,\dots,q+1\}^r\times\{0,1,\dots,q\}^{m-r}.$$

Let $a_i\in A_i$ be as in (3.6). As $a_i-m\notin A_i$, we have $a_i-m=p_{\mu_i}^{(i)}m+\mu_i-1$ for some $1\leq\mu_i\leq m$. On the other hand, $a_i=(p_{\mu_i}^{(i)}+1)m+\mu_i-1$ is by construction not contained in A_{i+1}. This implies that

$$p^{(i+1)}=p^{(i)}+e'_{\mu_i}, \tag{3.8}$$

where $e'_{\mu_i}\in\{0,1\}^m$ is the canonical basis vector having a 1 at position μ_i. We can thus regard $(p^{(0)},\dots,p^{(s)})$ as a walk in W which starts in $p^{(0)}=\underline{0}:=(0,\dots,0)$ and

must end in the opposite corner $p^{(s)} = w := (q+1, \ldots, q+1, q, \ldots, q)$. In this walk, a successor of a point is obtained by incrementing exactly one coordinate by 1.

It is now straightforward to check that we have found a bijection between the leaves of $T_{m,s}$ and the above-described walks in W from $\underline{0}$ to the opposite corner w. In particular, all leaves of $T_{m,s}$ carry the same label $[s-1, \ldots, 1, 0]$ corresponding to w, as was claimed at the beginning of the proof.

In what follows we will assume that $q \geq 1$. (The case $q = 0$ can be checked separately.) A node N of $T_{m,s}$ one level above the leaves corresponds to a walk in W ending in one of the points $w - e'_M$, where $1 \leq M \leq m$. To such a walk in turn there corresponds a sequence of sets $(A_0, \ldots, A_{s-1})$. Assume first that $1 \leq M \leq r$. Then it is easily checked that

$$A_{s-1} = (\{0, 1, \ldots, s-1\} \setminus \{qm + M - 1\}) \cup \{(q+1)m + M - 1\}.$$

By comparing this with the set of components of L_k, we obtain $qm + M - 1 = s - k$, and hence $k = r - M + 1$. We conclude that the node N carries the label L_{r-M+1}. The number of nodes of level $s-1$ carrying the label L_{r-M+1} equals the number of walks in W from $\underline{0}$ to $w - e'_M$. Such a walk is uniquely described (cf. (3.8)) by a sequence $(M_0, M_1, \ldots, M_{s-2})$ in $\{1, 2, \ldots, m\}^{s-1}$, in which $\mu \in \{M\} \cup \{r+1, \ldots, m\}$ occurs with frequency q, and $\mu \in \{1, \ldots, r\} \setminus \{M\}$ occurs with frequency $q+1$. The number of these sequences equals the multinomial coefficient

$$\alpha := \frac{(s-1)!}{q!^{m-r+1}(q+1)!^{r-1}}.$$

This proves that $\alpha_1 = \cdots = \alpha_r = \alpha$.

In the case $r < M \leq m$ one can show similarly that N carries the label $L_{r-M+m+1}$ and that the number of nodes carrying this label equals

$$\beta := \frac{(s-1)!}{(q-1)!\, q!^{m-r-1}(q+1)!^{r}},$$

which yields $\alpha_{r+1} = \cdots = \alpha_m = \beta$. A straightforward calculation shows that indeed $\alpha - \beta = \gamma_{m,s}$ and $\beta = q\gamma_{m,s}$, where $\gamma_{m,s} = (s-1)!/(q!^{m-r}(q+1)!^{r})$. □

4. P-definability of immanants. Frobenius' formula (3.1) for the generating function of S_n-characters allows us to express immanants as coefficients of p-computable families of polynomials. We will use this observation to prove the following proposition.

PROPOSITION 4.1. *Any family of immanant polynomials corresponding to a p-sequence of diagrams is p-definable.*

Proof. Let the *cycle format polynomial* of ρ be defined as $\mathrm{CF}_\rho := \sum_\pi \prod_{i=1}^n X_{i,\pi(i)}$, where the sum is over all permutations π having cycle format $\rho \models n$.

We multiply Frobenius' formula (3.1) for $s = n$ with CF_ρ and take the sum over all cycle formats $\rho \models n$. This yields

$$F_n := \Delta_n \sum_{\rho \models n} u_{n,1}^{\rho_1} \cdots u_{n,n}^{\rho_n} \mathrm{CF}_\rho = \sum_\ell \left(\sum_{\rho \models n} \chi_\ell(\rho)\, \mathrm{CF}_\rho \right) Z_1^{\ell_1} \cdots Z_n^{\ell_n}.$$

The immanent $\mathrm{IM}_\lambda = \sum_\rho \chi_\ell(\rho)\mathrm{CF}_\rho$ appears in F_n as the coefficient of the power product $\prod_j Z_j^{\ell_j} = \prod_j Z_j^{\lambda_j + n - j}$. By Proposition 2.1(iii) it is therefore sufficient to

prove that (F_n) is p-definable. Let $T_{i,j}$ be further indeterminates for $1 \le i,j \le n$, and define

$$G_n := \sum_{\rho \models n} \mathrm{CF}_\rho \prod_{(i,j):j \le \rho_i} T_{i,j}.$$

By substituting $T_{i,j}$ by $u_{n,i} = Z_1^i + \cdots + Z_n^i$ in G_n and multiplying the resulting polynomial with $\Delta_n = \prod_{i<j}(Z_i - Z_j)$, we obtain F_n. By Proposition 2.1(i)–(ii) it is therefore enough to show that the family (G_n) is p-definable. If we denote by $\rho_i(\pi)$ the number of i-cycles of π, we may write

$$\begin{aligned} G_n &= \sum_{\pi \in S_n} \left(\prod_{\alpha \le n} X_{\alpha,\pi(\alpha)} \right) \left(\prod_{(i,j):j \le \rho_i(\pi)} T_{i,j} \right) \\ &= \sum_{e,\epsilon \in \{0,1\}^{n \times n}} \varphi_n(e,\epsilon) \prod_{1 \le \alpha,\beta \le n} X_{\alpha,\beta}^{e_{\alpha,\beta}} \prod_{1 \le i,j \le n} T_{i,j}^{\epsilon_{i,j}} \end{aligned}$$

with a uniquely determined function

$$\varphi_n \colon \{0,1\}^{n \times n} \times \{0,1\}^{n \times n} \to \{0,1\}.$$

It is obvious that the extension of all φ_n to a function $\varphi \colon \{0,1\}^* \to \{0,1\}$ is computable by a polynomial time Turing machine. Therefore, Proposition 2.2 implies that (G_n) is p-definable. □

We can alternatively deduce Proposition 4.1 from the following interesting result due to Hepler [12], which shows that computing characters of the symmetric groups is a #P-complete problem. This result should be contrasted with Proposition 7.4 of [3], which shows that the evaluation of characters of GL_n is possible in polynomial time.

THEOREM 4.2. *The function which assigns to $\lambda \vdash n$, $\rho \models n$ the value $\chi_\lambda(\rho) + n^n$ is #P-complete (n given in unary).*

In fact, to deduce Proposition 4.1, we need only the easy part of this theorem stating that the above function is contained in #P. Let a p-sequence of diagrams $(\lambda^{(n)})$ be given. We can interpret $\lambda^{(n)}$ as a "polynomial advice" for n in the sense of Karp and Lipton [15]. By Theorem 4.2, the function which assigns to the description π of the power product $\prod_{i=1}^n X_{i,\pi(i)}$ its coefficient in the polynomial $\mathrm{IM}_{\lambda^{(n)}} + n^n \mathrm{PER}_n$ is in the class #P/poly. Therefore, by Proposition 2.2, the family $(\mathrm{IM}_{\lambda^{(n)}} + n^n \mathrm{PER}_n)_n$ is p-definable. This again implies Proposition 4.1.

5. Completeness proofs. This section is divided into five parts. We first describe the general strategy of our completeness proofs. Using this strategy and applying Lemma 3.1, we then provide a short proof of a special case of Theorem 1.2, namely for rectangular diagrams satisfying a certain divisibility condition.

In order to settle the general case of Theorem 1.2, we develop in subsection 5.3 technical results which allow us to obtain permanents or Hamilton cycle polynomials as projections of linear combinations of hook immanants. These technical results in combination with the general strategy then allow us to prove Theorem 1.2 for hook immanants. Finally, we are able to complete the proof of Theorem 1.2 for rectangular immanants.

5.1. The general proof strategy. We introduce some notation and work out the relationship between matrices and weighted digraphs.

There is a canonical bijective correspondence between square matrices and (edge) weighted digraphs. Namely, we can regard an n by n matrix $A = [a_{i,j}]$ over some field as the adjacency matrix of a weighted digraph G on n nodes, where $a_{i,j} \neq 0$ gives the weight of the edge from node i to node j, and there is no such edge if $a_{i,j} = 0$. We define the weight $\mathrm{wt}(\mathcal{C})$ of a subgraph $\mathcal{C}$ of G as the product of the weights of all its edges. The permutations $\pi \in S_n$ with $\prod_{i=1}^n a_{i,\pi(i)} \neq 0$ correspond bijectively to the *cycle covers* $\mathcal{C}$ of G, i.e., n-node subgraphs of G consisting of node-disjoint cycles. Note that $\mathrm{wt}(\mathcal{C}) = \prod_i a_{i,\pi(i)}$ if $\mathcal{C}$ corresponds to π. A cycle cover $\mathcal{C}$ has a (cycle) format $\rho \models n$, where ρ_i counts the number of i-cycles of $\mathcal{C}$.

We define now the *cycle format value* $\mathrm{cf}_\rho(G)$ of a weighted digraph G as the sum of the weights of all cycle covers of G having the format ρ. A special case of this is the *Hamilton cycle value* $\mathrm{hc}(G) := \mathrm{cf}_n(G)$. (Recall our convention on notations at the beginning of section 2.) The *immanant* $\mathrm{im}_\lambda(G)$ of the weighted digraph G is defined as

$$\mathrm{im}_\lambda(G) := \sum_{\rho \models n} \chi_\lambda(\rho) \mathrm{cf}_\rho(G).$$

If all the weights of G are either rationals, constants, or indeterminates, then $\mathrm{im}_\lambda(G)$ is obviously a projection of IM_λ. Moreover, if DK_n denotes the complete weighted digraph DK_n on n nodes with edge (i,j) carrying the indeterminate weight $X_{i,j}$, then $\mathrm{im}_\lambda(\mathrm{DK}_n) = \mathrm{IM}_\lambda$. Similarly, we define the *cycle format polynomial* by setting $\mathrm{CF}_\rho := \mathrm{cf}_\rho(\mathrm{DK}_n)$. Finally, we will denote the value of the *hook immanant* $\mathrm{HI}_{n,i}$ on G by $\mathrm{hi}_{n,i}(G)$.

We explain now our general strategy for proving completeness for a family of immanants. Let $1 \leq h \leq n$, and let G be the disjoint union of the complete weighted digraph DK_{n-h} on $n-h$ nodes with a directed cycle Z of length h, all of whose edges carry the weight 1. Every cycle cover $\mathcal{C}$ of G consists of a cycle cover $\mathcal{C}'$ of DK_{n-h} and the h-cycle Z, and we have $\mathrm{wt}(\mathcal{C}) = \mathrm{wt}(\mathcal{C}')$. Therefore, we obtain for $\rho \models n-h$

$$\mathrm{cf}_{\rho \cdot h^1}(G) = \mathrm{cf}_\rho(DK_{n-h}) = \mathrm{CF}_\rho,$$

and all other cycle format values of G vanish. From this and the Murnaghan–Nakayama rule (3.3), we obtain

$$\begin{aligned}\mathrm{im}_\lambda(G) &= \sum_{\rho \models n-h} \chi_\lambda(\rho \cdot h^1)\, \mathrm{CF}_\rho = \sum_{\rho \models n-h} \sum_{\mu} (-1)^{r(\lambda,\mu)} \chi_\mu(\rho)\, \mathrm{CF}_\rho \\ &= \sum_{\mu} (-1)^{r(\lambda,\mu)} \sum_{\rho \models n-h} \chi_\mu(\rho)\, \mathrm{CF}_\rho = \sum_{\mu} (-1)^{r(\lambda,\mu)} \mathrm{IM}_\mu.\end{aligned}$$

Let us summarize this important insight in slightly more general form.

LEMMA 5.1. *Let $\lambda^{(1)}, \ldots, \lambda^{(t)} \vdash n$, $\alpha_1, \ldots, \alpha_t \in \mathbb{Q}$, and $1 \leq h \leq n$. Then the linear combination of immanants $\sum_{i=1}^t \alpha_i \mathrm{IM}_{\lambda^{(i)}}$ has as a projection the linear combination of immanants*

$$\sum_{i=1}^{t} \alpha_i \sum_{\mu^{(i)}} (-1)^{r(\lambda^{(i)},\mu^{(i)})} \mathrm{IM}_{\mu^{(i)}},$$

where $\mu^{(i)}$ runs over all partitions $\mu^{(i)} \vdash n-h$ which can be obtained from $\lambda^{(i)}$ by removing a skew hook containing h boxes.

5.2. Completeness of particular rectangular immanants. We present here the proof of a special case of Theorem 1.2, namely for rectangular immanants satisfying a certain divisibility condition.

A rectangular diagram $(m,\dots,m) \vdash sm$ is said to be of *height* s.

PROPOSITION 5.2. *Take a sequence of rectangular diagrams $(\lambda^{(m)})$ of polynomially growing width such that the width of $\lambda^{(m)}$ is a divisor of the height of $\lambda^{(m)}$ for all m. Then the corresponding family of rectangular immanants is VNP-complete.*

Proof. By Proposition 4.1 we know already that the given family is p-definable, so it suffices to show that the family of Hamilton cycle polynomials is a p-projection of the given family.

Let us write $\lambda^{(m)} = (m,\dots,m) \vdash s_m m$ and $s_m = q_m m + r_m$ with $1 \le r_m \le m$, $q_m \ge 0$. The height s_m is p-bounded in m since $(\lambda^{(m)})$ is a p-sequence of diagrams. We will use the divisibility assumption $r_m = m$ only at the end of the proof, in order to make the following reasonings reusable.

Let G_m be the disjoint union of the complete weighted digraph DK_m with $s_m - 1$ directed cycles of length m, all of whose edges having weight 1. By applying our general strategy explained in section 5.1, we obtain that

$$f_m := \mathrm{im}_{m^{s_m}}(G) = \sum_{\rho \models m} \chi_{m^{s_m}}(m^{s_m-1} \cdot \rho)\,\mathrm{CF}_\rho$$

is a projection of $\mathrm{IM}_{m^{s_m}}$. Lemma 3.1 implies that

$$\begin{aligned} f_m &= (r_m\gamma_{m,s_m} + m\beta_{m,s_m})\mathrm{CF}_m + \sum_{\substack{\rho\models m\\ \rho\neq m}} \gamma_{m,s_m} \sum_{i=0}^{r_m-1} (-1)^i \chi_{m,i}(\rho)\,\mathrm{CF}_\rho \\ &= m\beta_{m,s_m}\mathrm{CF}_m + \sum_{\rho\models m} \gamma_{m,s_m} \sum_{i=0}^{r_m-1} (-1)^i \chi_{m,i}(\rho)\,\mathrm{CF}_\rho. \end{aligned}$$

Therefore, we get

$$f_m = m\beta_{m,s_m}\mathrm{HC}_m + \gamma_{m,s_m} \sum_{i=0}^{r_m-1} (-1)^i\,\mathrm{HI}_{m,i}. \tag{5.1}$$

Since we assume that $r_m = m$, we conclude with formula (3.4) that f_m is a nonzero scalar multiple of HC_m. This shows that HC is a p-projection of $\mathrm{IM}_{m^{s_m}}$ and proves our proposition. □

5.3. Projections of linear combinations of hook immanants. In order to settle the general case of Theorem 1.2, we have to develop some technical results which allow us to obtain permanents or Hamilton cycle polynomials as projections of linear combinations of hook immanants.

The following lemma is proved similarly to Theorem 2 in Hartmann [11].

LEMMA 5.3.

(i) *Let $0 \le r < n$ and $\alpha_0,\dots,\alpha_r,\beta \in \mathbb{Q}$. Put $p := \lfloor n/(r+1) \rfloor$ and write $\gamma := \sum_{i=0}^{r}(-1)^i\alpha_i$. Then $\beta\mathrm{HC}_p + \gamma\mathrm{PER}_p$ is a projection of $\beta\mathrm{HC}_n + \sum_{i=0}^{r}\alpha_i\mathrm{HI}_{n,i}$.*

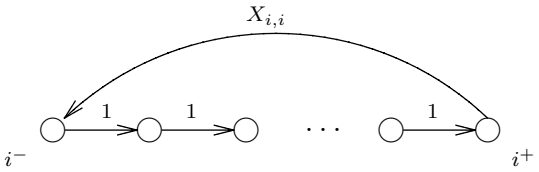
(ii) *Let $1 \le r < n$ and $\alpha_0,\alpha_r,\dots,\alpha_{n-1},\beta \in \mathbb{Q}$. Put*

$$p := \Big\lfloor \frac{n}{n-r+\delta} \Big\rfloor,\ R := n - p(n-r+\delta),\ \gamma := (-1)^{n+R-1} \sum_{i=r}^{n-1} (-1)^i\alpha_i,$$

where $\delta \in \{0,1\}$, $\delta \equiv n-r+1 \bmod 2$. *Then* $\alpha_0\mathrm{PER}_p+\beta\mathrm{HC}_p+\gamma\mathrm{DET}_p$ *is a projection of* $\alpha_0\mathrm{PER}_n + \beta\mathrm{HC}_n + \sum_{i=r}^{n-1} \alpha_i \mathrm{HI}_{n,i}$.

Proof. We note first that by the Murnaghan–Nakayama rule we have for the hook characters $\chi_{n,i}$ that $\chi_{n,i}(\rho) = (-1)^i$ if the cycle format ρ does not contain cycles of length at most i. This observation is crucial for the proof.

(i) For each $i \in \{2,\dots,p\}$ we introduce a cycle of length $r+1$ with edge weights as indicated in the following figure.



Analogously, we introduce a cycle of length $n-(p-1)(r+1)$ for $i=1$. Moreover, we connect the node i^+ with the node j^- by a (directed) edge of weight $X_{i,j}$ for all distinct $i,j \in \{1,2,\dots,p\}$. The resulting weighted digraph G has n nodes and girth $\geq r+1$; that is, all cycles of G have length at least $r+1$. A cycle cover $\tilde{\mathcal{C}}$ of G corresponds bijectively to a cycle cover $\mathcal{C}$ of the complete weighted digraph DK_p. Moreover, $\mathrm{wt}(\tilde{\mathcal{C}}) = \mathrm{wt}(\mathcal{C})$, and $\tilde{\mathcal{C}}$ is a Hamilton cycle iff $\mathcal{C}$ is so.

By the observation at the beginning of the proof we have $\chi_{n,i}(\tilde{\mathcal{C}}) = (-1)^i$ for all $0 \leq i \leq r$ and cycle covers $\tilde{\mathcal{C}}$ of G (which we identify with permutations in S_n). We get therefore for $0 \leq i \leq r$

$$\mathrm{hi}_{n,i}(G) = \sum_{\mathcal{C}} \chi_{n,i}(\tilde{\mathcal{C}})\mathrm{wt}(\tilde{\mathcal{C}}) = (-1)^i \sum_{\mathcal{C}} \mathrm{wt}(\mathcal{C}) = (-1)^i\,\mathrm{PER}_p,$$

and $\mathrm{hc}(G) = \mathrm{HC}_p$; hence

$$\left(\beta\,\mathrm{hc} + \sum_{i=0}^{r} \alpha_i \mathrm{hi}_{n,i}\right)(G) = \beta\,\mathrm{HC}_p + \gamma\mathrm{PER}_p.$$

This proves statement (i).

(ii) We assume that $n-r$ is odd; thus $\delta = 0$. (In the case where $n-r$ is even one can argue analogously.) As in the proof of (i) we construct a weighted digraph G by introducing for each $i \in \{2,\dots,p\}$ a cycle of length $n-r$, and for $i=1$ a cycle of length $n-(p-1)(n-r) = n-r+R$. Again, a cycle cover $\tilde{\mathcal{C}}$ of G corresponds bijectively to a cycle cover $\mathcal{C}$ of DK_p. Note that an ℓ-cycle of DK_p which does not pass through 1 is assigned to a cycle of G having length $\ell(n-r)$, and that $\ell \equiv \ell(n-r) \bmod 2$. To an ℓ-cycle of DK_p passing through 1 there corresponds a cycle of DK_p having length $\ell(n-r)+R$, which is congruent to $\ell + R$ modulo 2. From this we see that for any cycle cover $\mathcal{C}$ of DK_p

$$\mathrm{sgn}(\tilde{\mathcal{C}}) = (-1)^R \mathrm{sgn}(\mathcal{C}).$$

By interchanging rows and columns in the hook diagram $(n-i,1,\dots,1)$ we get the diagram of $(i+1,1,\dots,1)$. Therefore, the corresponding characters are conjugated:

$$\chi_{n,i}(\tilde{\mathcal{C}}) = \mathrm{sgn}(\tilde{\mathcal{C}})\chi_{n,n-i-1}(\tilde{\mathcal{C}})$$

(cf. [9, Ex. 4.4, p. 47]). On the other hand, we have for all $r \leq i < n$ and cycle covers $\tilde{\mathcal{C}}$ that $\chi_{n,n-i-1}(\tilde{\mathcal{C}}) = (-1)^{n-i-1}$, since $\mathrm{girth}(G) \geq n-r$. From these observations we

conclude that for all $r \le i < n$

$$\begin{aligned}\mathrm{hi}_{n,i}(G) &= \sum_{\mathcal{C}} \chi_{n,i}(\tilde{\mathcal{C}})\mathrm{wt}(\tilde{\mathcal{C}}) \\ &= \sum_{\mathcal{C}} (-1)^R \mathrm{sgn}(\mathcal{C})\chi_{n,n-i-1}(\tilde{\mathcal{C}})\mathrm{wt}(\mathcal{C}) \\ &= (-1)^{n+R-1}(-1)^i \sum_{\mathcal{C}} \mathrm{sgn}(\mathcal{C})\mathrm{wt}(\mathcal{C}) \\ &= (-1)^{n+R-1}(-1)^i \,\mathrm{DET}_p.\end{aligned}$$

Taking into account that $\mathrm{hc}(G) = \mathrm{HC}_p$ and $\mathrm{per}(G) = \mathrm{PER}_p$, the claim follows now readily. □

LEMMA 5.4. *Let $\alpha, \beta \in \mathbb{Q}$, $\alpha \neq 0$. Then the following hold.*

(i) PER_{n-1} *is a projection of* $\alpha\mathrm{PER}_n + \beta\mathrm{DET}_n$.

(ii) HC_{n-2} *is a projection of* $\alpha\mathrm{HC}_n + \beta\mathrm{DET}_n$.

Proof. (i) We concatenate the matrix $[X_{i,j}]_{1\le i,j<n}$ of indeterminates with the column $[0,\ldots,0,(2\alpha)^{-1}]^T$ and repeat in the resulting matrix the last row. In this way we get an n by n matrix M having determinant zero. By expanding along the last column we get $\mathrm{per}(M) = 2(2\alpha)^{-1}\mathrm{per}([X_{i,j}]_{i,j<n}) = \alpha^{-1}\mathrm{PER}_{n-1}$. Hence $(\alpha\,\mathrm{per} + \beta\,\det)(M) = \mathrm{PER}_{n-1}$.

(ii) Consider the n by n matrix

$$M := \begin{bmatrix} 0 & 0 & 1 & X_{1,2} & \ldots & X_{1,n} \\ \alpha^{-1} & 0 & 0 & 0 & \ldots & 0 \\ 0 & 0 & 1 & X_{1,2} & \ldots & X_{1,n} \\ 0 & X_{2,1} & 0 & X_{2,2} & \ldots & X_{2,n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & X_{n,1} & 0 & X_{n,2} & \ldots & X_{n,n} \end{bmatrix}.$$

As the first and third row of M coincide, we have $\det(M) = 0$. We claim that $\mathrm{hc}(M) = \alpha^{-1}\mathrm{HC}_{n-2}$. This is best seen by considering the weighted digraph with adjacency matrix M which is built up as follows from DK_{n-2}. Assume that $\{1,2,\ldots,n-2\}$ is the set of nodes of DK_{n-2}. We delete the loop $(1,1)$ and split the node 1 into nodes 1^- and 1^+. The edges $(i,1)$ of DK_{n-2} are thus replaced by edges $(i,1^-)$, and the edges $(1,i)$ of DK_{n-2} are replaced by edges $(1^+,i)$ for $2 \le i \le n$. Now we introduce a new node O, an edge $(1^-,O)$ of weight α^{-1}, an edge $(O,1^+)$ of weight 1, a loop $(1^+,1^+)$ of weight 1, and edges (O,i) of weight $X_{1,i}$ for $2 \le i \le n$. The reader may easily verify that the resulting weighted digraph G has indeed the adjacency matrix M. On the other hand, it is clear from the construction that every Hamilton cycle of G passes through the edges $(1^-,O)$ and $(O,1^+)$. This shows that indeed $\mathrm{hc}(G) = \alpha^{-1}\mathrm{hc}(\mathrm{DK}_{n-2}) = \alpha^{-1}\mathrm{HC}_{n-2}$. Altogether, we have $(\alpha\,\mathrm{hc} + \beta\,\det)(G) = \mathrm{HC}_{n-2}$, which proves the claim. □

We call a p-family (f_n) *monotone* iff f_n is a projection of f_{n+1} for all n. The above lemma in particular shows that PER is monotone. Also, HC is monotone, which can be demonstrated similarly as part (ii) of the above lemma.

5.4. Completeness of hook immanants. The technical results of the previous section in combination with our general strategy allow us to provide the proof of Theorem 1.2 for hook immanants, stated explicitly below.

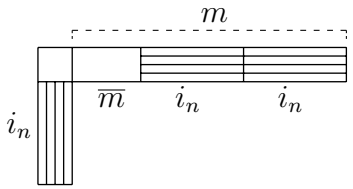


FIG. 5.1. *The hook diagram* $(n - i_n, 1, \ldots, 1)$ *when* $q = 2$.

PROPOSITION 5.5. *Any family of hook immanants of polynomially growing width is VNP-complete.*

Proof. The following notation will be useful: we write $\varphi(n) \gtrsim \psi(n)$ for functions $\varphi, \psi \colon \mathbb{N} \to (0, \infty)$ iff $\liminf_{n\to\infty} \varphi(n)/\psi(n) \geq 1$.

Let $\epsilon > 0$, and let (i_n) be a sequence of natural numbers satisfying $n - i_n \geq n^\epsilon$. We have to prove that (HI_{n,i_n}) is VNP-complete. As we already know from Proposition 4.1 that (HI_{n,i_n}) is p-definable, it is sufficient to prove that PER is a p-projection of this p-family. We distinguish several cases.

Case 1. $i_n \leq \sqrt{n}$.

Let $p_n := \lfloor \frac{n}{i_n+1} \rfloor$. Then $p_n \gtrsim \sqrt{n}$ and Lemma 5.3(i) implies that $(-1)^{i_n}\mathrm{PER}_{p_n}$ is a projection of HI_{n,i_n}.

Case 2. $i_n > \sqrt{n}$.

Put $m := n - i_n - 1$ and let $m = qi_n + \overline{m}$, where $0 \leq \overline{m} < i_n$. By applying Lemma 5.1 with $h = i_n$ we obtain that $f_1 := \mathrm{HI}_{n-i_n,i_n} + (-1)^{i_n-1}\mathrm{HI}_{n-i_n,0}$ is a projection of HI_{n,i_n} (cf. Figure 5.1). Again applying this lemma with $h = i_n$ yields that

$$f_2 := \mathrm{HI}_{n-2i_n,i_n} + (-1)^{i_n-1}\mathrm{HI}_{n-2i_n,0} + (-1)^{i_n-1}\mathrm{HI}_{n-2i_n,0}$$

is a projection of f_1. If we continue in this way, we see that with $\overline{n} := n - qi_n = \overline{m} + i_n + 1$, the polynomial

$$f_q := \mathrm{HI}_{\overline{n},i_n} + q(-1)^{i_n-1}\mathrm{PER}_{\overline{n}}$$

is a projection of HI_{n,i_n}.

Subcase 2.1. $\overline{m} \geq n^{1/4}$.

We apply Lemma 5.1 with $h = i_n$ to f_q and get that $(q+1)(-1)^{i_n-1}\mathrm{PER}_{\overline{m}+1}$ is a projection of HI_{n,i_n}. (Recall that $\overline{m} < i_n$.) Note that the coefficient $(q+1)(-1)^{i_n-1}$ is nonzero!

Subcase 2.2. $\overline{m} < n^{1/4}$ and $q \geq 1$.

We apply Lemma 5.3 (ii) to f_q and obtain

$$g_n := \gamma_n \mathrm{DET}_{p_n} + q(-1)^{i_n-1}\mathrm{PER}_{p_n}$$

as a projection of f_q for some $\gamma_n \in \{-1, 1\}$ and some

$$p_n \geq \left\lfloor \frac{\overline{n}}{\overline{n} - i_n + 1} \right\rfloor \geq \left\lfloor \frac{i_n}{\overline{m} + 2} \right\rfloor \gtrsim n^{1/4}.$$

By invoking Lemma 5.4(i), we see that PER_{p_n-1} is a projection of g_n and thus of HI_{n,i_n}.

Subcase 2.3. $q = 0$.

As in subcase 2.1, we apply Lemma 5.1 with $h = i_n$ to $f_q = \mathrm{HI}_{n,i_n}$ and get that $(-1)^{i_n-1}\mathrm{PER}_{\overline{m}+1}$ is a projection of HI_{n,i_n}. But $\overline{m}+1 = n - i_n \geq n^\epsilon$ by assumption.

To summarize, let $\delta := \min\{\epsilon, 1/4\}$. We have shown the existence of a sequence $N_n \gtrsim n^\delta$ of natural numbers such that PER_{N_n} is a projection of HI_{n,i_n} for all sufficiently large n. Taking into account that PER is monotone, this implies that PER is a p-projection of $(\mathrm{HI}_{n,i_n})_n$ and finishes the proof. ∎

5.5. Completeness of rectangular immanants. The proof of Theorem 1.2 will be achieved by identifying either a hook immanant or a Hamilton cycle polynomial as a projection of a rectangular immanant. This will be sufficient to establish completeness according to the following little observation.

LEMMA 5.6. *Let $f = (f_n)$ and $g = (g_n)$ be VNP-complete families and assume f to be monotone. Moreover, let $I \subseteq \mathbb{N}$ and define the corresponding* mixture $h = (h_n)$ *of f and g as*

$$h_n := \begin{cases} f_n & \text{if } n \in I, \\ g_n & \text{otherwise.} \end{cases}$$

Then h is VNP-complete as well.

Proof. The p-family $(f_1, g_1, f_2, g_2, f_3, g_3, \dots)$ is obviously p-definable and h is a p-projection of it; hence h is also p-definable.

Let $\varphi = (\varphi_n)$ be any p-definable family. Since f is complete, there is a p-bounded function $n \mapsto t_1(n)$ such that φ_n is a projection of $f_{t_1(n)}$: $\varphi_n \leq f_{t_1(n)}$ for all n. We put $D(n) := \max\{\deg g_i \mid 1 \leq i \leq n\}$ and consider the p-definable family

$$\Phi_n := \varphi_n + Z^{1+D(t_1(n))},$$

where Z is a new variable. As g is complete, there is a p-bounded $n \mapsto t_2(n)$ such that $\Phi_n \leq g_{t_2(n)}$ for all n. In particular,

$$D(t_1(n)) < \deg \Phi_n \leq \deg g_{t_2(n)},$$

which implies $t_2(n) > t_1(n)$. By substituting $Z \mapsto 0$ we see that $\varphi_n \leq g_{t_2(n)}$. On the other hand, we also have $\varphi_n \leq f_{t_2(n)}$, as f is monotone. From this it immediately follows that $\varphi_n \leq h_{t_2(n)}$ for all n. This shows that h is a complete family. ∎

We remark that the monotonicity assumption is necessary. Namely, if (f_n) is complete, then the families $(f_1, 0, f_2, 0, \dots)$ and $(0, f_1, 0, f_2, \dots)$ are both complete as well, but their mixture with respect to $I = \{2, 4, 6, 8, \dots\}$ equals the zero sequence.

Finally, we present the proof of Theorem 1.2 in the general situation.

Proof. Suppose that (s_m) is a p-bounded sequence of natural numbers. We wish to show that the sequence of rectangular immanants $\mathrm{IM}_{m^{s_m}}$ is VNP-complete. We write $s_m = q_m m + r_m$ with $1 \leq r_m \leq m$, $q_m \geq 0$.

As in the proof of Proposition 5.2, (5.1), we find that

$$f_m = m\beta_{m,s_m}\mathrm{HC}_m + \gamma_{m,s_m} \sum_{i=0}^{r_m-1} (-1)^i \,\mathrm{HI}_{m,i}$$

is a projection of $\mathrm{IM}_{m^{s_m}}$.

We distinguish now two cases.

Case 1. $1 \leq r_m \leq m - \sqrt{m}$.

We apply Lemma 5.1 with $h = 1$ and obtain that

$$g_m := \gamma_{m,s_m}\left(\mathrm{HI}_{m-1,0} + \sum_{i=1}^{r_m-1}(-1)^i\left[\mathrm{HI}_{m-1,i} + \mathrm{HI}_{m-1,i-1}\right]\right)$$

is a projection of $\gamma_{m,s_m}\sum_{i=0}^{r_m-1}(-1)^i\,\mathrm{HI}_{m,i}$. Recall that this is shown by adding to DK_{m-1} an isolated vertex with a loop (of weight 1). The resulting digraph does not have a Hamilton cycle. From this one easily sees that g_m is also a projection of f_m. The formula for g_m simplifies to (telescoping sum)

$$g_m = \gamma_{m,s_m}(-1)^{r_m-1}\mathrm{HI}_{m-1,r_m-1}.$$

If $r_m > m - \sqrt{m}$, then we define $g_m := \mathrm{HI}_{m-1,0}$. The family (g_m) of hook immanants is complete by Proposition 5.5.

Case 2. $m - \sqrt{m} < r_m \leq m$.

By using relation (3.4) we can rewrite f_m as follows:

$$\begin{aligned} f_m &= m\beta_{m,s_m}\mathrm{HC}_m + \gamma_{m,s_m}\left(m\mathrm{HC}_m - \sum_{i=r_m}^{m-1}(-1)^i\,\mathrm{HI}_{m,i}\right)\\ &= \kappa_m\mathrm{HC}_m + \gamma_{m,s_m}\sum_{i=r_m}^{m-1}(-1)^{i+1}\mathrm{HI}_{m,i}, \end{aligned}$$

where $\kappa_m := m(\beta_{m,s_m} + \gamma_{m,s_m}) > 0$. Lemma 5.3(ii) shows that

$$\varphi_m := \kappa_m\mathrm{HC}_{p_m} + c_m\mathrm{DET}_{p_m}$$

is a projection of f_m for some $c_m \in \mathbb{Q}$ and some

$$p_m \geq \left\lfloor\frac{m}{m - r_m + 1}\right\rfloor \geq \left\lfloor\frac{m}{\sqrt{m}+1}\right\rfloor \geq \lfloor\sqrt{m}\rfloor - 1.$$

Lemma 5.4(ii) together with the fact that HC is monotone implies that the Hamilton cycle polynomial $h_m := \mathrm{HC}_{\lfloor\sqrt{m}\rfloor-3}$ is a projection of φ_m and thus of f_m. The family (h_m) is monotone and complete, as HC has these properties.

To summarize, we have proved that some mixture of the families (g_m) of hook immanants and (h_m) of Hamilton cycle polynomials is a p-projection of (f_m), and thus of $(\mathrm{IM}_{m^{s_m}})$. Hence the family of rectangular immanants $(\mathrm{IM}_{m^{s_m}})$ is complete by Lemma 5.6. □

Finally, we remark that in order to obtain Corollary 1.4 from Theorem 1.2, one can check that the projections $f_n \leq g_m$ occurring in the proof of this theorem actually yield relations $N_n f_n(X) = g_m(a)$, where the components of a are either indeterminates, 0, or 1, and the factor N_n is a nonzero integer. Moreover, m, N_n, and a are computable in polynomial time from n. Thus we get "weakly parsimonious"reductions (cf. [14, p. 107]) between the corresponding problems to evaluate f_n, g_m at $0,1$-values. Moreover, one can obtain from Theorem 4.2 that the problem to compute $\mathrm{im}_{\lambda^{(n)}}(A) + n^{2n}$ from $A \in \{0,1\}^{n\times n}$ is in contained in #P for any polynomial time computable map $n \mapsto \lambda^{(n)}$.

Acknowledgment. Thanks go to M. Clausen for valuable discussions.

REFERENCES

[1] A. Barvinok, *Computational complexity of immanants and representations of the full linear group*, Funct. Anal. Appl., 24 (1990), pp. 144–145.
[2] H. Boerner, *Representations of Groups*, Elsevier–North Holland, Amsterdam, 1970.
[3] P. Bürgisser, *The computational complexity to evaluate representations of general linear groups*, SIAM J. Comput., 30 (2000), pp. 1010–1022.
[4] P. Bürgisser, *Cook's versus Valiant's hypothesis*, Theoret. Comput. Sci., 235 (2000), pp. 71–88.
[5] P. Bürgisser, *On the structure of Valiant's complexity classes*, Discrete Math. Theor. Comput. Sci., 3 (1999), pp. 73–94.
[6] P. Bürgisser, *Completeness and Reduction in Algebraic Complexity Theory*, Algorithms Comput. Math. 7, Springer Verlag, Berlin, 2000.
[7] P. Bürgisser, M. Clausen, and M. Shokrollahi, *Algebraic Complexity Theory*, Grundlehren Math. Wiss. 315, Springer Verlag, New York, 1997.
[8] S. Cook, *The complexity of theorem proving procedures*, in Proceedings of the 3rd ACM Symposium on Theory of Computing, Shaker Heights OH, 1971, pp. 151–158.
[9] W. Fulton and J. Harris, *Representation Theory*, Grad. Texts in Math. 129, Springer Verlag, New York, 1991.
[10] J. v. z. Gathen, *Feasible arithmetic computations: Valiant's hypothesis*, J. Symbolic Comput., 4 (1987), pp. 137–172.
[11] W. Hartmann, *On the complexity of immanants*, Linear and Multilinear Algebra, 18 (1985), pp. 127–140.
[12] C. Hepler, *On the Complexity of Computing Characters of Finite Groups*, Tech. Report 94/545/14, Dept. of Computer Science, University of Calgary, Canada, 1994.
[13] G. James and A. Kerber, *The Representation Theory of the Symmetric Group*, Addison-Wesley, Reading, MA, 1981.
[14] D. Johnson, *A catalog of complexity classes*, in Handbook of Theoretical Computer Science A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 61–161.
[15] R. Karp and R. Lipton, *Turing machines that take advice*, in Logic and Algorithmic: An International Symposium Held in Honor of Ernst Specker, Monographies de L'Enseignement Mathématique 30, L'Enseignement Mathématique, Geneva, 1982, pp. 255–273.
[16] D. Littlewood, *The Theory of Group Characters and Matrix Representations of Groups*, Oxford University Press, Oxford, UK, 1940.
[17] R. Merris, *On vanishing decomposable symmetrized tensors*, Linear and Multilinear Algebra, 5 (1977), pp. 79–86.
[18] R. Merris, *Recent advances in symmetry classes of tensors*, Linear and Multilinear Algebra, 7 (1979), pp. 317–328.
[19] R. Merris, *Single-hook characters and Hamiltonian circuits*, Linear and Multilinear Algebra, 14 (1983), pp. 21–35.
[20] H. J. Ryser, *Combinatorial Mathematics*, Carus Math. Monogr. 14, Math. Assoc. America, New York, 1963.
[21] V. Strassen, *Algebraic complexity theory*, in Handbook of Theoretical Computer Science A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 634–672.
[22] L. Valiant, *Completeness classes in algebra*, in Proceedings of the 11th ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 249–261.
[23] L. Valiant, *The complexity of computing the permanent*, Theoret. Comput. Sci., 8 (1979), pp. 189–201.
[24] L. Valiant, *Reducibility by algebraic projections*, in Logic and Algorithmic: An International Symposium Held in Honor of Ernst Specker, Monographies de L'Enseignement Mathématique 30, L'Enseignement Mathématique, Geneva, 1982, pp. 365–380.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1041–1066

AN ALGORITHMIC REGULARITY LEMMA FOR HYPERGRAPHS*

ANDRZEJ CZYGRINOW† AND VOJTECH RÖDL‡

Abstract. In this paper, we will consider the problem of designing an efficient algorithm that finds an ϵ-regular partition of an l-uniform hypergraph.

Key words. hypergraphs, regularity lemma, algorithms

AMS subject classifications. 05C65, 68R05

PII. S0097539799351729

1. Introduction. The regularity lemma of Szemerédi [12] is a powerful tool used in extremal combinatorics and graph theory. The lemma states that the vertex set of any graph can be partitioned into subsets that meet certain regularity conditions. The original proof of the lemma is not constructive, but recently, Alon, et al. [1] found a way to convert it into an efficient algorithm. The algorithm is based on the characterization of regularity which states that a pair of subsets of vertices is irregular if and only if either the degrees of "many" vertices are "far from average" or the cardinality of the intersection of neighborhoods of many pairs of vertices is "far from the average case." The algorithmic version of the lemma has already been applied to design the algorithms for various combinatorial problems. Applications include the max-cut problem [7], the tournament ranking problem [3], or the fast algorithm for computing the frequency of a subgraph [4]. Many of these problems have appealing generalizations to hypergraphs, which leads us to a natural question: Can the regularity lemma be extended to hypergraphs?

In the hypergraph case regularity can be measured in a few different ways. The most straightforward and perhaps natural approach defines the density and regularity in the same way as for graphs. The corresponding hypergraph regularity lemma can be then proved along the lines of Szemerédi's proof for graphs (see [11]). In this paper we consider an algorithmic version of this regularity lemma. Note that other versions of regularity lemma were considered in [2], [5], [6], and recently in [10]. In a proof of the regularity lemma it is necessary to distinguish between bipartite graphs which have uniformly distributed edges (say, which are ϵ-regular) from these which are not. For an algorithmic proof we need to have an efficient algorithm. It was proved, however, in [1] that it is co-NP complete to decide if a bipartite graph is ϵ-regular. Authors of [1] got around this difficulty by finding another polynomial checkable characterization which distinguishes between bipartite graphs which fail to be $\epsilon^{1/5}$-regular and those which are ϵ-regular. Note that in the hypergraph case we were unable to find even such a simple characterization.

We found it convenient to consider a slightly more general weighted version ($l = 2$ and $\omega : [V]^l \to \{0, 1\}$ gives the original Szemerédi version). Let $H = (V, \omega)$ be an l-uniform hypergraph with nonnegative weights $\omega : [V]^l \to Z_+ \cup \{0\}$, and let

*Received by the editors February 11, 1999; accepted for publication (in revised form) November 29, 1999; published electronically August 29, 2000.
http://www.siam.org/journals/sicomp/30-4/35172.html
†Department of Mathematics, Arizona State University, P.O. Box 871804, Tempe, Arizona 85287-1804 (andrzej@math.la.asu.edu).
‡Department of Mathematics and Computer Science, Emory University, 1784 N. Decatur Road Suite 100, Atlanta, Georgia 30322 (rodl@mathcs.emory.edu).

$K = \max_{\{v_1,\dots,v_l\}\in[V]^l} \omega(v_1,\dots,v_l) + 1$ (for technical reasons we require that K is strictly larger than $\max_{\{v_1,\dots,v_l\}\in[V]^l} \omega(v_1,\dots,v_l)$). For l subsets $V_1,\dots,V_l$ of V such that $V_i \cap V_j = \emptyset$ if $i \neq j$ define

$$d_\omega(V_1,\dots,V_l) = \frac{\sum\{\omega(v_1,\dots,v_l) : (v_1,\dots,v_l) \in V_1 \times \dots \times V_l\}}{K|V_1|\dots|V_l|}. \tag{1.1}$$

An l-tuple $(V_1,\dots,V_l)$ of subsets of V with $V_i \cap V_j = \emptyset$ is called (ϵ,ω)-regular if for every $W_i \subset V_i$, $i = 1,\dots,l$, with $|W_i| \geq \epsilon|V_i|$ we have

$$|d_\omega(V_1,\dots,V_l) - d_\omega(W_1,\dots,W_l)| < \epsilon. \tag{1.2}$$

An l-tuple $(W_1,\dots,W_l)$ that violates (1.2) is called a witness. A partition $V_0 \cup V_1 \cup \dots \cup V_t$ of V is called (ϵ,ω)-regular if

1. $|V_0| \leq \epsilon|V|$,
2. $|V_i| = |V_j|$ for all $i,j \in [t]$,
3. all but at most ϵt^l l-tuples $(V_{i_1},\dots,V_{i_l})$ with $\{i_1,\dots,i_l\} \subset [t]^l$ are (ϵ,ω)-regular.

The regularity lemma for hypergraphs states that for every $\epsilon > 0$ and every integer m there exist $M = M(\epsilon,m)$ and $N = N(\epsilon,m)$ such that every hypergraph $H = (V,\omega)$ with $|V| \geq N$ admits an (ϵ,ω)-regular partition $V_0 \cup V_1 \cup \dots \cup V_t$ with $m \leq t \leq M$. We show the following theorem.

THEOREM 1.1. *For every l, K, m, and ϵ there exist M, L, and an algorithm which for any l-uniform weighted hypergraph $H = (V,\omega)$ with $K = \max \omega(v_1,\cdots,v_l)+1$ and $|V| = n \geq L$ finds in $O(n^{2l-1}\log^2 n)$ time an (ϵ,ω)-regular partition $V_0 \cup V_1 \cup \dots \cup V_t$ of H with $m \leq t \leq M$.*

Similar results were obtained in Frieze and Kannan [8] (compare also [7]). In particular, [8] contains a randomized algorithm which for every $\epsilon > 0$ and every $\delta > 0$ finds a subset which with probability $1-\delta$ contains all of the information necessary to construct an ϵ-regular partition. Also, [7] and [9] contain many applications of the constructive graph and hypergraph regularity lemma. The approach taken by Frieze and Kannan in [7], [8], and [9] is different from ours. In addition to (1.1), the following density function will be used. Let $(V_1,\dots,V_l)$ be an l-tuple of subsets with $V_i \cap V_j = \emptyset$, and let $1 \leq k \leq l-1$. For $x \in V_{k+1} \times \dots \times V_l$ define

$$d_\omega(x,V_1,\dots,V_k) = \frac{\sum\{\omega(v_1,\dots,v_k,x) : (v_1,\dots,v_k) \in V_1 \times \dots \times V_k\}}{K|V_1|\dots|V_k|}.$$

For an l-tuple $(V_1,\dots,V_l)$ of pairwise disjoint sets and for $1 \leq k \leq l-1$ define

$$ind_k(V_1,\dots,V_l) = \sum_{x\in V_{k+1}\times\dots\times V_l} \frac{(d_\omega(x,V_1,\dots,V_k))^2}{|V_{k+1}|\dots|V_l|} \tag{1.3}$$

and

$$ind_l(V_1,\dots,V_l) = \sum_{x\in V_{l-1}} \frac{(d_\omega(x,V_1,\dots,V_{l-2},V_l))^2}{|V_{l-1}|}. \tag{1.4}$$

Note that

$$ind_{l-1}(V_1,\dots,V_{l-2},V_l,V_{l-1}) = ind_l(V_1,\dots,V_{l-1},V_l). \tag{1.5}$$

Finally, define an index of a partition $P = V_0 \cup V_1 \cup \cdots \cup V_t$ of V as follows:

$$ind(P) = \frac{1}{lt^l} \sum_{(V_{i_1},\ldots,V_{i_l})} \left(\sum_{k=1}^{l} ind_k(V_{i_1},\ldots,V_{i_l}) \right).$$

Clearly, $ind(P)$ is always less than or equal to 1. In the same way as in [1] and [12], if a partition of V is (ϵ,ω)- irregular, then one can construct a subpartition P' such that $ind(P') \geq ind(P) + \frac{\delta^{17}}{16 \cdot l}$, where $\delta = \frac{1}{96} \frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}}$. Then, iterating our "improvement" process $\frac{16 \cdot l}{\delta^{17}} + 1$ times, we obtain an (ϵ,ω)-regular partition of the hypergraph. The rest of the paper is organized as follows. In section 2 we prove some facts about hypergraph densities. Sections 3 and 4 contain two procedures that construct "witness sets" for irregular l-tuples. Both procedures are applied in section 5, where we give the description of the algorithm that constructs an (ϵ,ω)-regular partition of a hypergraph. Section 6 contains the analysis of the algorithm. In section 7, we outline two applications of Theorem 1.1. Finally, it should be noted that there was no attempt made to optimize the constants.

2. Preliminary facts. Let us first observe the following property of densities; compare with [12]. Note that l is a fixed constant independent of n and $1 \leq i \leq l$.

FACT 2.1 (continuity of densities).

1. *Let $G = (X, Y, \omega)$ be a weighted bipartite graph. For $\delta \in (0,1)$, let $X' \subset X$, $Y' \subset Y$ be such that $|X'| \geq (1-\delta)|X|$ and $|Y'| \geq (1-\delta)|Y|$. Then*

$$|d_\omega(X', Y') - d_\omega(X, Y)| \leq 2\delta$$

and

$$|(d_\omega(X', Y'))^2 - (d_\omega(X, Y))^2| \leq 4\delta.$$

2. *Let $(V_1 \cup \cdots \cup V_l, \omega)$ be an l-uniform hypergraph with $V_i \cap V_j = \emptyset$. For $\delta \in (0,1)$, let $\bar{V}_1 \subset V_1, \ldots, \bar{V}_i \subset V_i$ be such that $|\bar{V}_j| \geq (1-\delta)|V_j|$, where $j = 1,\ldots,i$. Then for $x \in V_{i+1} \times \cdots \times V_l$,*

$$|d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i) - d_\omega(x, V_1, \ldots, V_i)| \leq i\delta$$

and

$$|(d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i))^2 - (d_\omega(x, V_1, \ldots, V_i))^2| \leq 2i\delta.$$

In many places of the proof, we will use the following defect form of the Schwarz inequality (see [12]).

FACT 2.2. *If for some $m < n$, $\sum_{k=1}^{m} X_k = \frac{m}{n} \sum_{k=1}^{n} X_k + \rho$, then*

$$\sum_{k=1}^{n} X_k^2 \geq \frac{1}{n} \left(\sum_{k=1}^{n} X_k \right)^2 + \frac{\rho^2 n}{m(n-m)}.$$

Let $X = \{x_1, x_2, \ldots, x_N\}$ and $U = \{u_1, u_2, \ldots, u_M\}$ be two disjoint sets. Let $G = (X, U, \omega)$ be a bipartite graph with nonnegative weights on edges $\omega(x,u) < K$. For $i = 1, 2, \ldots, M$, $j = 1, 2, \ldots, N$ set

$$deg(x_j) = \sum_{i=1}^{M} \omega(x_j, u_i),$$

$$d_j = \frac{deg(x_j)}{MK},$$

$$deg(u_i) = \sum_{j=1}^{N} \omega(x_j, u_i),$$

$$\Delta_i = \frac{deg(u_i)}{NK}.$$

We also set

$$d = d_\omega(X,U) = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N}\omega(x_j,u_i)}{MNK} = \frac{\sum_{j=1}^{N} d_j}{N} = \frac{\sum_{i=1}^{M}\Delta_i}{M}$$

and $T = dMNK$ for the total weight of all edges. For $x_i \in X$ consider the vector

$$\overrightarrow{x_i} = (\omega(x_i,u_k))_{k=1}^{M}.$$

DEFINITION 2.3. *A graph $G = (X,U,\omega)$ is called δ-vector regular if $|\langle \overrightarrow{x_i}, \overrightarrow{x_j}\rangle - d_i d_j MK^2| < \delta MK^2$ for all but at most $\delta\binom{N}{2}$ pairs $\{x_i, x_j\}$.*

LEMMA 2.4. *For $\epsilon, \delta \in (0,1)$, suppose $G = (X,U,\omega)$ is δ-vector regular and $|X| \geq 1/\delta$. Then for every $U' \subset U$ with $|U'| \geq \epsilon|U|$*

$$|d_\omega(X,U') - d_\omega(X,U)| < \epsilon,$$

provided $\epsilon^3 \geq 3\delta$.

Proof. Suppose the lemma is false and consider $U' \subset U$ such that $m = |U'| \geq \epsilon|U|$ and

$$|d_\omega(X,U') - d_\omega(X,U)| \geq \epsilon. \tag{2.1}$$

Without loss of generality, assume that $U' = \{u_1, \ldots, u_m\}$, where $m \geq \epsilon M$. Equation (2.1) is equivalent to

$$\left|\sum_{i=1}^{m}\Delta_i - dm\right| \geq \epsilon m. \tag{2.2}$$

From Fact 2.2 (applied with $\rho = \epsilon m$ and $n = M$), we infer that

$$\sum_{i=1}^{M}\Delta_i^2 \geq M\left(d^2 + \frac{\epsilon^3}{1-\epsilon}\right), \tag{2.3}$$

which implies

$$\sum_{i=1}^{M}\binom{deg(u_i)}{2} = \sum_{i=1}^{M}\binom{\Delta_i NK}{2} = \sum_{i=1}^{M}\frac{\Delta_i NK(\Delta_i NK - 1)}{2}$$

$$(2.4) \geq \frac{1}{2}N^2MK^2\left(d^2 + \frac{\epsilon^3}{1-\epsilon}\right) - \frac{NK}{2}\sum_{i=1}^{M}\Delta_i = \frac{1}{2}N^2MK^2\left(d^2 + \frac{\epsilon^3}{1-\epsilon}\right) - \frac{T}{2}.$$

On the other hand, we have

$$\sum_{i=1}^{M}\binom{deg(u_i)}{2}=\sum_{i=1}^{M}\binom{\sum_{j=1}^{N}\omega(x_j,u_i)}{2}=\frac{1}{2}\sum_{i=1}^{M}\left(\sum_{j=1}^{N}\omega(x_j,u_i)\left(\sum_{k=1}^{N}\omega(x_k,u_i)-1\right)\right)$$

$$\leq\frac{1}{2}\left(\sum_{j,k\in[N]}\left(\sum_{i=1}^{M}\omega(x_k,u_i)\omega(x_j,u_i)\right)-T\right)=\frac{1}{2}\left(\sum_{j,k\in[N]}\langle\overrightarrow{x_j},\overrightarrow{x_k}\rangle-T\right). \tag{2.5}$$

By assumption, we know that for all but at most $\delta\binom{N}{2}$ of pairs $\{x_i,x_j\}$, $\langle\overrightarrow{x_i},\overrightarrow{x_j}\rangle\leq(d_id_j+\delta)MK^2$, and we can bound the scalar product for the remaining pairs: $\langle\overrightarrow{x_i},\overrightarrow{x_j}\rangle<MK^2$. Therefore,

$$\sum_{j,k\in[N]}\langle\overrightarrow{x_j},\overrightarrow{x_k}\rangle<\sum_{j,k\in[N]}(d_jd_k+\delta)MK^2+\sum_{j=1}^{N}\langle\overrightarrow{x_j},\overrightarrow{x_j}\rangle+\delta N^2MK^2$$

$$\leq\sum_{j,k\in[N]}(d_jd_k+\delta)MK^2+2\delta N^2MK^2$$

$$\leq\left(\sum_{i=1}^{N}d_i\right)^2MK^2+3\delta N^2MK^2\leq(d^2+3\delta)N^2MK^2. \tag{2.6}$$

By combining (2.5) and (2.6), we see that

$$\sum_{i=1}^{M}\binom{deg(u_i)}{2}<\frac{1}{2}((d^2+3\delta)N^2MK^2-T). \tag{2.7}$$

Comparing (2.4) and (2.7) gives

$$(d^2+3\delta)N^2MK^2>N^2MK^2\left(d^2+\frac{\epsilon^3}{1-\epsilon}\right),$$

which gives $3\delta>\frac{\epsilon^3}{1-\epsilon}$. This contradicts our assumption that $\epsilon^3\geq3\delta$. □

Let $V_1,\ldots,V_k$ be pairwise disjoint sets and let $(V_1\cup\cdots\cup V_k,\omega)$ be a weighted k-uniform hypergraph. We consider a bipartite graph (U,X,ω), where $U=V_1$, $X=\{x_1,\ldots,x_N\}=V_2\times\cdots\times V_k$, and where $\omega(u,x)=\omega(v_1,v_2,\ldots,v_k)$ if $u=v_1$ and $x=(v_2,\ldots,v_k)$. The next lemma shows that irregularity of an l-tuple can be reduced either to vector irregularity or to irregularity of an $(l-1)$-tuple, with weights appropriately defined.

LEMMA 2.5. *If (X,U) is the bipartite graph defined above which moreover satisfies the conditions*

1. $|\langle\overrightarrow{x_i},\overrightarrow{x_j}\rangle-d_id_jMK^2|<\delta MK^2$ *for all but at most* $\delta\binom{N}{2}$ *pairs* $\{x_i,x_j\}$,
2. $|d_\omega(V_1,V_2',\ldots,V_k')-d_\omega(V_1,V_2,\ldots,V_k)|<\frac{\epsilon}{2}$ *for every* $V_i'\subset V_i$ *with* $|V_i'|\geq\frac{\epsilon}{2}|V_i|$,

then for every $V_i' \subset V_i$ *with* $|V_i'| \geq \epsilon |V_i|$

$$|d_\omega(V_1', V_2', \ldots, V_k') - d_\omega(V_1, V_2, \ldots, V_k)| < \epsilon,$$

provided $48\delta \leq \epsilon^{2k+1}$.

Proof. The triangle inequality and the second condition imply that for every $V_i' \subset V_i$ with $|V_i'| \geq \epsilon |V_i|$

$$|d_\omega(V_1', V_2', \ldots, V_k') - d_\omega(V_1, V_2, \ldots, V_k)| \leq \frac{\epsilon}{2} + |d_\omega(V_1', V_2', \ldots, V_k') - d_\omega(V_1, V_2', \ldots, V_k')|.$$

Let $X' = V_2' \times \cdots \times V_k'$. Observe that for at most $\delta\binom{N}{2} \leq \delta\frac{N^2}{2} \leq \frac{\delta}{\epsilon^{2(k-1)}}\frac{|X'|^2}{2} \leq \frac{2\delta}{\epsilon^{2(k-1)}}\binom{|X'|}{2}$ of the pairs $\{x_i, x_j\}$

$$|\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - d_i d_j M K^2| \geq \delta M K^2,$$

and so Lemma 2.4 (applied to X') implies that

$$|d_\omega(V_1', X') - d_\omega(V_1, X')| < \frac{\epsilon}{2}$$

as $6\frac{\delta}{\epsilon^{2(k-1)}} \leq (\frac{\epsilon}{2})^3$. Clearly, $d_\omega(Y, X') = d_\omega(Y, V_2', \ldots, V_k')$, which shows that

$$|d_\omega(V_1', V_2', \ldots, V_k') - d_\omega(V_1, V_2, \ldots, V_k)| < \epsilon. \qquad \square$$

Let $\omega'(v_2, \ldots, v_k) = \sum_{v_1 \in V_1} \omega(v_1, v_2, \ldots, v_k)$ and $K' = K|V_1|$. Then from Lemma 2.5 we see that if a k-tuple $(V_1, V_2, \ldots, V_k)$ is (ϵ, ω)- irregular, then either $|\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - d_i d_j M K^2| > \delta M K^2$ for at least δN^2 pairs $\{x_i, x_j\}$, with $\delta = \epsilon^{2k+1}/48$, or the $(k-1)$-tuple $(V_2, \ldots, V_k)$ is $(\frac{\epsilon}{2}, \omega')$-irregular.

3. Finding witnesses of vector irregularity. Let (X, U, ω) be a weighted "vector irregular" bipartite graph with $X = \{x_1, \ldots, x_N\}$ and $U = \{u_1, \ldots, u_M\}$. In this section we will show how to construct sets $X' \subset X$ and $U' \subset U$ such that $d_\omega(X', U')$ essentially differs from $d_\omega(X', U)$.

THEOREM 3.1. *Let* $\delta \leq \frac{1}{3}$ *and assume that for at least* $\delta\binom{|X|}{2}$ *of the pairs* $\{x_i, x_j\}$ *we have* $|\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - d_i d_j M K^2| \geq \delta M K^2$. *Then there is an* $O(N^2 M \log^2 K)$ *algorithm that finds sets* $X' \subset X$ *with* $|X'| > \frac{\delta^7}{2}|X|$ *and* $U' \subset U$ *with* $|U'| > \delta^6 |U|$ *such that*

$$|d_\omega(X', U') - d_\omega(X', U)| > \delta^2.$$

Proof. Set $\epsilon = \delta^3$ (we will assume that $1/\epsilon$ is an integer), and partition the set X into sets X_r in the following way:

$$X_r = \{x \in X : \epsilon r K M \leq deg(x) < \epsilon(r+1) K M\} \tag{3.1}$$

for $r = 0, \ldots, \frac{1}{\epsilon} - 1$. Since at least $\delta\binom{|X|}{2}$ of the pairs $\{x_i, x_j\}$ satisfy $|\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - d_i d_j M K^2| \geq \delta M K^2$, we can find $x_0 \in X$ such that for at least $\frac{\delta}{2}N$ of the x_j's

$$\langle \overrightarrow{x_0}, \overrightarrow{x_j} \rangle - d_0 d_j M K^2 \geq \delta M K^2, \tag{3.2}$$

or

$$-(\langle \overrightarrow{x_0}, \overrightarrow{x_j} \rangle - d_0 d_j M K^2) \geq \delta M K^2. \tag{3.3}$$

We assume (3.2) and note that in the case of (3.3) the proof can be repeated with minor changes. Then, we can find $r \in \{0, \dots, \frac{1}{\epsilon} - 1\}$ such that for at least $\frac{\epsilon\delta}{2}N$ of x_j's in X_r, $\langle \overrightarrow{x_0}, \overrightarrow{x_j} \rangle - d_0 d_j MK^2 \geq \delta MK^2$. Set $\bar{X} = \{x_j \in X_r : \langle \overrightarrow{x_j}, \overrightarrow{x_0} \rangle \geq (d_0 d_j + \delta)MK^2\}$, and observe that $|\bar{X}| \geq \frac{\epsilon\delta}{2}N$. In addition to the partition $X = \bigcup X_r$, we compute a partition of $U = \bigcup U_s$, where

$$U_s = \{u \in U : \epsilon s K \leq \omega(x_0, u) < \epsilon(s+1)K\} \tag{3.4}$$

with $s = 0, \dots, \frac{1}{\epsilon} - 1$.

CLAIM 3.2. $\sum_{s=0}^{1/\epsilon-1} (s+1)\epsilon K |U_s| \leq (d_0 + \epsilon)MK$.

Proof.

$$\sum_{s=0}^{1/\epsilon-1} (s+1)\epsilon K |U_s| = \sum s\epsilon K |U_s| + \epsilon K \sum |U_s| \leq deg(x_0) + \epsilon MK = (d_0 + \epsilon)MK. \qquad \square$$

CLAIM 3.3. *For every* $x_j \in \bar{X}$

$$\sum_{s;|U_s|>\epsilon^2 M} (s+1)\epsilon K \sum_{u_i \in U_s} \omega(x_j, u_i) \geq (d_0 d_j + \delta - \epsilon)MK^2.$$

Proof. In view of the definition of $\bar{X}$, for every $x_j \in \bar{X}$ the following holds.

$$(d_0 d_j + \delta)MK^2 \leq \langle \overrightarrow{x_0}, \overrightarrow{x_j} \rangle \leq \sum_{s=0}^{1/\epsilon-1} (s+1)\epsilon K \sum_{u_i \in U_s} \omega(x_j, u_i)$$

$$\leq \sum_{s;|U_s|>\epsilon^2 M} (s+1)\epsilon K \sum_{u_i \in U_s} \omega(x_j, u_i) + \epsilon^3 K^2 M \sum_s (s+1)$$

$$\leq \sum_{s;|U_s|>\epsilon^2 M} (s+1)\epsilon K \sum_{u_i \in U_s} \omega(x_j, u_i) + \epsilon K^2 M. \qquad \square$$

Next, we will show the following claim.

CLAIM 3.4. *Fix* $x_{j_0} \in \bar{X}$ *(arbitrarily). Then there exists* $\bar{s} \in \{0, \dots, \frac{1}{\epsilon} - 1\}$ *such that*

(i) $|U_{\bar{s}}| \geq \epsilon^2 M$, *and*

(ii) $\sum_{u_i \in U_{\bar{s}}} \omega(x_j, u_i) \geq |U_{\bar{s}}|(d_{j_0} + \delta^2 + \epsilon)K$ *holds for at least* $\epsilon|\bar{X}|$ *of the* $x_j \in \bar{X}$.

Proof. First, we show that for every $x_j \in \bar{X}$ there is $s \in \{0, \dots, \frac{1}{\epsilon} - 1\}$ such that

$$|U_s| > \epsilon^2 M \tag{3.5}$$

and

$$\sum_{u_i \in U_s} \omega(x_j, u_i) \geq |U_s| \frac{(d_0 d_j + \delta - \epsilon)K}{d_0 + \epsilon}. \tag{3.6}$$

Indeed, assume that there exists j such that for every $|U_s| > \epsilon^2 M$ we have

$$\sum_{u_i \in U_s} \omega(x_j, u_i) < |U_s| \frac{(d_0 d_j + \delta - \epsilon)K}{d_0 + \epsilon}.$$

Then

$$\sum_{s;|U_s|>\epsilon^2 M}(s+1)\epsilon K\sum_{u_i\in U_s}\omega(x_j,u_i)<\sum_s(s+1)\epsilon K|U_s|\frac{(d_0d_j+\delta-\epsilon)K}{d_0+\epsilon},$$

which by Claim 3.2 is less than or equal to $(d_0d_j+\delta-\epsilon)MK^2$. This, however, contradicts Claim 3.3.

Since $\epsilon=\delta^3$ and $\delta\le\frac{1}{3}$, one can further simplify the right-hand side of (3.6) to infer that for every $x_j\in\bar{X}$ there exists s such that

$$|U_s|>\epsilon^2M$$

and

$$\sum_{u_i\in U_s}\omega(x_j,u_i)\ge|U_s|K(d_j+\delta^2+2\epsilon). \tag{3.7}$$

It follows from the definition of X_r that for every $x_{j_1},x_{j_2}\in\bar{X}$, $|d_{j_1}-d_{j_2}|\le\epsilon$, which implies that

$$\sum_{u_i\in U_s}\omega(x_j,u_i)\ge|U_s|(d_{j_0}+\delta^2+\epsilon)K, \tag{3.8}$$

if x_{j_0} is chosen arbitrarily from $\bar{X}$. Therefore, for every $x_{j_0}\in\bar{X}$ and every $x_j\in\bar{X}$ there is an $s\in\{0,\dots,\frac{1}{\epsilon}-1\}$ such that

$$|U_s|>\epsilon^2M$$

and

$$\sum_{u_i\in U_s}\omega(x_j,u_i)\ge|U_s|(d_{j_0}+\delta^2+\epsilon)K.$$

In order to prove (i) and (ii) we need to "reverse" the quantifiers of j and s. We know that for every j there is a "big" set U_s such that (3.8) holds. Since there are at most $\frac{1}{\epsilon}$ choices of s, there exists $\bar{s}\in\{0,\dots,\frac{1}{\epsilon}-1\}$ such that $U_{\bar{s}}$ is "big" and for at least $\epsilon|\bar{X}|$ of the x_j's (3.8) holds. More precisely, there exists $\bar{s}\in\{0,\dots,\frac{1}{\epsilon}-1\}$ such that

(i) $|U_{\bar{s}}|\ge\epsilon^2M$, and

(ii) $\sum_{u_i\in U_{\bar{s}}}\omega(x_j,u_i)\ge|U_{\bar{s}}|(d_{j_0}+\delta^2+\epsilon)K$ holds for at least $\epsilon|\bar{X}|$ of the $x_j\in\bar{X}$,

which proves the claim. □

Let $U'=U_{\bar{s}}$, and let X' be the set of those $\epsilon|\bar{X}|$ vertices from $\bar{X}$ that satisfy (3.8). Observe that $|U'|\ge\delta^6|U|$ and $|X'|\ge\frac{\delta^7}{2}|X|$. We have

$$d_\omega(X',U')=\frac{\sum_{x_j\in X',u_i\in U'}\omega(x_j,u_i)}{K|X'||U'|}\ge\frac{|U'||X'|(d_{j_0}+\delta^2+\epsilon)K}{K|X'||U'|}=(d_{j_0}+\delta^2+\epsilon)$$

and

$$d_\omega(X',U)=\frac{\sum_{x\in X'}deg(x)}{K|X'||U|}\le\frac{|X'|(deg(x_{j_0})+\epsilon KM)}{K|X'||U|}=d_{j_0}+\epsilon.$$

Therefore,

$$d_\omega(X',U')-d_\omega(X',U)\ge(d_{j_0}+\delta^2+\epsilon)-d_{j_0}-\epsilon=\delta^2. \tag{3.9}$$

The above proof gives an efficient algorithm: First compute scalar products to find x_0 and partitions of X and U. Then check all sets U_s to find one that satisfies Claim 3.4. The main computational task is to compute $O(N^2)$ scalar products $\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle = \sum_{l=1}^{M} \omega(x_i, u_l)\omega(x_j, u_l)$. Since $\omega(x_j, u_l) < K$, the multiplication $\omega(x_i, u_l)\omega(x_j, u_l)$ can be done in $O(\log^2 K)$ time. Thus the total number of steps is $O(N^2 M \log^2 K)$. ∎

4. Finding a witness of irregularity in a weighted bipartite graph. Let $G = (X, Y, \omega)$ be a bipartite graph with nonnegative weights on edges $\omega(x, y) < K$. In this section, we will show how to find $X' \subset X$ and $Y' \subset Y$ such that

$$|d_\omega(X', Y') - d_\omega(X, Y)| > \delta^2.$$

Unlike in the previous section neither X nor Y will be products of other sets. Note that the algorithm of this section is a generalization of the algorithm of [1] to weighted graphs. Let $M = |X| = |Y|$ and let $d = d_\omega(X, Y)$. We first observe the following fact.

FACT 4.1. *For $\rho \in (0, 1)$, let $X^* = \{x \in X : |deg(x) - dMK| \geq \rho MK\}$. If $|X^*| \geq \rho M$, then there is $X^{**} \subset X^*$ with $|X^{**}| \geq \frac{\rho}{2} M$ such that*

$$|d_\omega(X^{**}, Y) - d_\omega(X, Y)| \geq \rho.$$

Proof. Let $X^{**} = \{x \in X^* : deg(x) - dMK \geq \rho MK\}$. Without loss of generality we can assume that $|X^{**}| \geq \frac{|X^*|}{2}$. Then,

$$d_\omega(X^{**}, Y) - d_\omega(X, Y) = \frac{\sum_{x \in X^{**}} deg(x) - d|X^{**}|MK}{|X^{**}|MK} \geq \rho. \qquad \blacksquare$$

LEMMA 4.2. *For $\epsilon, \delta \in (0, 1)$, $\delta^2 < \epsilon$, $M \geq \frac{1}{\delta}$, let $X^* = \{x \in X : |deg(x) - dMK| > \delta^2 MK\}$. If both of the following conditions are satisfied*

1. *$|X^*| < \delta^2 M$,*
2. *for all but at most $\delta \binom{M}{2}$ of the pairs $\{x_i, x_j\}$ of vertices from X, $|\langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - d^2 K^2 M| < \delta K^2 M$,*

then for every $X' \subset X$ with $|X'| \geq \epsilon M$ and every $Y' \subset Y$ with $|Y'| \geq \epsilon M$ we have

$$|d_\omega(X', Y') - d_\omega(X, Y)| \leq 2\frac{\delta^2}{\epsilon} + \frac{\sqrt{5\delta}}{\epsilon^2 - \epsilon\delta^2}.$$

Proof. Fix $X' \subset X$ and $Y' \subset Y$ with $|X'| \geq \epsilon|X|$ and $|Y'| \geq \epsilon|Y|$. Let $X'' = X' \setminus X^*$. Note that $|X''| > (1 - \frac{\delta^2}{\epsilon})|X'|$. Without loss of generality, we can assume that $X'' = \{x_1, \ldots, x_m\}$ and $Y' = \{y_1, \ldots, y_n\}$. For $i = 1, \ldots, m$, we consider vectors $\overrightarrow{a_i} = \overrightarrow{x_i} - (dK, \ldots, dK)$ and hence $\overrightarrow{a_i} = (a_{i1}, \ldots, a_{iM})$, where

$$a_{ij} = \omega(x_i, y_j) - dK. \tag{4.1}$$

Then, due to the fact that

$$||\overrightarrow{a_1} + \cdots + \overrightarrow{a_m}||^2 = \sum_{i=1}^{m} ||\overrightarrow{a_i}||^2 + \sum_{i \neq j} \langle \overrightarrow{a_i}, \overrightarrow{a_j} \rangle$$

and from the form of a_i's, we see that

$$\langle \overrightarrow{a_i}, \overrightarrow{a_j} \rangle = \langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - dK \sum_{l=1}^{M} \omega(x_i, y_l) - dK \sum_{l=1}^{M} \omega(x_j, y_l) + d^2K^2M$$

$$\leq \langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle - 2dK(dKM - \delta^2 KM) + d^2K^2M \leq \langle \overrightarrow{x_i}, \overrightarrow{x_j} \rangle + 2\delta^2 dK^2M - d^2K^2M.$$

Therefore, for all but at most $\delta\binom{M}{2}$ of $\{x_i, x_j\}$

$$\langle \overrightarrow{a_i}, \overrightarrow{a_j} \rangle \leq \delta K^2 M + 2\delta^2 dK^2 M \leq 3\delta K^2 M$$

and always

$$\langle \overrightarrow{a_i}, \overrightarrow{a_j} \rangle \leq K^2 M.$$

We infer that

$$||\overrightarrow{a_1} + \cdots + \overrightarrow{a_m}||^2 \leq \sum_{i=1}^{m} ||\overrightarrow{a_i}||^2 + 3\delta K^2 M^3 + \delta K^2 M^3$$

$$\leq K^2M^2 + 4\delta K^2 M^3 \leq 5\delta K^2 M^3,$$

as $M \geq 1/\delta$. On the other hand, we have $||\overrightarrow{a_1} + \cdots + \overrightarrow{a_m}||^2 = \xi_1^2 + \cdots + \xi_M^2$, where $\xi_i = a_{1i} + \cdots + a_{mi}$. Then $\xi_1^2 + \cdots + {\xi_M}^2 \geq \xi_1^2 + \cdots + \xi_n^2 \geq \frac{1}{n}(\xi_1 + \cdots + \xi_n)^2$, which implies

$$(\xi_1 + \cdots + \xi_n)^2 \leq 5\delta K^2 M^3 n \leq 5\delta K^2 M^4.$$

We infer that

$$|\xi_1 + \cdots + \xi_n| \leq \sqrt{5\delta} K M^2, \tag{4.2}$$

and by (4.1) we can write (4.2) as follows:

$$\left| \sum_{x \in X'', y \in Y'} \omega(x, y) - |X''||Y'|dK \right| \leq \sqrt{5\delta} K M^2.$$

Therefore,

$$|d_\omega(X'', Y') - d| \leq \frac{\sqrt{5\delta} M^2}{|X''||Y'|},$$

and, since $|X''| \geq (1 - \frac{\delta^2}{\epsilon})|X'|$, we infer by continuity of density (Fact 2.1) that

$$|d_\omega(X', Y') - d| \leq |d_\omega(X'', Y') - d| + |d_\omega(X'', Y') - d_\omega(X', Y')|$$

$$\leq 2\frac{\delta^2}{\epsilon} + \frac{\sqrt{5\delta} M^2}{|X''||Y'|}. \tag{4.3}$$

Now, since $|Y'| \geq \epsilon M$ and $|X''| \geq (1-\frac{\delta^2}{\epsilon})\epsilon M$, we can further estimate the right-hand side of (4.3) to obtain

$$|d_\omega(X',Y') - d| \leq 2\frac{\delta^2}{\epsilon} + \frac{\sqrt{5\delta}}{\epsilon^2 - \epsilon\delta^2}. \qquad \square$$

THEOREM 4.3. *Let $G = (X, Y, \omega)$ be a weighted bipartite graph with $M = |X| = |Y|$ and $0 \leq \omega(x,y) < K$. For $\epsilon < 1/10$ and $\delta = \epsilon^7$, if $M > \frac{1}{\delta}$, then there exists an $O(M^3 \log^2 K)$ algorithm such that if (X,Y) is (ϵ,ω)-irregular, then it finds $X' \subset X$, with $|X'| > \delta^4|X|$ and $Y' \subset Y$, with $|Y'| > \delta^4|Y|$ such that*

$$|d_\omega(X',Y') - d_\omega(X,Y)| > \delta^2.$$

Proof. First observe that due to the assumptions about δ and ϵ, $2\frac{\delta^2}{\epsilon} + \frac{\sqrt{5\delta}}{\epsilon^2-\epsilon\delta^2} \leq \epsilon$. Since (X,Y) is (ϵ,ω)-irregular, we can infer from Lemma 4.2 that either

(i) $|X^*| \geq \delta^2 M$, or

(ii) for at least $\delta\binom{M}{2}$ pairs $\{x_i, x_j\}$, $|\langle \overrightarrow{x_i}, \overrightarrow{x_j}\rangle - d^2K^2M| \geq \delta K^2 M$.

In case (i), set $X' = X^{**}$ from Fact 4.1, and $Y' = Y$. Then, by Fact 4.1,

$$|d_\omega(X',Y') - d_\omega(X,Y)| \geq \delta^2.$$

Observe that constructing the set X^{**} requires computing the degrees of $x_j \in X$ which can be done in $O(M^2 \log K)$ time. Hence, in this case we are done. Next, we will show how to construct the witness sets X' and Y' if (ii) holds while (i) is false. Assume that for at least $\delta\binom{M}{2}$ of pairs $\{x_i, x_j\}$, $|\langle \overrightarrow{x_i}, \overrightarrow{x_j}\rangle - d^2K^2M| > \delta K^2 M$ and that $|X^*| < \delta^2 M$. Then for at least $(\delta - 2\delta^2)\binom{M}{2}$ of pairs $\{x_i, x_j\}$, where $x_i, x_j \in X \setminus X^*$, we have $|\langle \overrightarrow{x_i}, \overrightarrow{x_j}\rangle - d^2K^2M| > \delta K^2 M$. Therefore, we can find $x_0 \in X \setminus X^*$ such that for at least $\frac{\delta - 2\delta^2}{2} M$ of $x_j \in X$, either

$$\langle \overrightarrow{x_0}, \overrightarrow{x_j}\rangle - d^2K^2M > \delta K^2 M, \tag{4.4}$$

or

$$-(\langle \overrightarrow{x_0}, \overrightarrow{x_j}\rangle - d^2K^2M) > \delta K^2 M. \tag{4.5}$$

We assume (4.4) and set $\rho = \delta^2$. Let $\bar{X} = \{x_j \in X \setminus X^* : \langle \overrightarrow{x_0}, \overrightarrow{x_j}\rangle - d^2K^2M > \delta K^2 M\}$. We partition the set Y as follows:

$$Y_s = \{y \in Y : s\rho K \leq \omega(x_0, y) \leq (s+1)\rho K\}, \tag{4.6}$$

where $s = 0, \ldots, \frac{1}{\rho} - 1$.

FACT 4.4.

1. $\sum_{s=0}^{1/\rho-1}(s+1)\rho K|Y_s| \leq (d+2\rho)KM$.
2. *For every $x_j \in \bar{X}$*

$$\sum_{s;|Y_s|\geq\rho^2 M} (s+1)\rho K \sum_{y\in Y_s} \omega(x_j, y) \geq (d^2 + \delta - \rho)K^2 M. \tag{4.7}$$

Proof. From (4.6) and the fact that for any $x_0 \in X \setminus X^*$, $deg(x_0) \leq (d+\rho)KM$ we infer that

$$\sum_{s=0}^{1/\rho-1} (s+1)\rho K|Y_s| = \sum s\rho K|Y_s| + \rho K \sum |Y_s| \leq deg(x_0) + \rho KM \leq dKM + 2\rho KM,$$

which shows the first part. To prove the second part, we observe that

$$\langle \overrightarrow{x_0}, \overrightarrow{x_j} \rangle = \sum_{y \in Y} \omega(x_0, y)\omega(x_j, y) \leq \sum_{s=0}^{1/\rho - 1} (s+1)\rho K \sum_{y \in Y_s} \omega(x_j, y)$$

$$\leq \sum_{s; |Y_s| \geq \rho^2 M} (s+1)\rho K \sum_{y \in Y_s} \omega(x_j, y) + \rho^3 M K^2 \sum (s+1)$$

$$\leq \sum_{s; |Y_s| \geq \rho^2 M} (s+1)\rho K \sum_{y \in Y_s} \omega(x_j, y) + \rho M K^2. \tag{4.8}$$

Comparing (4.4) and (4.8), we infer the inequality (4.7). □

We will first show that for every $x_j \in \bar{X}$ there is s such that

$$|Y_s| \geq \rho^2 M \tag{4.9}$$

and

$$\sum_{y \in Y_s} \omega(x_j, y) \geq |Y_s| \frac{(d^2 + \delta - \rho)K}{(d + 2\rho)}. \tag{4.10}$$

Indeed, assume that there exists j such that for every s such that $|Y_s| \geq \rho^2 M$ we have $\sum_{y \in Y_s} \omega(x_j, y) < |Y_s| \frac{(d^2+\delta-\rho)K}{(d+2\rho)}$. Then averaging over all "big" Y_s we see that

$$\sum_{s; |Y_s| \geq \rho^2 M} (s+1)K\rho \sum_{y \in Y_s} \omega(x_j, y) < \sum_{s; |Y_s| \geq \rho^2 M} (s+1)\rho K |Y_s| \frac{(d^2 + \delta - \rho)K}{(d + 2\rho)},$$

which by Fact 4.4 part (1) is less than or equal to $(d^2 + \delta - \rho)K^2 M$. This however contradicts part (2) of Fact 4.4.

Since $\delta = \epsilon^7 < 1/10^7$, the right-hand side of (4.10) can be simplified:

$$\sum_{y \in Y_s} \omega(x_j, y) \geq |Y_s| K \left(d + \frac{\delta}{2} \right). \tag{4.11}$$

Thus for every $x_j \in \bar{X}$ there is $s \in \{0, \ldots, \frac{1}{\rho} - 1\}$ such that $|Y_s| \geq \rho^2 M$ and (4.11) holds. Similarly as in section 3 we use the simple pigeonhole principle to "reverse" the quantifiers of j and s to obtain the existence of $\bar{s} \in \{0, \ldots, \frac{1}{\rho} - 1\}$ that satisfies

(i) $|Y_{\bar{s}}| \geq \rho^2 M$ and

(ii) for at least $\rho|\bar{X}|$ of x_j's from $\bar{X}$, $\sum_{y \in Y_{\bar{s}}} \omega(x_j, y) \geq |Y_{\bar{s}}| K(d + \frac{\delta}{2})$.

Let $Y' = Y_{\bar{s}}$ and let X' be the set of those x_j's that satisfy (ii). Then

$$d_\omega(X', Y') - d = \frac{\sum_{y \in Y', x_j \in X'} \omega(x_j, y)}{K|X'||Y'|} - d \geq \frac{|X'||Y_{\bar{s}}| K(d + \frac{\delta}{2})}{K|X'||Y'|} - d = \frac{\delta}{2} > \delta^2.$$

In order to construct X' and Y' we must compute the scalar products to check (4.4), compute the partition (4.6), and check condition (ii). Since $\omega(x_j, y) < K$ this can be done in $O(M^3 \log^2 K)$ time. □

5. The main algorithm. In this section, we will describe the main algorithm which finds a refinement of an (ϵ,ω)-irregular partition such that the value of the index of the refined partition is closer to one. Let us first outline the idea in the case $l=3$. For each (ϵ,ω)-irregular triple (V_1,V_2,V_3) we either find (using the algorithm from Theorem 3.1) $V_1'\subset V_1$ and $X'\subset V_2\times V_3$ such that

$$|d_{\omega_1}(X',V_1')-d_{\omega_1}(X',V_1)|\geq\delta_1,$$

or using the algorithm from Theorem 4.3, we find $V_2'\subset V_2$ and $V_3'\subset V_3$ such that

$$|d_{\omega_2}(V_2',V_3')-d_{\omega_2}(V_2,V_3)|\geq\delta_2,$$

where ω_1, ω_2, δ_1, δ_2 are defined below. In the first case, only V_1' is used to refine a given partition; in the second case both V_2' and V_3' are used. Let (V,H,ω) be an l-uniform weighted hypergraph with $K=\max|\omega(v_1,v_2,\dots,v_l)|+1$. We introduce some additional notation: For an l-tuple $(V_1,\dots,V_l)$

$$\omega_1(v_1,\dots,v_l)=\omega(v_1,\dots,v_l),\qquad K_1=K,$$

$$\omega_2(v_2,\dots,v_l)=\sum_{v_1\in V_1}\omega_1(v_1,v_2,\dots,v_l),\qquad K_2=|V_1|K_1.$$

In general,

$$\omega_i(v_i,\dots,v_l)=\sum_{v_{i-1}\in V_{i-1}}\omega_{i-1}(v_{i-1},v_i,\dots,v_l),\qquad K_i=|V_{i-1}|K_{i-1}. \tag{5.1}$$

Also, we set

$$\epsilon_k=\frac{\epsilon}{2^{k-1}}$$

and

$$\delta_i=\frac{1}{48}\epsilon_i^{2(l-i+1)+1}.$$

Also, set $X_k=V_{k+1}\times\cdots\times V_l$. Note that for every $i\in[l]$,

$$\delta_i\geq\frac{1}{48}\frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}}. \tag{5.2}$$

We can now describe the procedure that "improves" a given partition P. For each (ϵ_1,ω_1)-irregular l-tuple, $(V_1,\dots,V_l)$ consider the weighted bipartite graph (V_1,X_1,ω_1), where $X_1=V_2\times\cdots\times V_l$ and for $x=(v_2,\dots,v_l)$, $\omega_1(v,x)=\omega_1(v,v_2,\cdots,v_l)$. Set $M=|V_1|$ and $N=|X_1|$. The algorithm is illustrated in Figure 5.1: Lemma 2.5 implies that if $(V_1,\dots,V_l)$ is (ϵ_1,ω_1)-irregular, then either (X_1,V_1) is vector irregular and for at least $\delta_1\binom{N}{2}$ of pairs $\{x_i,x_j\}$ of vertices in X_1

$$|\langle\overrightarrow{x_i},\overrightarrow{x_j}\rangle-K_1^2Md_id_j|>\delta_1K_1^2M, \tag{5.3}$$

or the $(l-1)$-tuple $(V_2,\dots,V_l)$ is (ϵ_2,ω_2)-irregular. If (5.3) holds, then we can use the algorithm from Theorem 3.1 to find $X_1'\subset X_1$ with $|X_1'|\geq\frac{\delta_1^7}{2}|X_1|$ and $V_1'\subset V_1$ with $|V_1'|\geq\delta_1^6|V_1|$ such that

$$|d_{\omega_1}(X_1',V_1')-d_{\omega_1}(X_1',V_1)|>\delta_1^2. \tag{5.4}$$

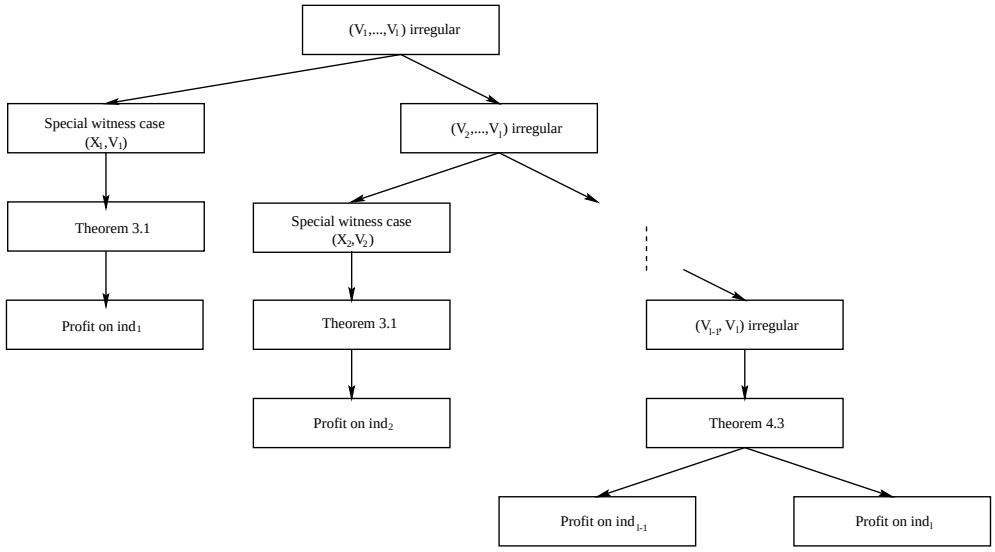


FIG. 5.1. *Proof of Theorem* 1.1.

Note that X_1' will not be used as a witness when improving the partition P; only V_1' will. In case (5.3) does not hold, we continue to apply Lemma 2.5 to the $(l-i)$- tuples $(V_i, \dots, V_l)$ until $i = l-2$. Finally, if we haven't found a witness set so far, we apply Theorem 4.3 to the pair (V_{l-1}, V_l); in this case we find two subsets $V_{l-1}' \subset V_{l-1}$ and $V_l' \subset V_l$ with $|V_{l-1}'| \geq \delta_{l-1}^4 |V_{l-1}|$ and $|V_l'| \geq \delta_{l-1}^4 |V_l|$ such that

$$|d_{\omega_{l-1}}(V_{l-1}', V_l') - d_{\omega_{l-1}}(V_{l-1}, V_l)| > \delta_{l-1}^2.$$

Both V_{l-1}' and V_l' will be used as witness sets to improve P. More precisely, the following algorithm can be used to improve the partition P.

Algorithm Improve

1. For every l-tuple $(V_1, \dots, V_l)$ in the partition P do:

2. For $i = 1$ to $l-2$ successively

3. Apply the procedure from Theorem 3.1 to search for $X_i' \subset X_i = V_{i+1} \times \cdots \times V_l$ and $V_i' \subset V_i$ such that

$$|d_{\omega_i}(X_i', V_i') - d_{\omega_i}(X_i', V_i)| > \delta_i^2. \tag{5.5}$$

4. If (5.5) is satisfied for some $i \leq l-2$, then V_i' is a witness set and we move to the next l-tuple.

5. If there is no $i \leq l-2$ for which (5.5) holds, then we apply the procedure from Theorem 4.3 to search for $V_{l-1}' \subset V_{l-1}$ and $V_l' \subset V_l$ such that

$$|d_{\omega_{l-1}}(V_{l-1}', V_l') - d_{\omega_{l-1}}(V_{l-1}, V_l)| > \delta_{l-1}^2. \tag{5.6}$$

6. In the case of (5.6), both V_{l-1}' and V_l' are witness sets for the l-tuple.

7. If the number of l-tuples for which witness sets were found is at least ϵt^l, then compute a subpartition P' (described below) of P that respects all the witness sets found in steps 3 and 5. Otherwise the partition P is (ϵ, ω)-regular.

Let us conclude this section with the following fact which shows what sets were found by the algorithm **Improve**.

FACT 5.1. *If an l-tuple $(V_1,\dots,V_l)$ is (ϵ,ω)-irregular, then* **Improve** *either finds $X_k'\subset X_k$ and $V_k'\subset V_k$ with $|X_k'|\geq\frac{\delta_k^7}{2}|X_k|$ and $|V_k'|\geq\delta_k^6|V_k|$ such that* (5.5) *holds for some $1\leq k\leq l-2$, or it finds $V_{l-1}'\subset V_{l-1}$ and $V_l'\subset V_l$ with $|V_{l-1}'|\geq\delta_{l-1}^4|V_{l-1}|$ and $|V_l'|\geq\delta_l^4|V_l|$ such that* (5.6) *holds.*

Proof. For every $k=1,\dots,l-2$, Lemma 2.5 implies that if an $(l-k+1)$-tuple $(V_k,\dots,V_l)$ is (ϵ_k,ω_k)-irregular, then either the $(l-k)$-tuple $(V_{k+1},\dots,V_l)$ is $(\epsilon_{k+1},\omega_{k+1})$-irregular or at least $\delta_k|X_k|^2$ of the pairs $\{x_i,x_j\}$ of vertices from X_k satisfy

$$|\langle\overrightarrow{x_i},\overrightarrow{x_j}\rangle-K_k^2Md_id_j|>\delta_kK_k^2M. \tag{5.7}$$

If (5.7) is satisfied, then the algorithm of Theorem 3.1 finds $X_k'\subset X_k$ and $V_k'\subset V_k$ such that (5.5) holds. If (5.7) does not hold for any $1\leq k\leq l-2$, then a pair (V_{l-1},V_l) is $(\epsilon_{l-1},\omega_{l-1})$-irregular and the algorithm of Theorem 4.3 finds $V_{l-1}'\subset V_{l-1}$ and $V_l'\subset V_l$ such that (5.6) holds. □

6. The analysis of the main algorithm. In this section, we will analyze the algorithm **Improve**. Although a little bit technical, the philosophy of the analysis is the same as the Szemerédi's proof of the regularity lemma [12]. We will show that for a subpartition P' computed by the algorithm the value of the index $ind(P')$ is bigger than the value of the index of the original partition $ind(P)$. In addition, we will show that the size of the exceptional class does not increase in any significant way. The proof is divided into five rather technical facts. In Fact 6.1, we will show that the size of the exceptional class does not increase too much. Fact 6.2 shows that the value of the index associated with each (ϵ,ω)-regular l-tuple will remain about the same after the refinement of the original partition. In Fact 6.3, we will show that we get a "profit" on the index if an (ϵ,ω)-irregular l-tuple is reduced to the vector irregularity. Fact 6.4 shows that the index will increase in case an (ϵ,ω)-irregular l-tuple is reduced to the irregularity of the weighted bipartite graph. Lemma 6.5 combines Fact 6.3 and Fact 6.4 to show that the value of the index of the refined partition is greater than the value of the original one.

Similarly as in the original proof of the regularity lemma [12], we consider a subpartition P' of P into atoms of size

$$m=\left\lfloor\frac{|V_i|}{2^{2t^l}}\right\rfloor$$

that respects the Venn diagram of witness sets found for each (ϵ,ω)-irregular l-tuple. For $j=1,\dots,p$, denote by $W_i(j)$ the jth atom in V_i (that is, the jth subset in the partition of V_i), and let

$$\bar{V_i}=\bigcup_{j=1}^{p}W_i(j).$$

Observe that for every i

$$||\bar{V_i}|-|V_i||\leq\frac{|V_i|}{2^{t^l}}. \tag{6.1}$$

We first observe the following fact.

FACT 6.1. *The size of the exceptional class V_0' increases by at most $\frac{n}{2^{t^l}}$ from the size of V_0.*

Proof. Indeed, there are at most 2^{t^l} equivalence classes of atoms and so in the process of refining P we increase $|V_0|$ by at most

$$t2^{t^l}\frac{|V_i|}{2^{2t^l}} \le \frac{n}{2^{t^l}}. \qquad \square$$

FACT 6.2. *For every l-tuple $(V_1,\dots,V_l)$ and every $i \in [l]$*

$$\frac{1}{p^i}\sum_{j_1,\dots,j_i}\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x,W_1(j_1),\dots,W_i(j_i)))^2}{|\bar{V}_{i+1}|\dots|\bar{V}_l|} \ge ind_i(V_1,\dots,V_l)-\frac{3l}{2^{t^l}}.$$

Proof. For every $x \in V_{i+1}\times\cdots\times V_l$,

$$\frac{1}{p^i}\sum_{j_1,\dots,j_i}(d_\omega(x,W_1(j_1),\dots,W_i(j_i)))^2 \ge \left(\frac{1}{p^i}\sum_{j_1,\dots,j_i} d_\omega(x,W_1(j_1),\dots,W_i(j_i))\right)^2$$

$$= (d_\omega(x,\bar{V}_1,\dots,\bar{V}_i))^2.$$

Since $|\bar{V}_i| \ge (1-\frac{1}{2^{t^l}})|V_i|$, by the continuity of density (Fact 2.1) we have

$$(d_\omega(x,\bar{V}_1,\dots,\bar{V}_i))^2 \ge (d_\omega(x,V_1,\dots,V_i))^2-\frac{2l}{2^{t^l}}, \tag{6.2}$$

and so

$$\frac{1}{p^i}\sum_{j_1,\dots,j_i}\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x,W_1(j_1),\dots,W_i(j_i)))^2}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\ge \sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x,V_1,\dots,V_i))^2-\frac{2l}{2^{t^l}}}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\ge \sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|}-\frac{2l}{2^{t^l}}. \tag{6.3}$$

Using (6.1) and the fact that $d_\omega(x,V_1,\dots,V_i)\le 1$,

$$\sum_{x\in V_{i+1}\times\cdots\times V_l}(d_\omega(x,V_1,\dots,V_i))^2-\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}(d_\omega(x,V_1,\dots,V_i))^2$$

$$\le \sum_{k=i+1}^{l}\sum\{(d_\omega(x,V_1,\dots,V_i))^2, x\in V_{i+1}\times\cdots\times V_{k-1}\times(V_k\setminus\bar{V}_k)\times V_{k+1}\times\cdots\times V_l\}$$

$$\le \frac{l}{2^{t^l}}|V_{i+1}|\dots|V_l|. \tag{6.4}$$

By (6.3) and (6.4),

$$\frac{1}{p^i}\sum_{j_1,\dots,j_i}\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x,W_1(j_1),\dots,W_i(j_i)))^2}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\geq\sum_{x\in V_{i+1}\times\cdots\times V_l}\frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|}-\frac{3l}{2^{t^l}}$$

$$=ind_i(V_1,\dots,V_l)-\frac{3l}{2^{t^l}}.\qquad\square$$

Suppose that an l-tuple $(V_1,\dots,V_l)$ is (ϵ,ω)-irregular and that for some $1\leq i\leq l-2$, we found two sets $V_i'\subset V_i$ with $|V_i'|>\delta_i^6|V_i|$ and $X_i'\subset X_i$ with $|X_i'|>\frac{\delta_i^7}{2}|X_i|$ such that

$$|d_{\omega_i}(X_i',V_i')-d_{\omega_i}(X_i',V_i)|>\delta_i^2. \tag{6.5}$$

Then we have the following fact.

FACT 6.3. *Assume that an l-tuple $(V_1,\dots,V_l)$ is (ϵ,ω)-irregular and for some $1\leq i\leq l-1$, we found two sets $V_i'\subset V_i$ with $|V_i'|>\delta_i^6|V_i|$ and $X_i'\subset X_i$ with $|X_i'|>\frac{\delta_i^7}{2}|X_i|$ such that $|d_{\omega_i}(X_i',V_i')-d_{\omega_i}(X_i',V_i)|>\delta_i^2$. If $\delta_i^8\geq\frac{l}{2^{t^l-2}}$, then*

$$\frac{1}{p^i}\sum_{j_1,\dots,j_i}\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x,W_1(j_1),\dots,W_i(j_i)))^2}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\geq ind_i(V_1,\dots,V_l)+\frac{\delta_i^{17}}{8(1-\delta_i^6)}-\frac{4l}{2^{t^l}}.$$

Proof. First observe that for $Z\subset X_i=V_{i+1}\times\cdots\times V_l$, $Y\subset V_i$

$$d_{\omega_i}(Z,Y)=\frac{\sum_{z\in Z,y\in Y}\omega_i(y,z)}{K_i|Z||Y|}=\frac{1}{|Z|}\sum_{z\in Z}\frac{\sum_{y\in Y}\sum_{v_1,\dots,v_{i-1}}\omega(v_1,\dots,v_{i-1},y,z)}{K|V_1|\dots|V_{i-1}||Y|}$$

$$=\frac{1}{|Z|}\sum_{z\in Z}d_\omega(z,V_1,\dots,V_{i-1},Y). \tag{6.6}$$

We may assume that for the witness V_i',

$$\bar{V}_i'=\bigcup_{j_i=1}^{q}W_i(j_i).$$

Then

$$||V_i'|-|\bar{V}_i'||\leq\frac{|V_i'|}{2^{t^l}\delta_i^6}.$$

For every $x \in X_i$, define

$$\alpha_x = \frac{1}{p^i} \sum_{j_1,\ldots,j_i \in [p]} d_\omega(x, W_1(j_1), \ldots, W_i(j_i))$$

$$-\frac{1}{qp^{i-1}} \sum_{j_1\ldots,j_{i-1}\in[p]} \sum_{j_i\in[q]} d_\omega(x, W_1(j_1), \ldots, W_i(j_i)).$$

Observe that

$$\sum_{j_1\ldots,j_{i-1}\in[p]} \sum_{j_i\in[q]} d_\omega(x, W_1(j_1), \ldots, W_i(j_i))$$

$$= \frac{q}{p} \sum_{j_1,\ldots,j_i\in[p]} d_\omega(x, W_1(j_1), \ldots, W_i(j_i)) - qp^{i-1}\alpha_x,$$

and using the defect form of Schwarz inequality (Fact 2.2), we infer that

$$\frac{1}{p^i} \sum_{j_1,\ldots,j_i} (d_\omega(x, W_1(j_1), \ldots, W_i(j_i)))^2$$

$$\geq \left(\frac{1}{p^i} \sum_{j_1\ldots,j_i} d_\omega(x, W_1(j_1), \ldots, W_i(j_i))\right)^2 + \frac{(\alpha_x qp^{i-1})^2 p^i}{p^i qp^{i-1}(p^i - qp^{i-1})}$$

$$= (d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i))^2 + \frac{\alpha_x^2 q}{p-q},$$

due to the fact that $\sum_{j_1,\ldots,j_i} d_\omega(x, W_1(j_1), \ldots, W_i(j_i)) = d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i)$ and since $q \geq \delta_i^6 p$, we have

$$\frac{1}{p^i} \sum_{j_1,\ldots,j_i} (d_\omega(x, W_1(j_1), \ldots, W_i(j_i)))^2 \geq (d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i))^2 + \alpha_x^2 \frac{\delta_i^6}{1-\delta_i^6}. \tag{6.7}$$

Since $|\bar{V}_i| \geq (1 - \frac{1}{2^{t^l}})|V_i|$ and $|\bar{V}_i'| \geq (1 - \frac{1}{2^{t^l}\delta_i^6})|V_i'|$, by the continuity of density (Fact 2.1), we have

$$d_\omega(x, V_1, \ldots, V_i) - d_\omega(x, V_1, \ldots, V_{i-1}, V_i') \leq d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i) - d_\omega(x, \bar{V}_1, \ldots, \bar{V}_{i-1}, \bar{V}_i')$$

$$+|d_\omega(x, V_1, \ldots, V_i) - d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i)| + |d_\omega(x, \bar{V}_1, \ldots, \bar{V}_{i-1}, \bar{V}_i') - d_\omega(x, V_1, \ldots, V_{i-1}, V_i')|$$

$$\leq d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i) - d_\omega(x, \bar{V}_1, \ldots, \bar{V}_{i-1}, \bar{V}_i') + \frac{2l}{2^{t^l}\delta_i^6} = \alpha_x + \frac{2l}{2^{t^l}\delta_i^6}, \tag{6.8}$$

as $\alpha_x = d_\omega(x, \bar{V}_1, \ldots, \bar{V}_i) - d_\omega(x, \bar{V}_1, \ldots, \bar{V}_{i-1}, \bar{V}_i')$.

Applying (6.6) with $Z = X_i'$, $Y = V_i$ (or V_i', respectively) combined with (6.8) yields

$$|d_{\omega_i}(X_i', V_i) - d_{\omega_i}(X_i', V_i')| = \frac{1}{|X_i'|}\left|\sum_{x\in X_i'}(d_\omega(x, V_1, \dots, V_i) - d_\omega(x, V_1, \dots, V_{i-1}, V_i')\right|$$

$$\leq \frac{1}{|X_i'|}\left|\sum_{x\in X_i'}\alpha_x\right| + \frac{2l}{2^{t^l}\delta_i^6}.$$

Since $|d_{\omega_i}(X_i', V_i) - d_{\omega_i}(X_i', V_i')| \geq \delta_i^2$, we have

$$\frac{1}{|X_i'|}\left|\sum_{x\in X_i'}\alpha_x\right| \geq \delta_i^2 - \frac{2l}{2^{t^l}\delta_i^6}. \tag{6.9}$$

Thus,

$$\frac{1}{|X_i|}\sum_{x\in X_i}\alpha_x^2 \geq \frac{1}{|X_i|}\sum_{x\in X_i'}\alpha_x^2 \geq \frac{1}{|X_i||X_i'|}\left(\sum_{x\in X_i'}\alpha_x\right)^2 \geq \frac{|X_i'|}{|X_i|}\left(\delta_i^2 - \frac{2l}{2^{t^l}\delta_i^6}\right)^2. \tag{6.10}$$

Since $|X_i'| \geq \frac{\delta_i^7}{2}|X_i|$ and $\delta_i^2 - \frac{2l}{2^{t^l}\delta_i^6} \geq \frac{\delta_i^2}{2}$,

$$\frac{1}{|X_i|}\sum_{x\in X_i}\alpha_x^2 \geq \frac{\delta_i^{11}}{8}. \tag{6.11}$$

Therefore, by (6.7),

$$\frac{1}{p^i}\sum_{j_1,\dots,j_i}\sum_{x\in \bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x, W_1(j_1), \dots, W_i(j_i)))^2}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\geq \sum_{x\in \bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x, \bar{V}_1, \dots, \bar{V}_i))^2 + \alpha_x^2\frac{\delta_i^6}{1-\delta_i^6}}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}.$$

By (6.2),

$$\sum_{x\in \bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x, \bar{V}_1, \dots, \bar{V}_i))^2 + \alpha_x^2\frac{\delta_i^6}{1-\delta_i^6}}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\geq \sum_{x\in \bar{V}_{i+1}\times\cdots\times\bar{V}_l}\frac{(d_\omega(x, V_1, \dots, V_i))^2 + \alpha_x^2\frac{\delta_i^6}{1-\delta_i^6} - \frac{2l}{2^{t^l}}}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}.$$

Since $|\bar{V}_j| \le |V_j|$, we have

$$\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l} \frac{(d_\omega(x,V_1,\dots,V_i))^2+\alpha_x^2\frac{\delta_i^6}{1-\delta_i^6}-\frac{2l}{2^{t^l}}}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\ge \sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l} \frac{(d_\omega(x,V_1,\dots,V_i))^2+\alpha_x^2\frac{\delta_i^6}{1-\delta_i^6}-\frac{2l}{2^{t^l}}}{|V_{i+1}|\dots|V_l|}$$

$$= \sum_{x\in\bar{V}_{i+1}\times\ldots\times\bar{V}_l} \left(\frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|}+\frac{\delta_i^6}{1-\delta_i^6}\frac{\alpha_x^2}{|X_i|}\right)-\frac{2l}{2^{t^l}}.$$

Clearly $d_\omega(x,V_1,\dots,V_i)\le 1$ and $\frac{\delta_i^6}{1-\delta_i^6}\alpha_x \le 1$. Thus, using the same argument as in (6.4) we have

$$\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l} \frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|} \ge \sum_{x\in V_{i+1}\times\cdots\times V_l} \frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|}-\frac{l}{2^{t^l}}$$

and

$$\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l} \frac{\delta_i^6}{1-\delta_i^6}\frac{\alpha_x^2}{|X_i|} \ge \sum_{x\in V_{i+1}\times\cdots\times V_l} \frac{\delta_i^6}{1-\delta_i^6}\frac{\alpha_x^2}{|X_i|}-\frac{l}{2^{t^l}}.$$

Therefore,

$$\sum_{x\in\bar{V}_{i+1}\times\cdots\times\bar{V}_l} \frac{(d_\omega(x,V_1,\dots,V_i))^2+\alpha_x^2\frac{\delta_i^6}{1-\delta_i^6}-\frac{2l}{2^{t^l}}}{|\bar{V}_{i+1}|\dots|\bar{V}_l|}$$

$$\ge \sum_{x\in V_{i+1}\times\cdots\times V_l} \left(\frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|}+\frac{\delta_i^6}{1-\delta_i^6}\frac{\alpha_x^2}{|X_i|}\right)-\frac{4l}{2^{t^l}}. \tag{6.12}$$

Using (6.11) we further estimate the right-hand side of (6.12) from below by

$$\sum_{x\in V_{i+1}\times\cdots\times V_l} \frac{(d_\omega(x,V_1,\dots,V_i))^2}{|V_{i+1}|\dots|V_l|}+\frac{\delta_i^{17}}{8(1-\delta_i^6)}-\frac{4l}{2^{t^l}}$$
$$= ind_i(V_1,\dots,V_l)+\frac{\delta_i^{17}}{8(1-\delta_i^6)}-\frac{4l}{2^{t^l}}. \qquad \square$$

FACT 6.4. *For an $0<\epsilon<1$ let $\delta=\frac{1}{96}\frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}}$. Suppose that an l-tuple is (ϵ,ω)-irregular and we found sets $V'_{l-1}\subset V_{l-1}$ with $|V'_{l-1}|\ge\delta_{l-1}^4|V_{l-1}|$ and $V'_l\subset V_l$ with $|V'_l|\ge\delta_{l-1}^4|V_l|$ such that*

$$|d_{\omega_{l-1}}(V'_{l-1},V'_l)-d_{\omega_{l-1}}(V_{l-1},V_l)|>\delta_{l-1}^2.$$

Then either

$$\frac{1}{p^{l-1}}\sum_{j_1,\dots,j_{l-1}}\sum_{x\in\bar{V}_l}\frac{(d_\omega(x,W_1(j_1),\dots,W_{l-1}(j_{l-1})))^2}{|\bar{V}_l|}$$

$$\geq ind_{l-1}(V_1,\dots,V_l)+\frac{\delta^{17}}{8(1-\delta^6)}-\frac{4l}{2^{t^l}},$$

or

$$\frac{1}{p^{l-1}}\sum_{j_1,\dots,j_{l-2},j_l}\sum_{x\in\bar{V}_{l-1}}\frac{(d_\omega(x,W_1(j_1),\dots,W_{l-2}(j_{l-2}),W_l(j_l)))^2}{|\bar{V}_{l-1}|}$$

$$\geq ind_l(V_1,\dots,V_l)+\frac{\delta^{17}}{8(1-\delta^6)}-\frac{4l}{2^{t^l}}.$$

Proof. If

$$|d_{\omega_{l-1}}(V'_{l-1},V'_l)-d_{\omega_{l-1}}(V_{l-1},V_l)|>\delta^2_{l-1}, \tag{6.13}$$

then either

$$|d_{\omega_{l-1}}(V'_{l-1},V_l)-d_{\omega_{l-1}}(V_{l-1},V_l)|>\frac{\delta^2_{l-1}}{2}, \tag{6.14}$$

or

$$|d_{\omega_{l-1}}(V'_{l-1},V'_l)-d_{\omega_{l-1}}(V'_{l-1},V_l)|>\frac{\delta^2_{l-1}}{2}. \tag{6.15}$$

Set $\delta=\frac{1}{96}\frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}}$ and note that by (5.2), $\frac{\delta^2_{l-1}}{2}\geq\delta^2$.

In the case of (6.14), we apply Fact 6.3 to $(V_1,\dots,V_{l-2},V_{l-1},V_l)$ with $i=l-1$, $X'_i=V_l$, and $V'_i=V'_{l-1}$ to conclude that

$$\frac{1}{p^{l-1}}\sum_{j_1,\dots,j_{l-1}}\sum_{x\in\bar{V}_l}\frac{(d_\omega(x,W_1(j_1),\dots,W_{l-1}(j_{l-1})))^2}{|\bar{V}_l|}$$

$$\geq ind_{l-1}(V_1,\dots,V_l)+\frac{\delta^{17}}{8(1-\delta^6)}-\frac{4l}{2^{t^l}}.$$

In the case of (6.15), we apply Fact 6.3 to $(V_1,\dots,V_l,V_{l-1})$ with $i=l-1$, $X'_i=V'_{l-1}$, and $V'_i=V'_l$ to conclude that

$$\frac{1}{p^{l-1}}\sum_{j_1,\dots,j_{l-2},j_l}\sum_{x\in\bar{V}_{l-1}}\frac{(d_\omega(x,W_1(j_1),\dots,W_{l-2}(j_{l-2}),W_l(j_l)))^2}{|\bar{V}_{l-1}|}$$

$$\geq ind_{l-1}(V_1,\dots,V_{l-2},V_l,V_{l-1})+\frac{\delta^{17}}{8(1-\delta^6)}-\frac{4l}{2^{t^l}},$$

which by (1.5) is equal to

$$ind_l(V_1,\dots,V_{l-1},V_l)+\frac{\delta^{17}}{8(1-\delta^6)}-\frac{4l}{2^{t^l}}. \qquad \square$$

We have the following lemma.

LEMMA 6.5. *For every $\epsilon \le 1/2$ and $\delta = \frac{1}{96}\frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}}$, let $P = V_0 \cup V_1 \cup \cdots \cup V_t$ be an (ϵ,ω)-irregular partition of an l-uniform hypergraph and let $\frac{4l}{2^{t^l}} \le \frac{\delta^{17}}{16\cdot l}$. If we use algorithm* **Improve** *to construct the subpartition P', then*

$$ind(P') \ge ind(P)+\frac{\delta^{17}}{16\cdot l}.$$

Proof. Assume that each V_i has been partitioned into $W_i(j)$ for $j=1,\dots,p$, and let P' be the resulting subpartition. Then

$$P' = W_0 \cup W_1(1)\cup\cdots\cup W_1(p)\cup\cdots\cup W_t(1)\cup\cdots\cup W_t(p).$$

To simplify the notation we will also write

$$P' = W_0 \cup W_1 \cup \cdots \cup W_{t'},$$

where $t' = tp$. Then

$$ind(P') = \frac{1}{l(tp)^l}\sum_{(W_{i_1},\dots,W_{i_l})}\left(\sum_{k=1}^{l} ind_k(W_{i_1},\dots,W_{i_l})\right),$$

and so

$$ind(P') \ge \frac{1}{lt^l}\sum_{(i_1,\dots,i_l)\in[t]^l}\left(\frac{1}{p^l}\sum_{j_1\dots j_l\in[p]}\left(\sum_{k=1}^{l} ind_k(W_{i_1}(j_1),\dots,W_{i_l}(j_l))\right)\right).$$

For every $1\le k\le l$,

$$\frac{1}{p^l}\sum_{j_1,\dots,j_l\in[p]} ind_k(W_{i_1}(j_1),\dots,W_{i_l}(j_l))$$

$$= \frac{1}{p^l}\sum_{j_1,\dots,j_l\in[p]}\ \sum_{x\in W_{i_{k+1}}(j_{k+1})\times\cdots\times W_{i_l}(j_l)} \frac{(d_\omega(x,W_{i_1}(j_1),\dots,W_{i_k}(j_k)))^2}{|W_{i_{k+1}}(j_{k+1})|\dots|W_{i_l}(j_l)|}$$

$$= \frac{1}{p^l}\sum_{j_1,\dots,j_k}\ \sum_{x\in \bar V_{i_{k+1}}\times\cdots\times\bar V_{i_l}} \frac{(d_\omega(x,W_{i_1}(j_1),\dots,W_{i_k}(j_k)))^2}{|W_{i_{k+1}}(j_{k+1})|\dots|W_{i_l}(j_l)|}$$

$$= \frac{1}{p^k}\sum_{j_1,\dots,j_k}\ \sum_{x\in \bar V_{i_{k+1}}\times\dots\times\bar V_{i_l}} \frac{(d_\omega(x,W_{i_1}(j_1),\dots,W_{i_k}(j_k)))^2}{|\bar V_{i_{k+1}}|\dots|\bar V_l|}, \tag{6.16}$$

as $|\bar V_{i_l}| = p|W_{i_l}(j_l)|$. Fact 6.2 implies that for every $1\le k\le l$,

$$\frac{1}{p^l}\sum_{j_1,\dots,j_l\in[p]} ind_k(W_{i_1}(j_1),\dots,W_{i_l}(j_l)) \ge ind_k(V_{i_1},\dots,V_{i_l})-\frac{2l}{2^{t^l}}.$$

If an l-tuple $(V_{i_1},\dots,V_{i_l})$ is (ϵ,ω)-irregular, then by Fact 5.1, we can find either

- $V'_{k_0} \subset V_{k_0}$ and $X'_{k_0} \subset X_{k_0}$ that satisfy the assumptions of Fact 6.3 for some $1 \leq k_0 \leq l-2$, or
- $V'_{l-1} \subset V_{l-1}$ and $V'_l \subset V_l$ that satisfy the assumptions of Fact 6.4.

If the former holds, we combine (6.16) and Fact 3.6 to infer that for some $1 \leq k_0 \leq l-2$,

$$\frac{1}{p^l} \sum_{j_1,\ldots,j_l \in [p]} ind_{k_0}(W_{i_1}(j_1), \ldots, W_{i_l}(j_l)) \geq ind_{k_0}(V_{i_1}, \ldots, V_{i_l}) + \frac{\delta_{k_0}^{17}}{8(1-\delta_{k_0}^6)} - \frac{4l}{2^{t^l}}.$$

If the latter holds, then by Fact 6.4 for $k_0 = l-1$ or for $k_0 = l$,

$$\frac{1}{p^l} \sum_{j_1,\ldots,j_l \in [p]} ind_{k_0}(W_{i_1}(j_1), \ldots, W_{i_l}(j_l)) \geq ind_{k_0}(V_{i_1}, \ldots, V_{i_l}) + \frac{\delta^{17}}{8(1-\delta^6)} - \frac{4l}{2^{t^l}}.$$

Since the partition P is (ϵ, ω)-irregular, at least ϵt^l of l-tuples $(V_{i_1}, \ldots, V_{i_l})$ are (ϵ, ω)-irregular. Thus, (by (5.2) $\delta_{k_0} \geq \frac{1}{48} \frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}} > \delta$, $1 - \delta \leq 1$)

$$ind(P') \geq ind(P) + \frac{\delta^{17}}{8 \cdot l} - \frac{4l}{2^{t^l}},$$

and since t satisfies $\frac{4l}{2^{t^l}} \leq \frac{\delta^{17}}{16 \cdot l}$ by assumption, we have

$$ind(P') \geq ind(P) + \frac{\delta^{17}}{16 \cdot l}. \qquad \square$$

Proof of Theorem 1.1. Set $\delta = \frac{1}{96} \frac{\epsilon^{2l+1}}{2^{(2l+1)(l-1)}}$, and partition the vertex set of a hypergraph into t subsets (arbitrarily), but such that

$$\frac{1}{2^{t^l}} \leq \frac{\epsilon \delta^{17}}{16 \cdot l + \delta^{17}}.$$

Invoke the procedure **Improve** $\frac{16 \cdot l}{\delta^{17}} + 1$ times. By Lemma 6.5 (note that $\frac{1}{2^{t^l}} \leq \frac{\delta^{17}}{16 \cdot l}$) after at most $\frac{16 \cdot l}{\delta^{17}} + 1$ iterations we find a partition Q with less than ϵt^l (ϵ, ω)-irregular l-tuples, otherwise $ind(Q) > 1$, which is not possible. Also, by Fact 6.1 the size of the exceptional class

$$|V_0| \leq \left(\frac{16 \cdot l}{\delta^{17}} + 1 \right) \frac{n}{2^{t^l}} \leq \epsilon n.$$

Next we will argue that the complexity of the algorithm is $O(n^{2l-1} \log^2 n)$. First observe that we iterate the algorithm **Improve** a constant number of times. Since l is constant and the number of partition classes is constant, the complexity of **Improve** depends only on the algorithms of Theorem 3.1 and Theorem 4.3. Recall that the complexity of the algorithm of Theorem 3.1 is $O(N^2 M \log^2 K_k)$, where $N = |V_{k+1} \times \cdots \times V_l|$ and $M = |V_k|$. Since $1 \leq k \leq l-1$ and by (5.1), $K_k \leq K n^l$ (K is constant by an assumption), the complexity of the algorithm from Theorem 3.1 is $O(n^{2l-1} \log^2 n)$. The complexity of the algorithm of Theorem 4.3 is $O(M^3 \log^2 K_{l-1})$, where $M = |V_{l-1}| = |V_l|$. Therefore, in this case, the complexity is $O(n^3 \log^2 n)$. The total complexity of **Improve** is $O(n^{2l-1} \log^2 n)$. $\square$

7. Applications. In this section, we outline the applications of Theorem 1.1 to the max-cut problem for hypergraphs and to the problem of estimating the chromatic number of a hypergraph. Let $H = (V, E)$ be an l-uniform hypergraph and let $n = |V|$. We consider the unweighted case $\omega : [V]^l \to \{0, 1\}$, and to simplify the notation we write $d(V_1, \dots, V_l)$ instead of $d_\omega(V_1, \dots, V_l)$. In the max-cut problem one wants to find a partition of V into l subsets which is such that the number of hyperedges that intersect each partition class (have nonempty intersection with each class) is maximized. Case $l = 2$ gives the max-cut problem for graphs and its "dense case" was considered in [7]. Let $OPT(H) = max|\{e \in E : |e \cap V_i| = 1; i = 1, \dots, l\}|$, where the maximum is taken over all partitions $V_1 \cup \dots \cup V_l$ of V.

THEOREM 7.1. *Let $H = (V, E)$ be an l-uniform hypergraph and let $n = |V|$. For every $\epsilon > 0$, there is an $O(n^{2l-1} \log^2 n)$ algorithm that finds a partition $V_1 \cup \dots \cup V_l$ of V which is such that the number of hyperedges that intersect each V_i, $i = 1, \dots, l$ is at least $OPT(H) - \epsilon n^l$.*

Proof. The following algorithm finds the postulated partition. The constant ϵ' depends on ϵ and can be computed explicitly.

1. Find an ϵ'-regular partition of H: $W_0, \dots, W_t$.
2. Check exhaustively all partitions $V_1, \dots, V_l$ in which for every $i \in [l]$ and every $j \in [t]$ we have if $W_j \cap V_i \neq \emptyset$, then $W_j \subset V_i$. Choose a partition $V_1, \dots, V_l$ that maximizes

$$\sum_{W_{j_k} \subset V_k} d(W_{k_1}, W_{k_2}, \dots, W_{k_l})|W_{k_1}| \dots |W_{k_l}|.$$

Note that since there are l^t partitions that are checked in the second step of the algorithm, the complexity of the procedure is $O(n^{2l-1} \log^2 n)$.

For a partition $U_1, \dots, U_l$ of V define

$$f(U_1, \dots, U_l) = \max|\{e \in E : |e \cap U_i| = 1, i = 1, \dots, l\}|$$

and

$$f^*(U_1, \dots, U_l) = \sum_{W_{j_k} \subset U_k} d(W_{j_1}, W_{j_2}, \dots, W_{j_l})|U_1 \cap W_{k_1}| \dots |U_l \cap W_{k_l}|.$$

One can verify that f^* is maximized for a partition $U_1, \dots, U_l$ which is of the form considered in the second step of the algorithm, i.e., if $W_j \cap U_i \neq \emptyset$, then $W_j \subset U_i$. Also, choosing ϵ' appropriately, one can show that for every partition $U_1, \dots, U_l$

$$|f(U_1, \dots, U_l) - f^*(U_1, \dots, U_l)| \leq \frac{\epsilon}{2} n^l.$$

Let $V_1, \dots, V_l$ be a partition found by the algorithm, and let $U_1, \dots, U_l$ be an optimal partition. Then

$$f(V_1, \dots, V_l) \geq f^*(V_1, \dots, V_l) - \frac{\epsilon}{2} n^l \geq f^*(U_1, \dots, U_l) - \frac{\epsilon}{2} n^l$$

$$\geq f(U_1, \dots, U_l) - \epsilon n^l = OPT(H) - \epsilon n^l. \qquad \square$$

Our second application concerns the chromatic number of a hypergraph. The chromatic number $\chi(H)$ is defined as the minimum number of colors needed to color the

vertices of H so that there is no hyperedge of H that contains more than one vertex of the same color. Define

$$\chi_\epsilon(H) = \min\{\chi(H \setminus E') : E' \subset E; |E'| \le \epsilon n^l\}.$$

We define a hyperedge $\{v_1, \dots, v_l\}$ to be crossing in an l-tuple $(V_1, \dots, V_l)$ if for every $i = 1, \dots, l$, $v_i \in V_i$.

THEOREM 7.2. *Let $H = (V, E)$ be an l-uniform hypergraph and let $n = |V|$. For every $\epsilon > 0$ there is an $O(n^{2l-1} \log^2 n)$ time algorithm that finds a number k satisfying*

$$\chi_\epsilon(H) \le k \le \chi(H).$$

Proof. Sketch. Set $\epsilon' = \frac{\epsilon}{4}$ and find an ϵ'-regular partition $W_0, \dots, W_t$. Construct a subhypergraph H' of H by deleting all the hyperedges adjacent to W_0, hyperedges that are crossing in ϵ'-irregular l-tuples, and the hyperedges that are crossing in the ϵ'-regular l-tuples that have densities not greater than ϵ'. In this process we delete at most $3\epsilon' n^l$ hyperedges. Construct a subhypergraph H'' of H' as follows. Group (arbitrarily) classes $W_0, W_1, \dots, W_t$ into $t' = \frac{1}{\epsilon'}$ sets $V_1, \dots, V_{t'}$ (say, $V_i = \bigcup_{j=(i-1)\epsilon' t}^{i\epsilon' t} W_j$), and delete the hyperedges that contain at least two vertices from the same V_i. In this process we delete at most $\epsilon' n^l$ hyperedges. Let $Aux(H'')$ be an l-uniform hypergraph with vertex set $\{W_0, \dots, W_t\}$ and with $\{W_{i_1}, \dots, W_{i_l}\} \in E(Aux(H''))$ if and only if there is at least one hyperedge of H'' contained in $W_{i_1} \cup \dots \cup W_{i_l}$.

CLAIM 7.3. $\chi(Aux(H'')) = \chi(H'')$.

Proof. Clearly, $\chi(Aux(H'')) \ge \chi(H'')$, as a proper coloring of $Aux(H'')$ induces the proper coloring of H''. Assume that $\chi(H'') < \chi(Aux(H''))$. Let $\bar{W}_i \subset W_i$ be a set of vertices colored in the most frequent color of W_i in a $\chi(H'')$-coloring of H'' (ties are resolved arbitrarily). Consider the coloring of $Aux(H'')$ induced by these "most frequent" colors. Then there exists a hyperedge of $Aux(H'')$, $\{W_{i_1}, \dots, W_{i_l}\}$ such that at least two of W_{i_j} have the same color. We next show that there must be a crossing hyperedge in $(\bar{W}_{i_1}, \dots, \bar{W}_{i_l})$. From the construction of H'' it follows that $\chi(Aux(H'')) \le \frac{1}{\epsilon'}$, and so for every $i = 1, \dots, t$

$$|\bar{W}_i| \ge \epsilon' |W_i|. \tag{7.1}$$

Since $\{W_{i_1}, \dots, W_{i_l}\}$ is a hyperedge of $Aux(H'')$, we have $d(W_{i_1}, \dots, W_{i_l}) > \epsilon'$, and by the ϵ'-regularity of $(W_{i_1}, \dots, W_{i_l})$

$$d(\bar{W}_{i_1}, \dots, \bar{W}_{i_l}) > 0. \tag{7.2}$$

Therefore, there is at least one crossing hyperedge in $(\bar{W}_{i_1}, \dots, \bar{W}_{i_l})$ which contradicts the fact that H'' was properly colored. ∎

Since $Aux(H'')$ has t vertices, $\chi(Aux(H''))$ can be found in a constant time by exhaustive search, and so we found $k = \chi(Aux(H'')) = \chi(H'')$ satisfying $k \ge \chi_\epsilon(H)$. ∎

Acknowledgments. We would like to thank referees for helpful comments and suggestions.

REFERENCES

[1] N. Alon, R.A. Duke, H. Lefmann, V. Rödl, and R. Yuster, *The algorithmic aspects of the regularity lemma*, J. Algorithms, 16 (1994), pp. 80–109.
[2] F.R.K. Chung, *Regularity lemmas for hypergraphs and quasi-randomness*, Random Structures Algorithms, 2 (1991), pp. 241–252.
[3] A. Czygrinow, S. Poljak, and V. Rödl, *Constructive quasi-Ramsey numbers and tournament ranking*, SIAM J. Discrete Math., 12 (1999), pp. 48–63.
[4] R.A. Duke, H. Lefmann, and V. Rödl, *A fast approximation algorithm for computing the frequencies of subgraphs in a given graph*, SIAM J. Comput., 24 (1995), pp. 598–620.
[5] P. Frankl and V. Rödl, *The uniformity lemma for hypergraphs*, Graphs Combin., 8 (1992), pp. 309–312.
[6] P. Frankl and V. Rödl, *Extremal problems with no exponent*, Random Structures Algorithms, submitted.
[7] A. Frieze and R. Kannan, *The regularity lemma and approximation schemes for dense problems*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, Burlington, VT, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 12–20.
[8] A. Frieze and R. Kannan, *Quick approximation to matrices and applications*, Combinatorica, 19 (1999), pp. 175–220.
[9] A. Frieze and R. Kannan, *A simple algorithm for constructing Szemerédi's regularity partition*, Electron. J. Combin., 6 (1999), http://www.combinatorics.org./Volume-6/PDF/v6i1r1.pdf.
[10] B. Nagle and V. Rödl, *The asymptotic number of triple systems not containing a fixed one*, to appear in Proceedings of the 5th Czech-Slovak International Symposium on Combinatorics, Graph Theory, Algorithms and Applications, DIMATIA, Prague, 1998.
[11] H. J. Prömel and A. Steger, *Excluding induced subgraphs* III; *a general asymptotic*, Random Structures Algorithms, 3 (1994), pp. 19–31.
[12] E. Szemerédi, *Regular partitions of graphs*, in Problemes Combinatories et Theorie des Graphes, Colloq. Internat. CNRS 260, Paris, 1978, pp. 399–401.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1067–1079

A POLYNOMIAL APPROXIMATION ALGORITHM FOR THE MINIMUM FILL-IN PROBLEM*

ASSAF NATANZON†, RON SHAMIR†, AND RODED SHARAN†

Abstract. In the *minimum fill-in* problem, one wishes to find a set of edges of smallest size, whose addition to a given graph will make it chordal. The problem has important applications in numerical algebra and has been studied intensively since the 1970s. We give the first polynomial approximation algorithm for the problem. Our algorithm constructs a triangulation whose size is at most eight times the optimum size squared. The algorithm builds on the recent parameterized algorithm of Kaplan, Shamir, and Tarjan for the same problem.

For bounded degree graphs we give a polynomial approximation algorithm with a polylogarithmic approximation ratio. We also improve the parameterized algorithm.

Key words. approximation algorithms, parameterized algorithms, graph algorithms, minimum fill-in, chordal graphs, chain graphs, chordal completion, chain completion

AMS subject classifications. 68W25, 68W99, 05C85, 05C99

PII. S0097539798336073

1. Introduction.

A *chord* in a cycle is an edge between nonconsecutive vertices on the cycle. A *chordless cycle* is a cycle of length greater than 3 that contains no chord. A graph is called *chordal* or *triangulated* if it contains no chordless cycle. If $G = (V, E)$ is not chordal and F is a set of edges such that $(V, E \cup F)$ is chordal, then F is called a *fill-in* or a *triangulation* of G. If $|F| \leq k$, then F is called a *k-triangulation* of G. We denote by $\Phi(G)$ the size of the smallest fill-in of G.

The *minimum fill-in* problem is to find a minimum triangulation (fill-in) of a given graph. The importance of the problem stems from its applications to numerical algebra. In many fields, including VLSI simulation, solution of linear programs, signal processing, and others (cf. [7]), one has to perform a Gaussian elimination on a sparse symmetric positive-definite matrix. During the elimination process zero entries may become nonzeros. Different elimination orders may introduce different sets of new nonzero elements into the matrix. The time of the computation and its storage needs are dependent on the sparseness of the matrix. It is therefore desirable to find an elimination order such that a minimum number of zero entries is filled in with nonzeros (even temporarily). Rose [21] proved that the problem of finding an elimination order for a symmetric positive-definite matrix M, such that fewest new nonzero elements are introduced, is equivalent to the minimum fill-in problem on a graph whose vertices correspond to the rows of M and in which (i, j) is an edge if and only if $M_{i,j} \neq 0$.

In 1979, Garey and Johnson [9] posed the complexity of the minimum fill-in problem as a major open problem. Yannakakis subsequently proved that the minimum fill-in problem is NP-complete [23]. Due to its importance the problem has been studied intensively [2, 11, 13, 22], and many heuristics have been developed for it [5, 12, 20, 21]. None of those gives a performance guarantee with respect to the size

*Received by the editors March 23, 1998; accepted for publication (in revised form) March 22, 2000; published electronically August 29, 2000. Portions of this paper appeared in the Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998 [18].

http://www.siam.org/journals/sicomp/30-4/33607.html

†Department of Computer Science, Tel Aviv University, Tel Aviv, Israel (natanzon@math.tau.ac.il, shamir@math.tau.ac.il, roded@math.tau.ac.il). The research of the second author was supported by a grant from the Ministry of Science, Israel. The research of the third author was supported by an Eshkol Fellowship from the Ministry of Science, Israel.

of the fill-in introduced. Note that in contrast, the *minimal* fill-in problem (finding a triangulation of G which is minimal with respect to inclusion) is polynomial [19].

Approximation attempts succeeded only for the related *minimum triangulated supergraph* problem (MTS). In MTS the goal is to add edges to the input graph in order to obtain a chordal graph with minimum *total* number of edges. While as optimization problems MTS and minimum fill-in are equivalent, they may differ drastically as approximation problems. For example, if the input graph has $\Omega(n^2)$ edges and fill-in of size $o(n^2)$, then one can trivially achieve a constant approximation ratio for MTS by making the graph an n-clique (a complete graph), while no such approximation guarantee exists for the minimum fill-in problem. (Throughout we use n and m to denote the number of vertices and edges, respectively, in a graph.) The approximation results regarding MTS use the nested dissection heuristic first proposed by George [10] (see [13] for details). Gilbert [14] showed that for a graph with maximum degree d there exists a balanced separator decomposition such that a nested dissection ordering based on that decomposition yields a chordal supergraph, in which the number of edges is within a factor of $O(d \log n)$ of optimal. The result was not constructive as one has yet to find such a decomposition. Leighton and Rao [17] gave a polynomial approximation algorithm for finding a balanced separator in a graph of size within a factor of $O(\log n)$ of optimal. Agrawal, Klein, and Ravi [1], using Gilbert's ideas and the result of [17], obtained a polynomial approximation algorithm with ratio $O(\sqrt{d} \log^4 n)$ for MTS on graphs with maximum degree d. They also gave a polynomial approximation algorithm for MTS on general graphs, which generates for an input graph G a chordal supergraph with total number of edges $O((m + \Phi(G))^{3/4} \sqrt{m} \log^{3.5} n)$.

In the *parametric* fill-in problem the input is a graph G and a parameter k. The goal is to find a k-triangulation of G, or to determine that none exists. Clearly this can be done in $n^{O(k)}$ time by enumeration. For fixed k and growing n, an algorithm with complexity $2^{O(k)} n^{O(1)}$ is superior. Parameterized complexity theory, initiated by Downey and Fellows (cf. [6]), studies the complexity of such problems. Parameterized problems that have algorithms of complexity $O(f(k)n^{\alpha})$ (with α a constant) are called *fixed parameter tractable*. Kaplan, Shamir, and Tarjan [16] and later independently Cai [3] proved that the minimum fill-in problem is fixed parameter tractable, by giving an algorithm of complexity $2^{O(k)}m$ for the problem. Both used the same algorithm, with the time bound in [3] being slightly tighter. Kaplan, Shamir, and Tarjan also gave a more efficient $2^{O(k)}+O(k^2nm)$-time algorithm (henceforth, the KST algorithm) for the problem.

In this paper we give the first polynomial approximation algorithm for the minimum fill-in problem. Our algorithm builds on ideas from [16]. For an input graph G with minimum fill-in of size k, our algorithm produces a triangulation of size at most $8k^2$, i.e., within a factor of $8k$ of optimal. The approximation is achieved by identifying in G a *kernel* set of vertices A of size at most $4k$, such that one can triangulate G by adding edges only between vertices of A. Our algorithm produces the triangulation without prior knowledge of k. Let $M(n)$ denote the number of operations needed to multiply two integer matrices of order $n \times n$. (The current upper bound on $M(n)$ is $O(n^{2.376})$ [4].) The algorithm works in time $O(knm + \min\{n^2 M(k)/k, nM(n)\})$, which makes it potentially suitable for practical use.

Our algorithm is particularly attractive for small fill-in values. Note that if $k = \Omega(n)$, then our algorithm guarantees only the trivial bound of fill-in size $O(n^2)$, but if, for example, the fill-in size is constant, then the approximation guarantee is a constant.

This type of approximation result is uncommon. It opens the question of obtaining polynomial approximation algorithms with performance guarantees depending on the optimal value for other important problems, even in the presence of hardness-of-approximation results.

We also obtain better approximation results for bounded degree graphs. For graphs with maximum degree d we give a polynomial algorithm which achieves an approximation ratio of $O(d^{2.5}\log^4(kd))$. Since $k = O(n^2)$, this approximation ratio is polylogarithmic in the input size.

In order to compare our results to the approximation results regarding MTS, we translate the latter to approximation ratios in terms of the fill-in obtained. We assume throughout that $m > n$. For general graphs the algorithm in [1] guarantees that the number of edges in the chordal supergraph obtained is $O((m + k)^{3/4}\sqrt{m}\log^{3.5} n)$. In terms of the fill-in obtained, the approximation ratio achieved is $O(m^{1.25}\log^{3.5} n/k + \sqrt{m}\log^{3.5} n/k^{1/4})$. We obtain a better approximation ratio whenever $k = O(m^{5/8}\log^{1.75} n)$. For graphs with maximum degree d, the algorithm in [1] achieves an approximation ratio of $O(((nd + k)\sqrt{d}\log^4 n)/k)$. We provide a better ratio when $k = O(n/d)$. When any of these upper bounds on k is satisfied, our algorithm also achieves a better approximation ratio than [1] for the MTS problem.

Kaplan, Shamir, and Tarjan posed in [16] an open problem of obtaining an algorithm for the parametric fill-in problem with time $2^{O(k)} + O(km)$. The motivation is to match the performance of the $2^{O(k)}m$ algorithm for all k. We make some progress towards solving that problem by providing a faster $2^{O(k)} + O(knm + \min\{n^2M(k)/k, nM(n)\})$-time implementation of their algorithm. We also give a variant of the algorithm which produces a smaller kernel. Finally, we apply our approximation algorithm to the *chain completion* problem and obtain an approximation ratio of $8k$, where k denotes the size of an optimum solution.

The paper is organized as follows. Section 2 contains a description of the KST algorithm and some background. Section 3 improves the complexity of the KST algorithm and reduces the size of the kernel produced. Section 4 describes our approximation algorithm for general graphs. Section 5 gives an approximation algorithm for graphs with bounded degree. Section 6 gives further reduction of the kernel size, and section 7 gives an approximation algorithm for the chain completion problem.

2. Preliminaries. Let $G = (V, E)$ be a graph. We denote its set V of vertices also by $V(G)$ and its set E of edges also by $E(G)$. For $U \subseteq V$ we denote by G_U the subgraph induced by the vertices in U. For a vertex $v \in V$ we denote by $N(v)$ the set containing all neighbors of v in G. We let $N[v] = N(v) \cup \{v\}$. A path with l edges is called an *l-path* and its *length* is l. A single vertex is considered a 0-path. We call a cycle with l edges an *l-cycle*.

Our polynomial approximation algorithm for the minimum fill-in problem builds on the KST algorithm [16]. In the following we describe this algorithm. Our presentation generalizes that in [16] in order to allow succinct description of the approximation algorithm in section 4.

FACT 2.1. *A minimal triangulation of a chordless l-cycle consists of $l-3$ edges.*

LEMMA 2.2 (see [16, Lemma 2.5]). *Let C be a chordless cycle and let p be an l-path on C, $1 \le l \le |C|-2$. If $l = |C|-2$, then in every minimal triangulation of C there are at least $l-1$ chords incident with vertices of p. If $l < |C|-2$, then in every minimal triangulation of C there are at least l chords incident with vertices of p.*

Let $\langle G = (V, E), k\rangle$ be the input to the parametric fill-in problem. The algorithm has two main stages. In the first stage, which is polynomial in n, m, and k, the

algorithm produces a partition A, B of V and a set F of nonedges in G_A, such that $|A| = O(k^3)$ and no chordless cycle in $G' = (V, E \cup F)$ intersects B. We shall call this stage the *partition algorithm*.

In the second stage, which is exponential in k, an exhaustive search is applied to find a minimum triangulation F' of G'_A. $F \cup F'$ is then proved to be a minimum triangulation of G. The search procedure can be viewed as traversing part of a search tree T, which is defined as follows. Each tree node v corresponds to a supergraph $G(v)$ of G. For the root r, $G(r) = G$. Each leaf of T corresponds to a chordal supergraph of G. At an internal node v, a chordless cycle C in $G(v)$ is identified. For each minimal triangulation F_C of C, a node u is added as a child of v, and its corresponding graph $G(u)$ is obtained by adding F_C to $G(v)$. The algorithm visits only nodes v of T for which $|E(G(v))\backslash E| \leq k$. If such a node is a leaf, then the search terminates successfully. Otherwise, no k-triangulation exists for G.

The partition algorithm applies sequentially the following three procedures. All three maintain a partition A, B of V and a lower bound cc on the minimum number of edges needed to triangulate G. Initially $A=\emptyset, B=V$, and $cc=0$.

(i) *Procedure* $P_1(k)$. *Extracting independent chordless cycles.* Search repeatedly for chordless cycles in G_B and move their vertices from B to A. For each chordless l-cycle found, increment cc by $l-3$. If at any time $cc > k$, stop and declare that the graph admits no k-triangulation.

(ii) *Procedure* $P_2(k)$. *Extracting related chordless cycles with independent paths.* Search repeatedly for chordless cycles in G containing at least two consecutive vertices from B. Let C be such a cycle, $|C| = l$. If $l > k+3$, stop with a negative answer. Otherwise, suppose that C contains $j \geq 1$ disjoint maximal subpaths in G_B, each of length at least 1. Move the vertices of those subpaths from B to A. Denote their lengths in nonincreasing order by $l_1, \ldots, l_j$. If $j = 1$, we increase cc either by l_1-1 if $l_1 = l-2$, or by l_1 if $l_1 < l-2$. Otherwise, cc is increased by $\max\{\frac{1}{2}\sum_{i=1}^{j} l_i, l_1\}$. If at any time $cc > k$, stop and declare that the graph admits no k-triangulation.

DEFINITION 2.3. *For every $x, y \in A$ such that $(x, y) \notin E$, denote by $A_{x,y}$ the set of all vertices $b \in B$ such that x, b, y occur consecutively on some chordless cycle in G. If $|A_{x,y}| > 2k$, then (x, y) is called a* k-essential edge.

(iii) *Procedure* $P_3(k)$. *Adding k-essential edges in G_A.* For every $x, y \in A$ such that $(x, y) \notin E$ compute the set $A_{x,y}$. If (x, y) is k-essential, then add it to G. Otherwise, move all vertices in $A_{x,y}$ from B to A.

Denote by A^i, B^i the partition obtained after procedure P_i is completed for $i = 1, 2, 3$. We shall omit the index i when it is clear from the context. Denote by cc_i the value of cc after procedure P_i is completed for $i = 1, 2$. The size of A^2 is at most $4k$ since $k \geq cc_2 = cc_1 + (cc_2 - cc_1) \geq \frac{1}{4}|A^1| + \frac{1}{2}|A^2 \setminus A^1| \geq \frac{1}{4}|A^2|$. The size of A^3 is $O(k^3)$ since there are $O(k^2)$ nonedges in G_{A^2} and the number of vertices moved to A due to any such nonedge is at most $2k$.

The partition algorithm is summarized in Figure 2.1. Let G' denote the graph obtained after the execution of procedure P_3. Kaplan, Shamir, and Tarjan prove

Execute procedure $P_1(k)$.
Execute procedure $P_2(k)$.
Execute procedure $P_3(k)$.

FIG. 2.1. *The KST partition algorithm.*

that every k-essential edge must appear in any k-triangulation of G [16, Lemma 2.7], and that in G' no chordless cycle intersects B [16, Theorem 2.10]. Therefore, by the following theorem it suffices to search for a minimum triangulation of G'_A.

THEOREM 2.4 (see [16, Theorem 2.13]). *Let A, B be a partition of the vertex set of a graph G, such that the vertices of every chordless cycle in G are contained in A. A set of edges F is a minimal triangulation of G if and only if F is a minimal triangulation of G_A.*

The complexity of the partition algorithm is $O(k^2nm)$ [16]. The complexity of finding a minimum triangulation of a given graph is $O(\frac{4^k}{(k+1)^{3/2}}m)$ [3]. Since G'_A contains $O(k^6)$ edges, a minimum triangulation of G'_A can be found in $O(k^{4.5}4^k)$ time. Hence, the complexity of the KST algorithm is $O(k^2nm + k^{4.5}4^k)$.

3. Improvements to the partition algorithm. In this section we show some improvements to the KST partition algorithm. We assume throughout that the input is $\langle G = (V, E), k\rangle$. We first show how to implement procedure P_3 in $O(nm + \min\{n^2M(k)/k, nM(n)\})$-time. We then prove that the size of A^3 is only $O(k^2)$. These results imply that the KST algorithm can be implemented in $O(knm + \min\{n^2M(k)/k, nM(n)\} + k^{2.5}4^k)$-time.

LEMMA 3.1. *There is an $O(nm+\min\{n^2M(k)/k, nM(n)\})$-time implementation of procedure P_3.*

Proof. Let $S = \{(x, y) \notin E : x, y \in A^2\}$. The bottleneck in the complexity of P_3 is computing the sets $A_{x,y}$ for every $(x, y) \in S$. To this end, we find for every $b \in B$ all pairs $(x, y) \in S$ such that $b \in A_{x,y}$. We then construct the sets $A_{x,y}$. This is done as follows.

Fix $b \in B$. Compute the connected components of $G^b = G \setminus N[b]$. This takes $O(m)$ time. Denote the connected components of G^b by $C^b_1, \ldots, C^b_l$. For each $x \in A^2 \cap N(b)$ compute a binary vector $\vec{v}_x = (v^x_1, \ldots, v^x_l)$ such that $v^x_j = 1$ if and only if C^b_j contains a neighbor of x, $1 \le j \le l$. Each vector can be computed in $O(n)$-time. Let $k' = |A^2 \cap N(b)|$, and number the vertices in $A^2 \cap N(b)$ arbitrarily according to some 1-1 mapping $\sigma : \{1, \ldots, k'\} \to A^2 \cap N(b)$. Define a $k' \times l$ boolean matrix M whose ith row is the vector $\vec{v}_{\sigma(i)}$, $1 \le i \le k'$. Note that $k' = O(k)$ and $l \le n$. Let $M^* = MM^T$. It can be seen that for every pair (i, j) such that $1 \le i < j \le k'$ and $(\sigma(i), \sigma(j)) \in S$, $M^*_{i,j} \ge 1$ if and only if $b \in A_{\sigma(i),\sigma(j)}$. Since $k', l \le n$ we can compute M^* in $O(M(n))$-time. If $k = o(n)$, then we can compute M^* in $O(nM(k)/k)$-time by partitioning M and M^T into $\lceil n/k'\rceil$ submatrices of order at most $k' \times k'$, multiplying corresponding pairs of submatrices, and summing the results. Hence, the computation of M^* takes $O(\min\{nM(k)/k, M(n)\})$ time.

After the above calculations are performed for every $b \in B$, it remains to compute the sets $A_{x,y}$. We can do that in $O(\min\{k^2n, n^3\})$-time. The total time is therefore $O(nm + \min\{n^2M(k)/k, nM(n)\})$. ∎

Observation 3.2. Let $x, y \in A^2, (x, y) \notin E$. If $A_{x,y} \ne \emptyset$, then for any triangulation F of G, either $(x, y) \in F$, or for every $b \in A_{x,y}$, F contains an edge incident on b.

LEMMA 3.3. *Assume that G admits a k-triangulation and that in procedure P_3 all sets $A_{x,y}$ moved into A are of size at most d. Then $|A^3 \setminus A^2| \le Mk$, where $M = \max\{d, 2\}$.*

Proof. Let the nonedges in G_{A^2} be $(x_1, y_1), \ldots, (x_l, y_l)$. We process the sets $A_{x_1,y_1}, \ldots, A_{x_l,y_l}$ in this order. Let $A^{(0)} = A^2$. Let $A^{(i)}$ be the set A right after A_{x_i,y_i} was processed, and let $\Delta_i = A_{x_i,y_i} \setminus A^{(i-1)}$ for $1 \le i \le l$.

Let t be a lower bound on the minimum number of edges needed to triangulate G. Initially P_3 starts with $t = 0$. Let t_i be the value of t right after A_{x_i,y_i} was processed ($t_0 = 0$). If $\Delta_i \neq \emptyset$, then by Observation 3.2, t should increase by $\min\{1, |\Delta_i|/2\}$. We must maintain $t \leq k$. If $t_i - t_{i-1} = 0$, then $|\Delta_i| = 0$. If $t_i - t_{i-1} = 1/2$, then $|\Delta_i| = 1$. If $t_i - t_{i-1} \geq 1$, then $|\Delta_i| \leq d$. Therefore for all $1 \leq i \leq l$, $|\Delta_i| \leq M(t_i - t_{i-1})$. Now,

$$|A^3 \setminus A^2| = |A^{(l)} \setminus A^{(0)}| = \sum_{i=1}^{l} |A^{(i)} \setminus A^{(i-1)}|$$
$$= \sum_{i=1}^{l} |\Delta_i| \leq M \sum_{i=1}^{l} (t_i - t_{i-1}) = M(t - t_0) \leq Mk\,. \qquad \square$$

COROLLARY 3.4. *If G has a k-triangulation, then the partition algorithm terminates with $|A| \leq 2k(k+2)$.*

Proof. Let us assume that all k-essential edges were added to G, and denote the new set of edges of G by E'. For all $x, y \in A^2, (x, y) \notin E'$, we know that $|A_{x,y}| \leq 2k$. By Lemma 3.3, $|A^3 \setminus A^2| \leq 2k^2$. Since $|A^2| \leq 4k$, the corollary follows. $\square$

THEOREM 3.5. *There is an $O(knm + \min\{n^2 M(k)/k, nM(n)\} + k^{2.5}4^k)$-time implementation of the KST algorithm.*

Proof. By the analysis in [16], P_1 takes $O(km)$ time, and P_2 takes $O(knm)$ time. By Lemma 3.1, the complexity of P_3 is $O(nm + \min\{n^2 M(k)/k, nM(n)\})$. By Corollary 3.4, if G admits a k-triangulation, then the size of A^3 is $O(k^2)$. Hence, a minimum triangulation of G'_A can be found in $O(k^{2.5}4^k)$-time [3]. The complexity follows. $\square$

4. The approximation algorithm. Let $G = (V, E)$ be the input graph. Let $k_{opt} = \Phi(G)$. The key idea in our approximation algorithm is to find a set of vertices $A \subseteq V$, such that $|A| = O(k_{opt})$ and, moreover, one can triangulate G by adding edges only between vertices of A. Since there are $O(k_{opt}^2)$ nonedges in G_A, we achieve an approximation ratio of $O(k_{opt})$.

In order to find such a set A we use ideas from the partition algorithm. If we knew k_{opt}, we could execute the partition algorithm and obtain a set A, with $|A| = O(k_{opt}^2)$ (by Corollary 3.4), such that G can be triangulated by adding edges only in G_A. This would already give an $O(k_{opt}^3)$ approximation ratio.

Before describing our algorithm we analyze the role of the parameter k given to the partition algorithm. If $k < k_{opt}$, then the algorithm might stop during P_1 or P_2 and declare that no k-triangulation exists. Moreover, k-essential edges are not necessarily k_{opt}-essential. If $k > k_{opt}$, then the size of A may be $\omega(k_{opt}^2)$. The algorithm is shown in Figure 4.1.

Procedures P'_1 and P'_2 execute P_1 and P_2, respectively, without bounding the size of the triangulation implied. Procedure P'_3 takes advantage of the fact that we no

Algorithm APPROX
  Procedure P'_1: Execute $P_1(\infty)$.
  Procedure P'_2: Execute $P_2(\infty)$.
  Procedure P'_3: Execute $P_3(0)$.
  Let G' be the resulting graph.
  Procedure P'_4: Find a minimal triangulation of G'_A.

FIG. 4.1. *The approximation algorithm.*

longer seek a minimum triangulation but rather a minimal one. In order to obtain our approximation result we want to keep A as small as possible. Hence, instead of moving new vertices to A we add new 0-essential edges accommodating for those vertices. By the same arguments as in [16] and section 2, the size of A after the execution of P_2' is at most $4k_{opt}$. Since P_3' does not add new vertices to A, its size remains at most $4k_{opt}$ throughout. The size of the triangulation found by the algorithm is therefore at most $8k_{opt}^2$. The correctness of Algorithm APPROX is established in what follows. We need the following lemma which is implied by the proof of [16, Lemma 2.9]. The subsequent theorem is a generalization of [16, Theorem 2.10].

LEMMA 4.1. *Let $G = (V, E)$ be a graph and let $v \in V$. Let F be a set of nonedges in $G \setminus \{v\}$, such that each $e = (x, y) \in F$ is a chord in a chordless cycle $C_e = (x, z_e, y, \ldots, x)$ in G, where z_e is not an endpoint of any edge in F. Let $G' = (V, E \cup F)$. If there exists a chordless cycle C in G' with v_1, v, v_2 occurring consecutively on C for some $v_1, v_2 \in N(v)$, then either there exists a chordless cycle in G on which v_1, v, v_2 occur consecutively, or there exists a chordless cycle in G, on which v and z_e occur consecutively for some $e \in F$.*

THEOREM 4.2. *Let $G = (V, E)$ be a graph. Let A, B be a partition of V such that no chordless cycle in G contains two consecutive vertices from B. Let $S = \{(x, y) \notin E : x, y \in A, A_{x,y} \neq \emptyset\}$. Then for any choice of $F \subseteq S$ no chordless cycle in $G' = (V, E \cup F)$ intersects $B' = B \setminus (\bigcup_{(x,y)\in S\setminus F} A_{x,y})$.*

Proof. Suppose to the contrary that C is a chordless cycle in G' intersecting B'. Let $v \in C \cap B'$. Let v_1 and v_2 be the neighbors of v on C. Since $v \in B'$, it is not an endpoint of any edge in F. Every edge $e = (x, y) \in F$ is a chord in a chordless cycle $C_e = (x, z_e, y, \ldots, x)$ of G, where $z_e \in B$. Applying Lemma 4.1, we find that two cases are possible.

1. There exists a chordless cycle in G on which v_1, v, v_2 occur consecutively. If $v_1 \in B$ or $v_2 \in B$, we arrive at a contradiction. Hence, $v_1, v_2 \in A$ and $v \in A_{v_1,v_2}$. We conclude that either $(v_1, v_2) \in F$ or $v \notin B'$, a contradiction.

2. There exists a chordless cycle in G on which v and z_e occur consecutively (for some $e \in F$), a contradiction. ∎

THEOREM 4.3. *Let G be a graph and let $k_{opt} = \Phi(G)$. The algorithm finds a triangulation of G of size at most $8k_{opt}^2$ and can be implemented to run in time $O(k_{opt}nm + \min\{n^2M(k_{opt})/k_{opt}, nM(n)\})$.*

Proof. Correctness. By Theorems 4.2 and 2.4 a minimal triangulation of G'_A is a minimal triangulation of G'. Therefore at the end of the algorithm G is triangulated. Throughout the algorithm the only edges added to G are between vertices of A. Since $|A| \leq 4k_{opt}$, the size of the triangulation is at most $8k_{opt}^2$.

Complexity. The complexity analysis of procedures P_1 and P_2 in [16] implies that P_1' and P_2' can be performed in $O(k_{opt}nm)$-time. By Lemma 3.1 the complexity of P_3' is $O(nm + \min\{n^2M(k_{opt})/k_{opt}, nM(n)\})$. Procedure P_4' requires finding a minimal triangulation of G'_A. Since $|A| = O(\min\{k_{opt}, n\})$ and $|E(G'_A)| = O(\min\{k_{opt}^2, n^2\})$, this requires $O(\min\{k_{opt}^3, n^3\})$ time [19]. Hence, the complexity of the approximation algorithm is $O(k_{opt}nm + \min\{n^2M(k_{opt})/k_{opt}, nM(n)\})$. ∎

Note that, although our analysis uses an upper bound of $\binom{t}{2}$ for the triangulation size of a t-vertex graph, replacing G'_A by the complete graph is not guaranteed to produce a triangulation of G.

5. Bounded degree graphs. In order to improve the approximation ratio for bounded degree graphs, we improve P_4'. Instead of simply finding a minimal triangulation of G'_A, we use the triangulation algorithm of Agrawal, Klein, and Ravi [1].

This alone does not suffice to prove a better approximation ratio, since adding 0-essential edges (in P_3') might not be optimal. In other words, if we denote by F the set of 0-essential edges added to G by P_3', then it might be that $|F| + \Phi(G') > \Phi(G)$. To overcome this difficulty we use the KST partition algorithm with $k = \infty$ as its input parameter, which implies that no new edge will be added to G_A by P_3. The approximation algorithm is as follows:

(i) Execute the KST partition algorithm with parameter $k = \infty$.
(ii) Find a minimal triangulation of G_A using the algorithm in [1].

Assume that the input graph G has maximum degree d, and let $k = \Phi(G)$. We will show that the algorithm achieves an approximation ratio of $O(d^{2.5}\log^4(kd))$. Since $k = O(n^2)$, this is in fact a polylogarithmic approximation ratio. It improves over the $O(k)$ approximation ratio obtained in the previous section, when $k/\log^4 k = \Omega(d^{2.5})$.

THEOREM 5.1. *The algorithm finds a triangulation of G of size within a factor of $O(d^{2.5}\log^4(kd))$ of optimal.*

Proof. Correctness. By the correctness of the KST partition algorithm, we obtain a partition A, B of $V(G)$ for which no chordless cycle in G intersects B. By Theorem 2.4 a minimal triangulation of G_A is a minimal triangulation of G. Therefore, the algorithm correctly computes a minimal triangulation of G.

Approximation ratio. When executing P_3, the size of each set $A_{x,y}$ is at most d. By Lemma 3.3, $|A^3 \setminus A^2| = O(kd)$. Since $|A^2| = O(k)$, the size of A when the partition algorithm terminates is $O(kd)$. Setting the parameter value to ∞ in P_3 guarantees that no new edge is added to G_A, and therefore its maximum degree is at most d and $|E(G_A)| = O(kd^2)$. Using the algorithm in [1] we can produce a chordal supergraph of G_A with $O((kd^2+k)\sqrt{d}\log^4(kd))$ edges. The size of the fill-in obtained is therefore within a factor of $O(d^{2.5}\log^4(kd))$ of optimal. ∎

6. Reducing the kernel size. We now return to the parametric fill-in problem. Let $\langle G = (V, E), k\rangle$ be the input instance. By modifying procedure P_3 in the KST partition algorithm we shall obtain a partition A, B of V and a set of nonedges F, such that no chordless cycle in $G' = (V, E \cup F)$ intersects B and $|A| = O(k)$. In fact we shall obtain at most 2^k such pairs (A, F) and prove that if G has a k-triangulation, then at least for one of those pairs G'_A admits a $(k - |F|)$-triangulation. Reducing the size of A results in improving the complexity of finding a minimum triangulation of G'_A to $O(\sqrt{k}4^k)$, although the total time of the algorithm increases, since we have to handle up to 2^k pairs. We include this result, since it gives further insight on the problem and presents ideas that may help resolve the open problem posed in [16].

As in the original algorithm we start by executing procedures $P_1(k)$ and $P_2(k)$. We also compute the sets $A_{x,y}$ for all $x, y \in A^2, (x, y) \notin E$. If (x, y) is k-essential, we add it to G. Otherwise, we do nothing. Let $\hat{E}$ be the set of k-essential edges, and let $e = |\hat{E}|$. Define $P := \{(x, y) \notin E \cup \hat{E} : x, y \in A^2, A_{x,y} \neq \emptyset\}$, and let $p = |P|$.

The algorithm now enumerates subsets $F \subseteq P$. For a given set F, every $(x, y) \in F$ is added as an edge in the triangulation, and for every $(x, y) \in P \setminus F$, the vertices in $A_{x,y}$ are moved from B to A (which was initialized to A^2). Instead of directly enumerating each set F, we branch and bound. We construct these sets incrementally and stop when a lower bound for the size of the triangulation implied by F exceeds k.

Specifically, the algorithm considers pairs in P one at a time in an arbitrary order $(x_0, y_0), \ldots, (x_{p-1}, y_{p-1})$. For the current pair (x_i, y_i) it distinguishes between three cases as follows. Let $t = |A_{x_i,y_i} \setminus A|$ with respect to the current A. Let cc denote a lower bound for the size of the triangulation implied by the set F constructed so

far (cc is initialized to e). If $t = 0$, then the algorithm does nothing. If $t = 1$, it updates A to $A \cup A_{x_i,y_i}$ and increases cc by $1/2$. Finally, if $t \geq 2$, then the algorithm branches into two cases. In the first case, (x_i, y_i) is added to the triangulation and cc is increased by 1. In the second case, the vertices in A_{x_i,y_i} are moved from B to A, and cc is increased by $t/2$. The algorithm is implemented by the recursive procedure shown in Figure 6.1 and is invoked by calling BRANCH$(e, \emptyset, 0, A^2)$.

```
Procedure BRANCH(cc, F, r, A)
  If cc > k then return.
  If r = p then save the pair (A, F ∪ Ê) and return.
  Let t = |A_{x_r,y_r} \ A|.
  If t = 0 then
    Call BRANCH(cc, F, r + 1, A).
  Else if t = 1 then
    Call BRANCH(cc + 1/2, F, r + 1, A ∪ A_{x_r,y_r}).
  Else /* t ≥ 2 */
    Call BRANCH(cc + 1, F ∪ {(x_r, y_r)}, r + 1, A).
    Call BRANCH(cc + t/2, F, r + 1, A ∪ A_{x_r,y_r}).
  Return.
```

FIG. 6.1. *Algorithm BRANCH.*

LEMMA 6.1. *The algorithm terminates after at most* $p2^{k+1} + 1$ *calls to procedure BRANCH.*

Proof. Denote by $T(i, j)$ the number of recursive calls invoked by BRANCH when called with parameters $cc = i, r = j$ (including this first call). Since always $i \geq 0$ and $0 \leq j \leq p$ in the following, we consider these ranges only. Clearly, $T(i, j) \leq 1 + \max\{T(i, j + 1), T(i + 1/2, j + 1), 2T(i + 1, j + 1)\}$ for all $j < p, i$. Also, $T(i, j) = 1$ for all $i > k, j$, and $T(i, p) = 1$ for all i. It follows that $T(0, 0) \geq T(i, j)$ for all i, j. Hence, it suffices to compute an upper bound for $T(0, 0)$.

We prove that $T(i, j) \leq (p - j)2^{k+1-i} + 1$ by induction on i, j. For $i > k$ or $j = p$ the claim is true. Suppose the claim holds for all i, where $i' \leq i \leq k + 1$, and for all j, where $j' < j \leq p$. Then for $i = i'$ and $j = j'$ we have

$$\begin{aligned} T(i, j) &\leq 1 + \max\{T(i, j + 1), T(i + 1/2, j + 1), 2T(i + 1, j + 1)\} \\ &\leq 2 + \max\{(p - j - 1)2^{k+1-i}, (p - j - 1)2^{k+\frac{1}{2}-i}, (p - j - 1)2^{k+1-i} + 1\} \\ &\leq 3 + (p - j - 1)2^{k+1-i} \leq (p - j)2^{k+1-i} + 1 \ . \end{aligned}$$

It follows that $T(0, 0) \leq p2^{k+1} + 1$. ∎

LEMMA 6.2. *The number of pairs saved by the algorithm is at most* 2^k.

Proof. The proof is analogous to that of Lemma 6.1. Denote by $N(i, j)$ the number of pairs saved by procedure BRANCH, when invoked with parameters $cc = i, r = j$. Since always $i \geq 0$ and $0 \leq j \leq p$, in the following we consider these ranges only. Clearly, $N(i, j) \leq \max\{N(i, j + 1), N(i + 1/2, j + 1), 2N(i + 1, j + 1)\}$ for all $j < p, i$. Also, $N(i, j) = 0$ for all $i > k, j$, $N(k, j) \leq 1$ for all j, and $N(i, p) \leq 1$ for all i. It follows that $N(0, 0) \geq N(i, j)$ for all i, j. Thus, it suffices to compute an upper bound for $N(0, 0)$.

We prove that $N(i,j) \leq 2^{k-i}$ by induction on i,j. If $i \geq k$ or $j = p$, then the claim holds. Suppose the claim holds for all i, where $i' \leq i \leq k$, and for all j, where $j' < j \leq p$. Then for $i = i'$ and $j = j'$ we have

$$\begin{aligned} N(i,j) &\leq \max\{N(i,j+1), N(i+1/2,j+1), 2N(i+1,j+1)\} \\ &\leq \max\{2^{k-i}, 2^{k-\frac{1}{2}-i}, 2^{k-i}\} \\ &= 2^{k-i} . \end{aligned}$$

It follows that $N(0,0) \leq 2^k$. □

As usual, for a set $A \subseteq V$ saved by the algorithm, B denotes $V \setminus A$. The following two claims establish the correctness of our partition algorithm.

LEMMA 6.3. *For every pair (A,F) saved by the algorithm, $|A| \leq 6k$, and no chordless cycle in $G' = (V, E \cup F)$ intersects B.*

Proof. Whenever a partition is saved, $cc \leq k$. By definition of BRANCH, $\frac{1}{2}|A \setminus A^2| \leq cc$. Hence, at most $2k$ new vertices were added to A^2 in any partition obtained. Since $|A^2| \leq 4k$, we conclude that $|A| \leq 6k$. By Theorem 4.2 no chordless cycle in G' intersects B. □

DEFINITION 6.4. *A pair (A,F) saved by BRANCH is called* good *if $\Phi(G) = \Phi(G') + |F|$, where $G' = (V, E \cup F)$.*

PROPOSITION 6.5. *If $\Phi(G) \leq k$, then at least one pair saved by the algorithm is good.*

Proof. Let T be the tree of recursive calls of BRANCH. The nodes of T correspond to invocations of BRANCH. The root of T corresponds to the first invocation of BRANCH. The leaves of T correspond to invocations of BRANCH in which either a pair was saved, or cc was found to exceed k. In nodes at level i of T, $0 \leq i < p$, the pair $(x_i, y_i) \in P$ is processed. Let cc_v, F_v, r_v, and A_v denote the parameters of the invocation of BRANCH which correspond to node v of T.

Let F^* denote a minimum triangulation of G. The proof will identify a root-leaf path in T which corresponds to F^*, and trace the changes to cc, A, and F along that path. We use the following notation:

$$\begin{aligned} P_v &:= \{(x_0,y_0), \ldots, (x_{r_v-1}, y_{r_v-1})\} , \\ F_v^* &:= P_v \cap F^* , \\ A_v^* &:= A^2 \cup \bigcup_{(x,y) \in P_v \setminus F_v^*} A_{x,y} , \\ cc_v^* &:= e + |F_v^*| + \frac{1}{2}|A_v^* \setminus A^2| . \end{aligned}$$

LEMMA 6.6. *For every node v of T, $cc_v^* \leq k$.*

Proof. Let v be any node of T. Let $cc^* = e + |P \cap F^*| + \frac{1}{2}|\bigcup_{(x,y) \in P \setminus F^*} A_{x,y}|$. Since $P_v \subseteq P$, it follows that $cc_v^* \leq cc^*$. By Observation 3.2, for every pair $(x,y) \in P$, either $(x,y) \in F^*$, or for every $b \in A_{x,y}$, F^* contains an edge incident on b. Hence, $cc^* \leq |F^*| \leq k$, where the last inequality follows from the fact that F^* is a k-triangulation. □

We now return to the proof of Proposition 6.5. We shall prove that T has a leaf in which a good pair is saved. To this end, we show that for every $0 \leq i \leq p$, T contains some vertex v at level i for which $F_v \subseteq F_v^*$ and $cc_v \leq cc_v^*$. In particular, this claim implies that T has a leaf z at level p for which $F_z \subseteq F_z^*$ and $cc_z \leq cc_z^*$. By Lemma 6.6, $cc_z \leq cc_z^* \leq k$. Hence, the pair $(A_z, F_z \cup \hat{E})$ is saved at z. By [16, Lemma

2.7], $\hat{E} \subseteq F^*$. In addition, $F_z \subseteq F_z^* \subseteq F^*$. Therefore $(A_z, F_z \cup \hat{E})$ is a good pair, since $F_z \cup \hat{E} \subseteq F^*$ and, by definition, $F^* \setminus (F_z \cup \hat{E})$ triangulates $G' = (V, E \cup F_z \cup \hat{E})$.

We prove the claim by induction on i. The base of the induction is obvious, and as for the root r at level 0, $F_r = \emptyset$ and $cc_r = e$. We assume that the claim is true for level $i-1$ $(i > 0)$ and prove its correctness for level i. By the induction hypothesis T contains a node v at level $i-1 < p$ for which $F_v \subseteq F_v^*$ and $cc_v \leq cc_v^*$. By Lemma 6.6 $cc_v \leq cc_v^* \leq k$, and therefore v is not a leaf. Thus, v has either one or two children in T. There are two cases to examine.

1. Suppose that $(x_i, y_i) \in F^*$. Then for any child w of v, $cc_w^* = cc_v^* + 1 \geq cc_v + 1$. If v has a single child w, then $F_w = F_v \subseteq F_v^* \subset F_w^*$ and $cc_w \leq cc_v + 1/2 < cc_w^*$. Otherwise, let w be the child of v for which $(x_i, y_i) \in F_w$. Then clearly $F_w \subseteq F_w^*$ and $cc_w = cc_v + 1 \leq cc_w^*$.

2. Suppose that $(x_i, y_i) \notin F^*$. Since $F_v \subseteq F_v^*$ and $A_v = A^2 \cup \bigcup_{(x,y) \in P_v \setminus F_v} A_{x,y}$, it follows that $A_v^* \subseteq A_v$. Let w be the child of v for which $(x_i, y_i) \notin F_w$. Then $F_w = F_v \subseteq F_v^* = F_w^*$ and

$$cc_w = cc_v + \frac{1}{2}|A_{x_i,y_i} \setminus A_v| \leq cc_v^* + \frac{1}{2}|A_{x_i,y_i} \setminus A_v^*| = cc_w^* . \qquad \square$$

THEOREM 6.7. *If $\Phi(G) \leq k$, then the new partition algorithm produces at least one pair (A, F) for which $|A| \leq 6k$ and $\Phi(G) = \Phi(G'_A) + |F|$, where $G' = (V, E \cup F)$. The complexity of the algorithm is $O(knm + \min\{n^2 M(k)/k, nM(n)\} + k^3 2^k)$.*

Proof. Correctness. By Lemma 6.3 for each pair (A, F) saved by the algorithm, $|A| \leq 6k$ and no chordless cycle in G' intersects B. Therefore, by Theorem 2.4 for each such pair $\Phi(G') = \Phi(G'_A)$. Since $\Phi(G) \leq k$, by Proposition 6.5 the algorithm saves some pair (A, F) for which $\Phi(G) = \Phi(G') + |F|$. Correctness follows.

Complexity. By [16] P_1 and P_2 take $O(knm)$ time. By Lemma 3.1, computing the sets $A_{x,y}$ for all $x, y \in A^2, (x, y) \notin E$ takes $O(nm + \min\{n^2 M(k)/k, nM(n)\})$ time. By Lemma 6.1 and the fact that $|P| = O(k^2)$, the number of calls to BRANCH is $O(k^2 2^k)$. By Lemma 6.3 and since $\Phi(G) \leq k$, the parameters A and F to each invocation of BRANCH satisfy $|A| = O(k)$ and $|F| \leq k$. Also, for all $(x, y) \in P$, $|A_{x,y}| \leq 2k$. Thus, each call can be carried out in $O(k)$ time. The total work done by BRANCH is therefore $O(k^3 2^k)$. $\square$

7. An approximation algorithm for the chain completion problem. A bipartite graph $G = (P, Q, E)$ is called a *chain graph* if there exists an ordering π of P, $\pi : P \to \{1, \ldots, |P|\}$, such that $N(\pi^{-1}(1)) \subseteq N(\pi^{-1}(2)) \subseteq \cdots \subseteq N(\pi^{-1}(|P|))$. This class of graphs was introduced by Yannakakis [23], and independently by Golumbic (cf. [15, page 260]). The *chain completion* problem is defined as follows: Given a bipartite graph $G = (P, Q, E)$, find a minimum set of nonedges F such that $(P, Q, E \cup F)$ is a chain graph. We call $|F|$ the *chain fill-in*. Yannakakis proved that the chain completion problem is NP-complete and used this result to show that the minimum fill-in problem is NP-complete [23]. Chain graphs have been also investigated in [8], where a similar graph modification problem arises.

THEOREM 7.1. *There exists a polynomial approximation algorithm for the chain completion problem, achieving an approximation ratio of $8k$, where k denotes the minimum chain fill-in. The complexity of the algorithm is $O(kn^3)$.*

Proof. Let $G = (U, V, E)$ be an input bipartite graph with chain fill-in k. We apply the following reduction given by Yannakakis [23] from the chain completion problem to the minimum fill-in problem. Build a graph $G' = (U \cup V, E')$, where $E' = E \cup \{(u, v) : u, v \in U\} \cup \{(u, v) : u, v \in V\}$. Observe that G is a chain graph

if and only if G' is chordal. Hence, a set of edges F triangulates G' if and only if $(U, V, E \cup F)$ is a chain graph.

Approximation ratio. By the above argument, k equals $\Phi(G')$. Using our approximation algorithm for the minimum fill-in problem, we can find a triangulation of G' of size at most $8k^2$. Adding these edges to G produces a chain graph. The number of new edges is within a factor of $8k$ of optimal.

Complexity. G' can be computed in $O(n^2)$ time. Due to the reduction, $|E(G')| = \Theta(n^2)$. Therefore the complexity of the approximation algorithm is $O(kn^3)$. □

Acknowledgments. We thank Itsik Pe'er for many useful remarks. We also thank an anonymous referee for many helpful suggestions.

REFERENCES

[1] A. Agrawal, P. Klein, and R. Ravi, *Cutting down on fill using nested dissection: Provably good elimination orderings*, in Graph Theory and Sparse Matrix Computation, A. George, J. R. Gilbert, and J. W. H. Liu, eds., Springer-Verlag, New York, 1993, pp. 31–55.

[2] J. R. Bunch and D. J. Rose, eds., *Sparse Matrix Computations*, Academic Press, New York, 1976.

[3] L. Cai, *Fixed-parameter tractability of graph modification problems for hereditary properties*, Inform. Proc. Lett., 58 (1996), pp. 171–176.

[4] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions*, J. Symbolic Comput., 9 (1990), pp. 251–280.

[5] E. Cuthill and J. McKee, *Reducing the bandwidth of sparse symmetric matrices*, in Proc. 24th Nat. Conf. ACM, 1969, pp. 157–172.

[6] R. Downey and M. Fellows, *Parameterized Complexity*, Springer-Verlag, New York, 1999.

[7] I. Duff, ed., *Sparse Matrices and Their Uses*, Academic Press, New York, 1981.

[8] D. P. Fasulo, T. Jiang, R. M. Karp, and N. Sharma, *Constructing maps using the span and inclusion relations*, in Proceedings of the 2nd International Conference on Computational Molecular Biology, ACM, New York, 1998, pp. 64–73.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, CA, 1979.

[10] A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[11] A. George, J. R. Gilbert, and J. W. H. Liu, eds., *Graph Theory and Sparse Matrix Computation*, Springer-Verlag, New York, 1993.

[12] A. George and J. Liu, *The evolution of the minimum degree ordering algorithm*, SIAM Rev., 31 (1989), pp. 1–19.

[13] A. George and J. W. Liu, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice–Hall, Englewood Cliffs, NJ, 1981.

[14] J. R. Gilbert, *Some nested dissection order is near optimal*, Inform. Process Lett., 26 (1988), pp. 325–328.

[15] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[16] H. Kaplan, R. Shamir, and R. E. Tarjan, *Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs*, SIAM J. Comput., 28 (1999), pp. 1906–1922.

[17] F. T. Leighton and S. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with application to approximation algorithms*, in Proceedings of the 29th Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1988, pp. 101–111.

[18] A. Natanzon, R. Shamir, and R. Sharan, *A polynomial approximation algorithm for the minimum fill-in problem*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 41–47.

[19] T. Ohtsuki, *A fast algorithm for finding an optimal ordering for vertex elimination on a graph*, SIAM J. Comput., 5 (1976), pp. 133–145.

[20] T. Ohtsuki, L. K. Cheung, and T. Fujisawa, *Minimal triangulation of a graph and optimal pivoting order in a sparse matrix*, J. Math. Anal. Appl., 54 (1976), pp. 622–633.

[21] J. D. Rose, *A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations*, in Graph Theory and Computing, R. C. Reed, ed., Academic Press, New York, 1972, pp. 183–217.

[22] U. Schendel, *Sparse Matrices: Numerical Aspects with Applications for Scientists and Engineers*, Ellis Horwood, Chichester, UK, 1989.

[23] M. Yannakakis, *Computing the minimum fill-in is NP-complete*, SIAM J. Alg. Discrete Methods, 2 (1981), pp. 77–79.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1080–1125

PARALLEL COMPLEXITY OF COMPUTATIONS WITH GENERAL AND TOEPLITZ-LIKE MATRICES FILLED WITH INTEGERS AND EXTENSIONS*

VICTOR Y. PAN†

Abstract. Computations with Toeplitz and Toeplitz-like matrices are fundamental for many areas of algebraic and numerical computing. The list of computational problems reducible to Toeplitz and Toeplitz-like computations includes, in particular, the evaluation of the greatest common divisor (gcd), the least common multiple (lcm), and the resultant of two polynomials, computing Padé approximation and the Berlekamp–Massey recurrence coefficients, as well as numerous problems reducible to these. Transition to Toeplitz and Toeplitz-like computations is currently the basis for the design of the parallel randomized NC (RNC) algorithms for these computational problems.

Our main result is in constructing nearly optimal randomized parallel algorithms for Toeplitz and Toeplitz-like computations and, consequently, for numerous related computational problems (including the computational problems listed above), where all the input values are integers and all the output values are computed exactly. This includes randomized parallel algorithms for computing the rank, the determinant, and a basis for the null-space of an $n \times n$ Toeplitz or Toeplitz-like matrix A filled with integers, as well as a solution $\mathbf{x}$ to a linear system $A\mathbf{x} = \mathbf{f}$ if the system is consistent. Our algorithms use $O((\log n)\log(n \log \|A\|))$ parallel time and $O(n \log n)$ processors, each capable of performing (in unit time) an arithmetic operation, a comparision, or a rounding of a rational number to a closest integer. The cost bounds cover the cost of the verification of the correctness of the output. The computations by these algorithms can be performed with the precision of $O(n \log \|A\|)$ bits, which matches the precision required in order to represent the output, except for the rank computation, where the precision of the computation decreases. The algorithms involve either a single random parameter or at most $2n - 1$ parameters.

The cited processor bounds are less by roughly factor n than ones supported by the known algorithms that run in polylogarithmic arithmetic time and do not use rounding to the closest integers.

Technically, we first devise new algorithms supporting our old nearly optimal complexity estimates for parallel computations with general matrices filled with integers. Then we decrease dramatically, by roughly factor $n^{1.376}$, the processor bounds required in these algorithms in the case where the input matrix is Toeplitz-like. Our algorithms exploit and combine some new techniques (which may be of independent interest, e.g., in the study of parallel and sequential computation of recursive factorization of integer matrices) as well as our earlier techniques of variable diagonal (relating to each other several known algebraic and numerical methods), stream contraction, and the truncation of displacement generators in Toeplitz-like computations; our development and application of these techniques may be of independent interest.

Key words. parallel algorithms, randomized algorithms, Toeplitz matrix computations, Toeplitz-like matrices, polynomial gcd, displacement rank, computational complexity, block Gauss–Jordan decomposition, p-adic lifting, Newton–Hensel's lifting

AMS subject classifications. 68Q22, 68Q25, 68Q40, 65Y20, 47B35, 65F30

PII. S0097539797349959

1. Introduction.

1.1. Toeplitz and Toeplitz-like matrices and some applications.

The fast version of Euclidean algorithm [AHU74], [BGY80] computes the greatest common di-

*Received by the editors January 18, 1999; accepted for publication (in revised form) March 20, 2000; published electronically August 29, 2000. The results of this paper have been presented at the ACM–SIAM Workshop on Mathematics of Numerical Analysis: Real Numbers Algorithms, Park City, Utah, 1995.

http://www.siam.org/journals/sicomp/30-4/34995.html

†Department of Mathematics and Computer Science, Lehman College, City University of New York, Bronx, NY 10468 (vpan@alpha.lehman.cuny.edu). The work of this author was supported by NSF grants CCR 9020690 and CCR 9625344 and PSC CUNY Award 666327.

visor (gcd) of two polynomials of degrees at most n by using $O(n(\log n)^2)$ arithmetic or field operations (that is, additions, subtractions, multiplications, and divisions), but to yield substantial parallel acceleration, one has to reduce the problem to the solution of the associated (possibly singular) Toeplitz-like (resultant or subresultant) or Toeplitz linear system of $O(n)$ equations, $T\mathbf{x} = \mathbf{f}$ [BGH82], [G84], [BP94]. (The concepts of Toeplitz and Toeplitz-like matrices are well known (see [KKM79], [CKL-A87], [BP94], pp. 47–48, 138–141, 148–151), but for the reader's convenience, we recall their definitions below. (Also see Definitions 2.18 and 13.2 in sections 2 and 13.))

The gcd computation is only one (though celebrated) example of various major problems of algebraic and numerical computing whose solution is reduced to solving Toeplitz or Toeplitz-like linear systems of equations. The list of such problems includes the computation of the resultant, the Sturm and subresultant sequences, and the least common multiple (lcm) for a pair of univariate polynomials ([BT71], [BGY80], [BP94], sections 2.8–2.10), as well as the shift register synthesis and linear recurrence computation [Be68], [Ma75], inverse scattering [BK87], adaptive filtering [K74], [H91], modelling of stationary and nonstationary processes [KAGKA89], [KVM78], [K87], [L-AK84], [L-AKC84], numerical computations for Markov chains [Ste94], Padé approximation of an analytic function [BGY80], polynomial interpolation and multipoint evaluation [PSLT93], [PZHY97], solution of partial differential and integral equations [Bun85], [C47/48], [KLM78], [KVM78], parallel computations with general matrices over an arbitrary field of constants [P91], [P92], [KP91], [KP92], approximating polynomial zeros [P95], [P96a], [P97], and the solution of polynomial systems of equations [EP97], [MP98], [BMP98].

Furthermore, the general reduction techniques of [P90] enable us to extend the algorithms available for Toeplitz and Toeplitz-like computations to computations with some other major classes of structured matrices, such as Cauchy-like and Vandermonde-like matrices, also highly important in many areas of computing [PSLT93], [H95], [GKO95], [PZHY97], [OP98], [OP99], [P00], [P00a].

The design of new effective algorithms for parallel solution of Toeplitz and Toeplitz-like linear systems will be our major goal. For the reader's convenience, we will next briefly recall the definitions and some basic properties of Toeplitz and Toeplitz-like matrices. (See section 13 and Definition 2.18 of section 2 for more details.)

$T = (t_{i,j})$ is an $n \times n$ *Toeplitz matrix* if

$$t_{i,j} = t_{i+1,j+1} \quad \text{for} \quad i, j = 0, 1, \dots, n-2, \tag{1.1}$$

that is, if the entries of T are invariant in their shifts into the diagonal direction. Toeplitz matrices are easy to store, since such an $n \times n$ matrix is fully represented by the $2n-1$ entries of its first column (or row, respectively) and its last column (or row). Multiplication of an $n \times n$ Toeplitz matrix by a vector can be reduced to three fast Fourier transforms (FFTs) (e.g., via its reduction to polynomial multiplication modulo x^{2n-1} (see [BP94], p. 133)) and can be performed by using $O(n \log n)$ arithmetic operations. Hereafter, arithmetic operations, as well as comparisons of pairs of rational numbers and the rounding of a rational number to a closest integer, are referred to as *ops*.

Due to the structural properties of Toeplitz matrices, one may solve a nonsingular Toeplitz linear system of n equations by using $O(n(\log n)^2)$ ops [BGY80], [Morf80], [BA80], [Mu81], [dH87], [AGr88], [K95]. (Note that we would need storage space $n^2 + n$ and $2n^2 - n$ ops to multiply a general matrix by a vector and order of n^d ops with $d > 2$ to solve a general nonsingular linear system of n equations.)

The above properties are extended to the class of $n \times n$ *Toeplitz-like matrices*, that is, ones represented in the form

$$T = \sum_{i=1}^{\ell} L_i U_i, \tag{1.2}$$

where L_i and U_i^T are $n \times n$ lower triangular Toeplitz matrices, U_i^T is the transpose of U_i, and ℓ is bounded by a fixed constant, $\ell = O(1)$. (Note that any $n \times n$ matrix can be represented in the form (1.2) for $\ell \le n$.) It suffices to store the $2\ell n$ entries of the first columns of L_i and U_i^T for $i = 1, \dots, \ell$ in order to represent T. These 2ℓ columns form a pair of $n \times \ell$ matrices called a *displacement generator* of T of length ℓ.

Representation (1.2) for a Toeplitz-like matrix enables us to manipulate with $O(n)$ entries of its displacement generator (rather than with its n^2 entries). Furthermore, we may immediately multiply a matrix T of (1.2) by a vector by using $O(\ell n \log n)$ ops, and also we may solve a linear system $T\mathbf{x} = \mathbf{f}$ in $O(\ell^2 n (\log n)^2)$ ops if T is nonsingular [Morf80], [BA80], [Mu81].

We have $\ell \le 2$ in (1.2) for Toeplitz matrices, their inverses, and resultant matrices, and $\ell \le g + h$ for $g \times h$ block matrices with Toeplitz blocks. Furthermore, the transposition of a matrix leaves ℓ invariant, whereas ℓ may grow only slowly in multiplication and addition/subtraction of pairs of matrices and stays unchanged or grows only nominally in the inversion of a nonsingular matrix (see section 13).

1.2. NC and RNC solutions (some background). Due to their reduction to Toeplitz/Toeplitz-like linear systems, several computational problems listed in the previous section, including the gcd, lcm, and resultant computation and Padé approximation, can be solved by using $O(n(\log n)^{d_1})$ ops, where n is the input size, and $d_1 \le 3$. (We may need to allow $d_1 = 3$ in order to handle singular Toeplitz/Toeplitz-like linear systems; we reduce their solution to computing the rank of the coefficient matrix and to the subsequent solution of a nonsingular Toeplitz-like linear system.) Like the Euclidean algorithm, however, such solution algorithms require an order of n parallel steps.

The known alternative algorithms yield NC or randomized NC (RNC) solutions of all the cited computational problems [BGH82], [G84], [BP94], that is, yield their solution by using $t(n) = O((\log n)^c)$ time and $p(n) = O(n^d)$ arithmetic processors, for two fixed constants c and d, under the customary exclusive read exclusive write random access machine (EREW PRAM) arithmetic model of parallel computing [KR90], [J92]. (Alternatively, we may define the NC and RNC solutions as the families of arithmetic, Boolean, or arithmetic-Boolean circuits for the above problems having depths $O((\log n)^c)$ and sizes $O(n^d)$ for two fixed constants c and d [G86].) Indeed, the NC/RNC solution of a linear system of n equations can be computed over any field of constants [Cs76], [Be84], [Ch85], [KP91], [KP92]. These algorithms, however, leave open the important problem of processor efficiency of (R)NC Toeplitz and Toeplitz-like computations, that is, of having the ratios $p(n)/T_+(n)$ or even $p(n)/T_-(n)$ at the level $O((\log n)^{c_1})$ for a constant c_1, where $T_+(n)$ and $T_-(n)$ denote the record upper and lower bounds on the sequential time of the solution, respectively. Indeed, on the one hand, we have already cited the bound $T_+(n) = O(n(\log n)^3)$, and, clearly, $T_-(n) \ge n$. On the other hand, $p(n)$ has order n^d for $d > 3$ in [Cs76], [Be84], [Ch85] and for $d > 2$ in [KP91], [KP92], for solving general linear systems in NC/RNC, whereas $p(n)$ has order n^d for $d \ge 2$ for the known NC/RNC Toeplitz/Toeplitz-like solvers over any field of constants [P92], [KP94], [P96], [P96b]. To yield NC/RNC

and processor efficiency, we must decrease d to the optimal level 1. An approach toward this goal was outlined in [BP94, p. 357], incorporating various nontrivial techniques developed earlier for computations with general and dense structured matrices [Morf80], [BA80], [P85], [P87], [P92], [P92b], [P93], [P93a], and our objective in the present paper is to show in detail how this can be done, under certain assumptions on the model of computing.

1.3. The model of computing. Our main assumption is that the input consists of integers (for the reduction from a real input, one may use binary or decimal chopping followed by scaling) and that rounding a rational number to a closest integer, as well as an arithmetic operation or comparison of two rationals, are allowed as unit cost operations. The bit-precision of these computations will be bounded at the optimal level of the output precision, so that we achieve solution at a low Boolean cost.

Stating our estimates for the computational cost, we will let $O_A(t,p)$ and $O_B(t,p)$ denote the simultaneous bounds $O(t)$ on the parallel time and $O(p)$ on the number of arithmetic or Boolean processors, respectively. We will routinely decrease the processor bounds slightly, by exploiting the B-*principle* of parallel computing, which is a variant of Brent's principle and according to which $O(s)$ time-steps of a single arithmetic or Boolean processor may simulate a single time-step of s arithmetic or, respectively, Boolean processors [KR90], [PP95]. According to the B-principle, the bound $O_A(t,kp)$ implies the bound $O_A(tk,p)$, and similarly $O_B(t,kp)$ implies the bound $O_B(tk,p)$ for a parameter $k \geq 1$. (For $p=1$, we arrive at sequential time bounds $O_A(tk,1)$ and $O_B(tk,1)$.) By applying the well-known technique based on the B-principle, one may slow down the computations at the stages requiring too many processors. In many cases this increases the time bound only by a constant factor but more substantially decreases the processor bound; a celebrated example is the summation of n values, where application of the B-principle decreases the asymptotic cost bound from $O_A(\log n, n)$ to $O_A(\log n, n/\log n)$ (cf. [Q94, pp. 44–46]; [BP94, pp. 297–298]). (The converse trade-off of time and processor bounds is not generally possible, but for almost all matrix computations that we consider and, more generally, for any task of the evaluation of a set of multivariate polynomials, one may always transform an NC/RNC algorithm into one using $O(\log^2 n)$ time and $O(n^d)$ arithmetic processors for some finite but generally quite large constant d [VSBR83], [MRK88]. We will not use the latter result as we are concerned about processor bounds.)

REMARK 1.1. *Our algorithms for Toeplitz/Toeplitz-like computations are essentially reduced to computing the convolutions (which can be performed via FFT, assuming that the* 2^h*th roots of* 1 *are available) and the inner products of pairs of vectors. These basic operations (for vectors of a dimension* n*) can be performed at the cost* $O_A(\log n, n)$ *and* $O_A(\log n, n/\log n)$*, respectively, under both the EREW PRAM model and more realistic models such as hypercube, butterfly and shuffle-exchange processor arrays* [Le92], [Q94]. *Thus, it is possible to implement our algorithms efficiently assuming the latter models.*

1.4. Our main results. The algorithms of this paper extend our previous work on parallel computations with general matrices [P85], [P87], [P93a], [BP94] (cf. also [PR91], [P92a], [P93b], [PR93]) by means of incorporation of some techniques developed in [P90], [P92], [P92b], [P93], [P93a] for computations with Toeplitz and Toeplitz-like matrices. As a result, we arrive at RNC algorithms for the most fundamental computations with the latter classes of matrices filled with integers (such as the computation of their ranks, null-spaces and determinants and solving linear sys-

tems of equations). These algorithms yield optimal (up to polylogarithmic factors) time and processor bounds, which improves by factor n the processor bounds of the known RNC algorithms. By using the known reduction to Toeplitz and Toeplitz-like computations, we also extend our results to yield similar nearly optimal upper bounds on the time and processor complexity (also achieving order of n improvement versus the known RNC algorithms) for many other related computations (e.g., the computation of polynomial gcd and lcm and Padé approximation), where the input values are integers.

We will emulate the historic line, by first treating the case of general matrices and then improving the algorithms in the Toeplitz/Toeplitz-like case. We will start with recalling the record parallel complexity bounds of [P85], [P87] for computations with a general $n \times n$ input matrix; we will give their alternative derivation. Stating these bounds in Theorem 1.1 below, we will use the value ω satisfying $2 \le \omega < 2.376$ and such that a pair of $n \times n$ matrices can be multiplied at the arithmetic cost $O_A(\log n, n^\omega)$. We note that the magnitudes of det A and the integer entries of adj $A = A^{-1}$ det A can be as large as $||A||^n$ or $||A||^{n-1}$, which means the output precision of an order of $n \log ||A||$. We ensure that the precision of the computations by our algorithms does not exceed this level. Furthermore, we compute the rank of A by using even a lower precision, which enables some decrease of the Boolean cost of the computation of the rank.

Technically, we will largely follow the cited outline, given by us in [BP94, chapter 4], and combine a variety of the known techniques, in particular ones developed in [P85], [P87], [P92b], [P93], [P93a], and some new ones (such as the combination of primal and dual recursive decompositions of an integer matrix with the objective to bound the magnitude of the intermediate and output values). The required combination of all these techniques is highly nontrivial and never was presented in either complete or accessible form.

The main purpose of our paper is to give such a presentation or, formally speaking, to give complete and accessible proofs of the two theorems below. The first of them only handles the case of general integer input matrices (in this case our parallel complexity results repeat ones of [P85], [P87], except for the presently improved Boolean cost of the rank computation), but we use distinct alternative proof, which should be technically interesting in its own right and is fully used in our subsequent relatively simpler extension to the Toeplitz/Toeplitz-like case, handled by our second theorem.

THEOREM 1.1. *Let A be a $k \times h$ matrix and let $\mathbf{f}$ be an h-dimensional vector, both matrix and vector filled with integers that range from -2^a to 2^a for some $a > 1$. Let $k + h = O(n)$. Then, with an error probability of at most n^{-c} for a fixed positive constant c, one may compute r, the rank of A, at a randomized computational cost bounded by $O_A((\log n) \log(n \log a), n^\omega)$ and by*

$$O_B((\log n)(\log(n \log a))^2 \log\log(n \log a), n^{\omega+1} \log(na)).$$

Furthermore, one may compute the determinant of A and, if A is nonsingular, then also the inverse of A and the solution to a linear system $A\mathbf{x} = \mathbf{f}$, all of them at a randomized computational cost bounded by $O_A((\log n) \log(na), n^\omega)$ and by

$$O_B((\log n)(\log(na))^2 \log\log(na), (\log n)(a + \log n) n^{\omega+1} / \log(na)).$$

If A is an $n \times n$ singular matrix, the latter bounds also apply to the computation of $n - r$ basis vectors of a null-space of A and a solution $\mathbf{x}$ to a linear system $A\mathbf{x} = \mathbf{f}$ provided

that this system is consistent. The same cost bounds apply to testing correctness of the computed value of r=rank A *as well as of all other output values. In these computations,* $2n-1$ *random parameters are used for computing* rank A, $\det A$, *and the null-space of* A, *and a single random parameter is used for all other tasks including the computation of* $|\det A|$. *The above complexity estimates do not cover the cost of generation of the random parameters.*

In the case of a $k \times h$ Toeplitz or Toeplitz-like input matrix (defined in section 1.1 and also in sections 2 (Definition 2.18) and 13 (Definition 13.2)), an extension of our approach yields much smaller (by factor $n^{\omega-1}/\log n$) upper bounds on the processor complexity of the same computations (with no increase of the asymptotic time-bounds).

THEOREM 1.2. *Under the assumptions of Theorem* 1.1, *let the input matrix* A *be a Toeplitz matrix or a Toeplitz-like matrix. Then all the processor complexity estimates of Theorem* 1.1 *can be decreased by factor* $n^{\omega-1}/\log n$, *preserving the time bounds, to yield the randomized parallel complexity bounds* $O_A((\log n)\log(n\log a), n\log n)$, $O_B\ ((\log n)(\log(n\log a))^2\log\log(n\log a), (n^2\log n)\log(na))$, *and* $O_A((\log n)\log(na), n\log n)$, $O_B((\log n)(\log(na))^2\log\log(na), (a+\log n)(n\log n)^2/\log(na))$, *respectively. Here, the inverse of* A *and the basis matrix for the null-space of* A *are assumed to be output in the form of their displacement generators.*

We refer the reader to Remark 12.2 on possible minor refinement of the estimates of both theorems.

Due to substantial economization of computational resources in our algorithms for Toeplitz/Toeplitz-like computations, they may become practically efficient provided that they are supported by subroutines for multiprecision parallel computations with integers and polynomials and by the development of the interface between algebraic and numerical computing, both required in our algorithms. Such a development is motivated by various potential benefits, our algorithms is but one of many examples. The practical implementation of our algorithms for general $n \times n$ matrices faces a harder problem of the storage of n^2 long integers in the computer memory (versus $2n-1$ in the Toeplitz case), and this task becomes practically infeasible at some point as n increases.

Our algorithms do not improve the known sequential algorithms for Toeplitz and Toeplitz-like computations [BGY80], [Morf80], [BA80], [Mu81], [dH87], which run in nearly optimal arithmetic time of $O(n\log^2 n)$, but some of our techniques may be of practical and theoretical interest for sequential computations too. In particular, our Toeplitz–Newton iteration techniques are effective for rapid practical improvement of approximate solution of Toeplitz and Toeplitz-like linear systems of equations [PBRZ99], and our study of integral version of recursive decomposition as well as our bounds on the growth of the auxiliary integers (particularly, of the auxiliary determinants) is a natural but nontrivial extension of the Bareiss version of Gaussian elimination (cf. [B68]). Even our simple idea of the precision decrease in the randomized computation of the rank (by performing the computation modulo a fixed prime) leads to a substantial decrease of the sequential Boolean time bounds (for both general and Toeplitz/Toeplitz-like matrices). The latter trick also applies to the closely related problem of the computation of the degree of the gcd and lcm of two polynomials with integer coefficients.

1.5. Extensions. We already cited [BGY80], [G84], [P92], [P96b], and [BP94] on the reduction of the computation of the gcd, the lcm, and the resultant of two polynomials as well as Padé approximation of a formal power series or of a polynomial—to

the computation of the rank of a Toeplitz/Toeplitz-like matrix and solving a nonsingular Toeplitz/Toeplitz-like linear system of equations. The integrality of the input can be preserved in this reduction, and the input size may grow by a factor of at most 2. Therefore, *the computational complexity estimates of Theorem* 1.2 *are immediately extended to the listed problems* of the gcd, lcm, resultant and Padé computations (assuming the restrictions on the size and integrality of the input), as well as to various computational problems reducible to the latter ones.

Furthermore, we refer the reader to [P90], on the general techniques that immediately enable extension of our results of Theorem 1.2 to computations with Cauchy-like and Vandermonde-like input matrices, to [BP93], [BP94] on the extension to the case of matrices represented as the sums of Hankel-like and Toeplitz-like matrices, and to [BGY80], [BP94], [P96b], [PSLT93], [PZHY97], and other references cited in the beginning of this paper on various applications of the computations with Toeplitz-like and other structured matrices (see also Remark 14.1).

Among possible extensions of Theorem 1.1, consider the case where the integer matrix A is symmetric positive definite, sparse, and associated with an $s(n)$-separatable graph given with its $s(n)$-separator family (cf. [LRT79], [P93b], [PR93]). If such a matrix A is well conditioned (even if its entries are not integers but any real numbers), then, at the arithmetic cost $O_A((\log n)^3, (s(n))^\omega/\log n)$, the parallel algorithm of [P93b], [PR93] numerically computes both recursive factorization of such a matrix and its determinant, as well as a solution $\mathbf{x} = A^{-1}\mathbf{f}$ to a linear system $A\mathbf{x} = \mathbf{f}$ (if $\det A \neq 0$). Numerical approximation is involved in this algorithm at the auxiliary stages of matrix inversions, where a parallel algorithm of [PR89] is applied. If A is filled with integers, then this stage can be performed exactly, by using the algorithms of [P85], [P87]. Then, the exact recursive factorization of A, $\det A$, and $A^{-1}\mathbf{f}$ can be computed at the arithmetic cost $O_A((\log n)^3, (s(n))^\omega + n)$. By employing the algorithm of this paper for recursive decomposition and inversion of a general integer matrix, one may improve the latter bounds a little, to yield $O_A((\log n)^2, (s(n))^\omega + n)$.

The results of Theorems 1.1 and 1.2 can be further extended to various other matrix computations by using the known reduction techniques of [BP94], [P96], [P96b]. For demonstration, consider the computation of the characteristic polynomial $c_A(x) = \det(xI - A)$ of the above sparse $n \times n$ matrix A. Such a polynomial has degree n. We may first concurrently compute $c_A(x)$ at $n+1$ distinct points $x_0, \dots, x_n$ and then obtain its coefficients by interpolation. If the chosen values of x_i are larger than $n\|A\|$, then the matrices $x_iI - A$ are positive definite, and we may compute $c_A(x_i)$ for $i = 0, 1, \dots, n$, at the overall computational cost $O_A((\log n)^2, ((s(n))^\omega + n)n)$. These bounds dominate the cost of the subsequent interpolation producing the polynomial $\det(xI - A)$.

As another example, the algorithms of [BP94, p. 357], for Padé approximation and polynomial gcd have been used in [P95] and [P96a] in order to obtain the record parallel arithmetic complexity estimates for approximating polynomial zeros. Our present improvement of these results of [BP94] in Theorem 1.2 immediatley implies the respective minor improvement of the results of [P95] and [P96a].

COROLLARY 1.3. *Given a positive b and the coefficients of an nth degree monic polynomial with zeros $z_1, \dots, z_n$ satisfying $\max_i |z_i| \leq 1$, one may compute approximations $z_1^*, \dots, z_n^*$ to $z_1, \dots, z_n$ satisfying $|z_i^* - z_i| < 2^{-b}$, $i = 1, \dots, n$; the computation is randomized; its arithmetic cost is bounded by $O_A((\log n)^3((\log n)^2 + \log(b+2)), \frac{n}{\log n})$.*

1.6. Outline of the method. A major ingredient of our approach is the *variable diagonal method* of [P85], [P87], which combines several algebraic and numerical

techniques to yield effective parallel inversion of a matrix A filled with integers. The method includes *Newton's iteration*, which effectively solves the latter problem provided that a good *initial approximation* to A^{-1} is available. Such an approximation is not available, however, for a general integer matrix A. The recipe of [P85], [P87] is to invert at first the auxiliary matrix $F = V - apI$, where $V = A \bmod p$, I is the identity matrix of an appropriate size, p is a prime, $p \geq n$, and a is a sufficiently large integer. (We follow this recipe and show that it suffices to choose $a = 10pn^2$ in our case.) Then the matrix $-I/(ap)$ is a good initial approximation to F^{-1}, which we rapidly improve by Newton's iteration, until F^{-1} is approximated closely enough. Since F is an integer matrix, $\det F$ and the entries of adj $F = (\det F)F^{-1}$ are integers, which can be recovered by rounding their approximations within absolute errors less than $1/2$. This gives us $(\det A) \bmod p$, $(\text{adj } A) \bmod p$, and $A^{-1} \bmod p$. Then the algebraic technique of *p-adic (Newton–Hensel's) lifting* is applied. In ℓ steps, for a sufficiently large ℓ, the matrix $A^{-1} \bmod p$ is lifted to $A^{-1} \bmod p^L$, $L = 2^l$. Then the lifting of $A^{-1} \bmod p$ is extended to lifting similarly $(\det A) \bmod p$ and $(\text{adj } A) \bmod p$. Finally, $\det A$ and adj A are easily recovered from $(\det A) \bmod p^L$ and $(\text{adj } A) \bmod p^L$.

The remaining ingredient is the approximation of $\det F$. In [P85], [P87], this is achieved as a by-product of solving the more general task of computing $\det(xI - F)$, the characteristic polynomial of F. In the present paper, we employ a more routine approach, based on the computation of *recursive* (block) *decomposition* (RD) of F (cf. [St69], [Morf74], [Morf80], [BA80], [P87]) or, equivalently, on the computation of nested *Schur's complements*, also called *Gauss transforms* (cf. [C74], [F64]). A single recursive step of this approach is the decomposition of the input matrix (represented as a 2×2 block matrix) into the product of a block diagonal matrix and two block triangular matrices (see (2.3)). Such a decomposition can be obtained by block Gauss–Jordan elimination and can be reduced to a few matrix multiplications (their parallel implementation is simple) and inversions (they are made simple by Newton's iteration, since good initial approximations are given by matrices $-I/(ap)$). As in [P85], [P87], random choice of a large prime p in a fixed large interval enables us to avoid degeneration and singularities (with a high probability).

An important point, as in [P85], [P87], is that in spite of computing all the matrix inverses approximately, we finally recover them exactly (as well as all the other matrices involved in the RDs) by exploiting the representation of their entries as the ratios of integers. To emplasize this point, we called the resulting RDs the *integral RDs* (*IRDs*). The only remaining nontrivial problem in the computation of the IRD of F and $\det F$ is *to bound the magnitudes* of the integers involved. In the present paper, this problem is solved based on the computation of the *dual RD*, that is, the RD of F^{-1}. (The celebrated techniques of [B68] do not suffice, and their extension to our case is nontrivial not just because we deal with recursive block decomposition, rather than with the more customary Gaussian elimination, but also because we need to control the magnitudes of the determinants of the matrices involved in the RD, which is a much harder problem, and we use the dual RD in order to solve it.)

As soon as the IRDs of F and F^{-1} are available, we obtain the RDs modulo p of A and A^{-1}. Now, we apply the techniques of p-adic (Newton–Hensel's) lifting not only to $A^{-1} \bmod p$ but to the entire RDs modulo p of A and A^{-1}, in order to obtain the RDs modulo p^L of A and A^{-1} for $L = 2^l$ and a sufficiently large l. $(\det A) \bmod p^L$ is recovered from such an RD of A. As $\det A$ is an integer, we recover easily $\det A$ and then the matrix adj $A = A^{-1} \det A$, whose entries are integers.

This approach only gives us an alternative derivation of the estimates of [P85],

[P87] for parallel complexity of some fundamental computations with general matrices. The new algorithms, however (unlike ones of [P85], [P87]), have an advantage of allowing their effective extension to Toeplitz/Toeplitz-like cases. Indeed, manipulation with displacement generators, rather than with matrices themselves, enables the decrease of the processor complexity of the RNC algorithms outlined above to the optimal level linear in n.

The nontrivial problem in such a Toeplitz/Toeplitz-like extension is the control of the length of the displacement generators in the process of Newton's iteration. (Uncontrolled growth of the length would immediately imply the growth of the processor bounds by factor $n^{\omega-1}/\log n$ for $\omega > 2.375$.) We solve this problem by applying two techniques of *truncation of generators (TG)*, which we borrow from [P92] and [P92b], [P93], [P93a], respectively.

The above outline was essentially given by us in [BP94, chapter 4]. Presently, we also add the technique of *stream contraction* specified in section 10 (and, essentially, being the *pipelining* of the two processes of RD and Newton's iteration) borrowed from [PR91]. Stream contraction enables additional acceleration of our algorithms by factor $\log n$. (Using the technique of stream contraction for the acceleration of Toeplitz-like computations was also proposed in [R95], though the algorithms of [R95] did not give any improvement of the processor bounds in the Toeplitz/Toeplitz-like case versus the much larger bounds known in the case of general input matrices (see our Remarks 6.1, 11.1, and 14.2 and our similar comments in [P96b])).

1.7. Organization of the paper. The order of our presentation will slightly differ from the one outlined above. After some preliminaries in section 2, we will introduce the RD and extended RD (ERD) of a matrix in section 3. In section 4, we define the IRD and show the transition from RD to IRD. We recall an algorithm for approximate matrix inversion via Newton's iteration in section 5 and apply it in order to approximate the RD and the IRD of an integer matrix in section 6. We estimate the errors and parallel complexity of these computations in sections 7–9. We apply pipelining (stream contraction) to achieve acceleration by factor $\log n$ in section 10, extend the results of sections 6 and 10 to computing the ERD modulo a fixed prime in section 11, and use p-adic lifting to recover (from the ERD) the inverse, the determinant, the rank, and the null-space of an integer matrix (thus proving Theorem 1.1) in section 12. In section 13, we recall some known definitions and properties for computations with Toeplitz and Toeplitz-like matrices. In section 14, we apply these properties to improve the results of section 12 in the Toeplitz and Toeplitz-like cases (thus proving Theorem 1.2). Section 15 is left for a brief discussion.

2. Some definitions and auxiliary results for matrix computations. We will next recall some customary definitions and well-known basic properties of general matrices.

DEFINITION 2.1 (matrix notation). *I and 0 denote the* identity *and* null *matrices, of appropriate sizes. W^T is the* transpose *of a matrix or vector W.* diag $(w_i)_{i=0}^{n-1}$ = diag $(w_0, \dots, w_{n-1})$ *is the* diagonal matrix *whose diagonal is filled with $w_0, \dots, w_{n-1}$; $D(W)$ = diag (W) = diag $(w_{i,i})_{i=0}^{n-1}$ for a matrix $W = (w_{i,j})$.* rank W *is the* rank *of W.* det W *is the* determinant *of a square matrix W.* adj W *is the* adjoint (adjugate) matrix *of W, equal to W^{-1}* det W *for a nonsingular matrix W.*

In our error analysis, we will use the customary vector and matrix norms [GL89/96].

DEFINITION 2.2 (vector and matrix norms). $\| \mathbf{v} \| = \| \mathbf{v} \|_1 = \sum_i |v_i|$, $\| \mathbf{v} \|_2 = (\sum_i v_i^2)^{1/2}$ *for a real vector* $\mathbf{v} = (v_i)$. $\| W \|_g = \max_{\|\mathbf{v}\|_g = 1} \| W\mathbf{v} \|_g$, $g = 1, 2$; $\| W \| = \| W \|_1$ *for a matrix W.*

PROPOSITION 2.3 (norm bounds). $\| W \| = \| W \|_1 = \max_j \sum_i |w_{i,j}|$ *for a matrix* $W = (w_{i,j})$. *Furthermore, if* W *is a* $k \times k$ *matrix and* V *is its submatrix, then*

$$||V||_g \le ||W||_g, \ g = 1, 2;$$
$$||W||/k^{1/2} \le ||W||_2 \le ||W||k^{1/2}.$$

Proof. To prove the bound $||V||_g \le ||W||_g$, note that $||W\mathbf{w}||_g \ge ||V\mathbf{v}||_g$ if $\mathbf{v}$ is a subvector of $\mathbf{w}$ and if $\mathbf{w}$ has zero components corresponding to the columns of W that are not in V. Other claimed relations can be found in [GL89/96]. ∎

We will also use the following known fact (cf. [GL89/96] or [BP94]).

PROPOSITION 2.4 (bounds on the determinant and the entries of the adjoint matrix). *Let* W *be a* $k \times k$ *matrix. Then* $|\det W| \le (\| W \|_g)^k$, *and furthermore,* $|v| \le (\| W \|_g)^{k-1}$ *for every entry* v *of adj* W, *where* $g = 1, 2$.

DEFINITION 2.5 (column-diagonally dominant (c.-d.d.) matrices). $d(W) = \| WD^{-1}(W) - I \|$. *A matrix* W *is* column-diagonally dominant *(hereafter, we will use the abbreviation* c.-d.d.*) if* $d(W) < 1$.

DEFINITION 2.6 (leading principal submatrix (l.p.s.) and its Schur complement). *For a* $k \times k$ *matrix* W, *let* $W^{(q)}$ *denote its* $q \times q$ *northwestern or* l.p.s., *formed by the intersection of the first* q *rows and the first* q *columns of* W, $q = 1, 2, \dots, k$. *If* B *is a nonsingular l.p.s. of* W *and if*

$$W = \begin{pmatrix} B & C \\ E & G \end{pmatrix}, \tag{2.1}$$

then the matrix

$$S = S(W, B) = G - EB^{-1}C \tag{2.2}$$

is called the Schur complement *of* B *in* W.

The Schur complement S of (2.2) can be obtained by Gaussian or block Gaussian elimination applied to the matrix W, provided that the elimination process can be carried out (cf. [GL89/96, P3.2.2, p. 103]). In particular, it is easily verified that for $k > q$ the Schur complement of $B = W^{(q)}$ in a $k \times k$ matrix W is the $(k-q) \times (k-q)$ matrix obtained from W in q steps of Gaussian elimination (without pivoting) provided that these steps can be carried out (with no division by 0). The latter assumption holds, in particular, if W is a c.-d.d. matrix.

By applying block Gauss–Jordan elimination to the 2×2 block matrix W of (2.1), with a nonsingular block B, we obtain the following decomposition, which will be fundamental for our study:

$$W = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix} \begin{pmatrix} B & 0 \\ 0 & S \end{pmatrix} \begin{pmatrix} I & B^{-1}C \\ 0 & I \end{pmatrix}. \tag{2.3}$$

If a matrix W is nonsingular, then (2.3) implies that the matrix S is also nonsingular. By inverting the matrices on both sides of (2.3), we obtain that

$$\tilde{W} = W^{-1} = \begin{pmatrix} I & -B^{-1}C \\ 0 & I \end{pmatrix} \begin{pmatrix} B^{-1} & 0 \\ 0 & S^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ -EB^{-1} & I \end{pmatrix}. \tag{2.4}$$

Equation (2.4) immediately implies the following proposition.

PROPOSITION 2.7. *Under* (2.1)–(2.4), *the matrix* S^{-1} *is the trailing principal (that is, southeastern) submatrix of* $\tilde{W} = W^{-1}$.

Our algorithms will rely on the decompositions of (2.3), (2.4), recursively applied to the matrices B, B^{-1}, S, and S^{-1}, which we will call RDs. The next definitions and results will cover the nonsingularity properties required for the existence of such recursive extension of (2.3), (2.4) and some other relevant properties of l.p.s.'s and Schur complements (the s.p.d. matrices will be used only at the very end of section 12).

DEFINITION 2.8 (strongly nonsingular matrices). *A matrix W is* strongly nonsingular *if all its leading principal submatrices are nonsingular.*

DEFINITION 2.9 (symmetric positive definite (s.p.d.) matrices). *A real matrix M is s.p.d. if it can be represented as the product AA^T for a nonsingular matrix A.*

The next proposition (cf. [BP94, exercise 4c, p. 212]), extends strong nonsingularity and the s.p.d. property to an l.p.s. and a Schur complement.

PROPOSITION 2.10. *If a matrix W of* (2.1) *is strongly nonsingular (respectively, if W is s.p.d.), then so are its every l.p.s., including the matrix B of* (2.1), *and the Schur complement S of B, defined by* (2.2).

COROLLARY 2.11. *Any s.p.d. matrix is strongly nonsingular.*

By recursively applying block Gaussian elimination at first to the block matrix W of (2.1) with $B = W^{(r)}$ and then to $W^{(r)}$, with the *l.p.* (that is, leading principal or northwestern) block $W^{(q)}$, $q < r$, we obtain the following.

PROPOSITION 2.12 (transitivity of Schur's complementation). *If $r > q$ and if $W^{(r)}$ and $W^{(q)}$ are nonsingular matrices, then $S(W, W^{(q)}) = S(S(W, W^{(r)}), S(W, W^{(r)})^{(r-q)})$.*

We also easily deduce the following.

PROPOSITION 2.13 (transitivity of the c.-d.d. property). *If $d(W) < 1$ for a matrix W of* (2.1), *then B, S, and W are nonsingular matrices, $d(B) \le d(W) < 1$, $d(S) \le d(W) < 1$, and* (2.3)–(2.4) *hold.*

The following result, together with Propositions 2.4 and 2.12, will be basic for our bounds on the values involved in recursive decompositions.

PROPOSITION 2.14. *Assuming* (2.1)–(2.2), *every entry of the matrix S* det B *is a subdeterminant (that is, the determinant of a submatrix) of the matrix W.*

Proof. Let $B = W^{(q)}$. Consider the l.p. (that is, northwestern) entry $s_{0,0}$ of S. By Proposition 2.12, it is the Schur complement of B in the submatrix $W^{(q+1)}$ of W. By Proposition 2.7, $s_{0,0}^{-1} = \det B / \det W^{(q+1)}$. Therefore, $s_{0,0} \det B = \det W^{(q+1)}$, which proves the proposition for $s_{0,0}$. To extend this result to any entry $s_{i,j}$ of S, interchange the ith and 0th rows and the jth and 0th columns of S and the respective pairs of rows and columns of W. □

Clearly, the matrix adj $B = B^{-1}$ det B is filled with integers if W is. Proposition 2.14 (or, alternatively, (2.2)) implies the similar property of the matrix S det B. We summarize these observations for future references.

PROPOSITION 2.15 (integrality of adjoints and scaled Schur complements). *If a matrix W of* (2.1) *is filled with integers, then so are the matrices* adj $B = B^{-1}$det B *and S* det B.

Recursive application of the next result enables us to recover det W from recursive decomposition of W.

PROPOSITION 2.16 (factorization of the determinants and submatrices implied by matrix decomposition). *Matrix equation* (2.3) *implies that* det $W =$ *(*det B*)* det S *and, furthermore, that*

$$W^{(t)} = \begin{pmatrix} I & 0 \\ EB^{-1} & I \end{pmatrix}^{(t)} \begin{pmatrix} B & 0 \\ 0 & S \end{pmatrix}^{(t)} \begin{pmatrix} I & B^{-1}C \\ 0 & I \end{pmatrix}^{(t)} \quad \text{for } t = 1, \dots, k\ ,$$

and $\det W^{(t)} = (\det B) \det S^{(t-q)}$ *for* $t = q+1, \dots, k$, *provided that* W *is a* $k \times k$ *matrix.*

In section 12, we will also use the following definitions and known results.

DEFINITION 2.17 (the null-space of a matrix; cf. [GL89/96] or [BP94]). *The null-space* $\mathbf{N}(A)$ *of a matrix* A *is the linear space formed by all vectors* $\mathbf{x}$ *satisfying the vector equation* $A\mathbf{x} = \mathbf{0}$.

FACT 2.1. *Two vector equations,* $A\mathbf{x} = \mathbf{f}$ *and* $A\mathbf{y} = \mathbf{f}$, *together imply that* $\mathbf{x} - \mathbf{y} \in \mathbf{N}(A)$, *or, equivalently, any solution* $\mathbf{x}$ *to a consistent linear system* $A\mathbf{x} = \mathbf{f}$ *can be represented in the form* $\mathbf{x} = \mathbf{x}_0 + \mathbf{z}$, *where* $\mathbf{x}_0$ *is a fixed specific solution and* $\mathbf{z} \in \mathbf{N}(A)$.

Toeplitz matrix computations will be studied in sections 13 and 14, but also the proposition below involves Toeplitz matrices and is needed in section 12.

DEFINITION 2.18 (Toeplitz matrices). $T = (t_{i,j})$ *is a* $k \times k$ *Toeplitz matrix if* $t_{i+1,j+1} = t_{i,j}$ *for* $i, j = 0, 1, \dots, k-2$ *(cf.* (1.1)*), that is, if the entries of* T *are invariant in their shifts in the diagonal direction. (Such a matrix is defined by its two columns (or rows)—the first one and the last one.) A square lower triangular Toeplitz matrix is defined by its first column* $\mathbf{u}$ *and is denoted* $L(\mathbf{u})$. $Z = Z_k = (z_{i,j})$ *is a* $k \times k$ *lower triangular Toeplitz matrix with the first column* $(0, 1, 0, \dots, 0)^T$, *so that* $z_{i+1,i} = 1$ *for* $i = 0, 1, \dots, k-2$, $z_{i,j} = 0$ *if* $i \neq j+1$.

PROPOSITION 2.19 ([KS91], (cf. [BP94, Lemmas 1.5.1 and 2.13.1])). *Let* S *be a fixed finite set of cardinality* $|S|$. *Let* A, L, *and* U *be* $n \times n$ *matrices, let* $\operatorname{rank} A = r$, *let* L *and* U^T *be unit lower triangular Toeplitz matrices, each defined by the* $n-1$ *entries of its first columns. Let these entries be chosen from* S *at random, independently of each other, under the uniform probability distribution on* S. *Then the matrix* $(UAL)^{(r)}$ *is strongly nonsingular with a probability at least* $1-(r+1)r/|S|$.

3. RD and ERD of a c.-d.d. matrix. With minor deviation from the order of our outine of section 1.6 but in accordance with section 1.7, we will next study the RD, then, in section 4, the integral RD (IRD), and in section 5, matrix inversion.

Hereafter, for convenience, let log stand for $\log_2$, let $n = 2^h$ for an integer $h = \log n$, and let V be a fixed $n \times n$ c.-d.d. matrix.

We will define an RD of such a matrix $W = V$ based on its representation in the form (2.3) for $q = n/2$. We will first apply (2.3) to $W = V$ and then, recursively, to $W = B$ and $W = S$, and so on, though in fact, we will mostly care about the diagonal blocks V, $V_0 = B$, $V_1 = S$, and so on, which we will identify with the nodes of a binary tree, T. Similarly, we define the *dual RD* of $\tilde{W} = W^{-1}$ based on the recursive application of (2.4) to $\tilde{W} = B^{-1}$ and $\tilde{W} = S^{-1}$. (We will study such a dual RD in section 11 and will use it in section 12.)

The node of T associated with a binary strings α of length $|\alpha|$ is a $k \times k$ matrix, denoted by V_α, where $k = n/2^{|\alpha|}$. The root of the tree is the $n \times n$ matrix $V = V_\Lambda$, associated with the empty string Λ. For a string β of length less than h, we let $\beta 0$ and $\beta 1$ denote the two strings obtained by appending 0 and 1 to β, respectively. (We use the two characters α and β to distinguish between the two classes of binary strings—of length at most h and less than h, respectively.) We will assume that the matrix equations (2.1)–(2.4) are satisfied for $W = V_\beta$, $B = V_{\beta 0}$, $S = V_{\beta 1}$, and for any binary string β of length $|\beta| < h$. The resulting RD of the matrix V continues up to the level h, where it reaches its leaves-matrices $V_\alpha = (v_\alpha)$ of size 1×1, where $|\alpha| = h$, and where v_α denotes the single entry of V_α.

PROPOSITION 3.1. *All the nodes* V_α *of the tree* T *are c.-d.d. matrices; moreover,* $d(V_\alpha) \leq d(V) < 1$ *for all binary strings* α.

Proof. Recall that V is a c.-d.d. matrix and recursively apply Proposition 2.13. □

Let us formalize the computation of the RD of V by extending the notation (2.1)–(2.2) to V_α. We will write

$$V_\beta = \begin{pmatrix} B_\beta & C_\beta \\ E_\beta & G_\beta \end{pmatrix}, \tag{3.1}$$

$$V_{\beta 0} = B_\beta\ , \quad V_{\beta 1} = S_\beta = G_\beta - E_\beta B_\beta^{-1} C_\beta\ , \tag{3.2}$$

for all binary strings β of length less than h.

Then, computation of the RD of a c.-d.d. matrix V amounts to recursive computation of $V_{\beta 1}$ of (3.2) for all binary strings β of length increasing from 0 to $h-1$. As a by-product, the computation produces the matrices $V_{\beta 0}^{-1} = B_\beta^{-1}$ for all binary strings β of length less than h. By appending these inverse matrices $V_{\beta 0}^{-1}$ to the nodes $V_{\beta 0}$ of the tree T (and, consequently, to the RD of V), we arrive at the ERD of V. The set of the matrices $V_{\beta 0}^{-1}$ will be called the *extending set of the RD of* V.

The RD and the ERD of any matrix V can be defined as long as all the involved nodes-matrices $V_{\beta 0}$ for all the binary strings β of length less than h are nonsingular. Recursive application of Proposition 2.10 and Corollary 2.11 yields the following.

PROPOSITION 3.2. *There exists the RD and the ERD of any strongly nonsingular (in particular, of any s.p.d.) matrix.*

REMARK 3.1. *As soon as we have the RD of a c.-d.d. matrix* V*, we immediately obtain the RD of* V^{-1} *based on recursive application of* (2.4). *Having such an RD available, we may compute the solution* $\mathbf{x} = V^{-1}\mathbf{f}$ *of a linear system* $V\mathbf{x} = \mathbf{f}$*, at a lower computational cost* $O_A((\log n)^2, n^2/(\log n)^2)$.

4. IRD of a c.-d.d. matrix filled with integers. Suppose that the input matrix V is filled with integers. Then, for all binary strings α, the matrices V_α are filled with rationals, and there exist integer multipliers m_α such that $m_\alpha V_\alpha$ are integer matrices. We will next specify a particular choice of such integer multipliers m_α.

We will use the notation $W^{(q)}$ of Definition 2.6 and the following definition.

DEFINITION 4.1. $(\alpha)_2$ *denotes the binary value represented by a binary string* α *(of length at most* h*).* $\alpha(q)$ *denotes the binary string that represents a nonnegative binary value* q*, so that* $(\alpha(q))_2 = q$. $H(\alpha)$ *denotes* $2^{h-|\alpha|} = n/2^{|\alpha|}$. $Q(\alpha)$ *denotes* $(\alpha)_2 H(\alpha)$.

By applying Proposition 2.12, we obtain the following.

PROPOSITION 4.2. *Let a binary string* α *end with bit* 1 *and have length at most* h*. Then the matrix* V_α *is the Schur complement of* $V^{(Q(\alpha))}$ *in* $V^{(Q(\alpha)+H(\alpha))}$.

By combining this proposition with Proposition 2.14, we obtain the following.

PROPOSITION 4.3. *For any binary string* $\alpha = \beta 1 \gamma$ *of length at most* h *with* γ *being a string of zeros, let*

$$m_\alpha = \det V^{(Q(\beta 1))}. \tag{4.1}$$

Then the entries of the matrix $V_\alpha m_\alpha$ *are the subdeterminants (that is, the determinants of some submatrices) of the matrix* V*. In particular, if* V *is filled with integers, then so are the matrices* $V_\alpha m_\alpha$ *for all* $\alpha, |\alpha| \le h$.

Now we replace the matrix $V_{\alpha\gamma}$ by the pair of the scalar $m_\alpha = m_{\alpha\gamma}$ and the matrix $m_\alpha V_{\alpha\gamma}$, in the binary tree T, for every pair of binary strings α and γ such that

α ends with 1, $|\alpha| + |\gamma| \le h$, and γ consists only of zeros. This gives us the IRD of a c.-d.d. integer matrix V. By definition, we will also include into the IRD of V the two sets, $\{\det V^{(k)},\ k = 1, \dots, n\}$ and $\{\det V_\alpha, |\alpha| \le h\}$, of the determinants associated to the RD of V. Clearly, having the IRD of V available, we may immediately compute the RD of V. Later in this section, we will specify a simple transition from the RD to the IRD. In section 12, we will also compute a dual IRD by similarly extending the dual RD.

By combining Propositions 2.4 and 4.3, we obtain that for all binary strings $\alpha,\ |\alpha| \le h$, we have

$$|m_\alpha| \le \| V \|^n,\ ||m_\alpha V_\alpha|| \le n||V||^n. \tag{4.2}$$

For a c.-d.d. integer matrix V given together with its RD, we will seek its IRD. We recall that v_α for $|\alpha| = h$ denotes the single entry of the 1×1 leaf-matrix V_α of the tree T of the RD. Recursive application of Propositions 2.12 and 2.16 immediately yields the two following results.

PROPOSITION 4.4. *For every binary string α of length at most h, we have*

$$\det V_\alpha = \prod_\beta v_\beta,$$

where $\prod_\beta$ denotes the product in all binary strings β of length h that have α as their prefix; that is, the associated nodes V_β are both leaves of the tree T and descendants of the node V_α in the tree T.

PROPOSITION 4.5. *$\det V^{(q)} = \prod_{(\alpha)_2 < q} v_\alpha$, where $\prod_{(\alpha)_2 < q}$ denotes the product in all binary strings α of length h for which $(\alpha)_2 < q$.*

By applying the well-known parallel prefix algorithm [EG88], [KR90], we deduce the following result from Propositions 4.2–4.5 and 2.15.

COROLLARY 4.6. *Given the RD of a c.-d.d. $n \times n$ matrix V filled with integers, one may compute the IRD of V at the cost $O_A(\log n, n/\log n)$.*

Due to the latter result and to matrix equations (2.1)–(2.3), the computation of the IRD of V can be reduced to a sequence of multiplications, inversions, and subtractions of integer matrices, and we obtain the following parallel arithmetic complexity estimates:

$$t_{\mathrm{IRD}}(n) \le t_I(n/2) + t_{\mathrm{IRD}}(n/2) + 2t_M(n/2) + 1, \tag{4.3}$$

$$p_{\mathrm{IRD}}(n) \le \max\{p_I(n/2), 2p_{\mathrm{IRD}}(n/2), 2p_M(n/2), n^2\}, \tag{4.4}$$

where $O_A(t_{IRD}(k), p_{IRD}(k)), O_A(t_I(k), p_I(k))$, and $O_A(t_M(k), p_M(k))$ denote the time and processor bounds for the computation of the IRD, the inverse and the product of $k \times k$ matrices, respectively, where

$$(O_A(t_M(k), p_M(k)) = O_A(\log k, k^\omega), \quad 2 \le \omega < 2.376 \tag{4.5}$$

[BP94], and where we also use the obvious complexity bounds $O_A(t_S(k), p_S(k)) = O_A(1, k^2)$ for the subtraction of $k \times k$ matrices.

As the computation of the IRD is our major goal, relations (4.3)–(4.5) motivate the next subject of our study, that is, matrix inversion.

5. Approximate matrix inversion via Newton's iteration.

Algorithm 5.1. *Newton's iteration for approximate matrix inversion.*
Input: a nonsingular $k \times k$ matrix B, two positive scalars b and c, $b > c$, and a matrix X_0 (a rough initial approximation to $-B^{-1}$) such that

$$c = -\log \| BX_0 + I \| . \tag{5.1}$$

Output: a matrix X such that

$$\| BX - I \| \leq 2^{-b} \tag{5.2}$$

and, consequently,

$$\| X - B^{-1} \| \leq 2^{-b} \| B^{-1} \| . \tag{5.3}$$

Computations:
1. Compute

$$g = \lceil \log(b/c) \rceil. \tag{5.4}$$

2. Recursively compute the matrices

$$X_i = X_{i-1}(2I + BX_{i-1}) , \quad i = 1, \dots, g. \tag{5.5}$$

3. Output the matrix $X = -X_g$.

To prove *correctness* of Algorithm 5.1, deduce from (5.5) that

$$I + BX_i = (I + BX_{i-1})^2 , \quad i = 1, 2, \dots, g,$$

and, consequently,

$$I + BX_i = (I + BX_0)^{2^i},$$

$$\| I + BX_i \| \leq \| I + BX_0 \|^{2^i}, \tag{5.6}$$

for $i = 1, 2, \dots, g$. In particular, for $i = g$, we have

$$\| I + BX_g \| \leq \| I + BX_0 \|^{2^g} = 2^{-c2^g} \leq 2^{-b}$$

due to (5.1) and (5.4). This gives us (5.2) and (5.3) for $X = -X_g$.

To estimate the overall *computational cost* of performing Algorithm 5.1, observe that the ith step (5.5) amounts to two matrix multiplications and to adding, at the cost $O_A(1, k)$, the matrix $2I$ to the matrix BX_{i-1}. Summarizing, we obtain the following result.

PROPOSITION 5.1. *For g of* (5.4), *g steps* (5.5) *of Newton's iteration, performed at the overall cost $O_A(gt_M(k), p_M(k)) = O_A(g \log k, k^\omega)$ for ω of* (4.5), $2 \leq \omega < 2.376$, *suffice in order to compute a matrix $X = -X_g$ satisfying* (5.2) *and* (5.3).

REMARK 5.1. *Proposition* 5.1 *enables us to estimate the complexity of approximate matrix inversion in terms of a scalar b and a matrix X_0. Their choice depends on the input matrix B and is critical for estimating the approximation errors and the number of iterations. We will elaborate this choice in the next sections. Here are some*

preliminary comments on the choice of X_0. If B is a c.-d.d. matrix so that $d(B) < 1$, then (5.1) *holds for $X_0 = -D^{-1}(B)$ and $c = -\log d(B)$; that is, $||BX_0 + I|| \le d(B)$. This gives us a good policy for the choice of X_0 and c over the class of c.-d.d. matrices B. In this paper, however, we will only need to invert the c.-d.d. matrices B that are close to the scaled identity matrices $-mI$ for a fixed large integer m. The inverse of such a matrix B is well approximated by the scaled identity matrices $X_0 = -I/m$ satisfying*

$$||BX_0 + I|| < 1/(5n^2),\ c > 2\log n. \tag{5.7}$$

In fact our algorithms and complexity estimates will remain valid under some assumptions that are weaker than (5.7). *Say, the bound*

$$c = -\log ||BX_0 + I|| > \theta > 0 \tag{5.8}$$

for a fixed constant θ would suffice. The choice of $X_0 = D^{-1}(B)$ also satisfies the error bound of (5.7), *but $X_0 = -I/m$ is a Toeplitz matrix (see Definition* 2.18), *which will be a crucially important advantage for the proof of Theorem* 1.2 *in sections* 13 *and* 14.

6. Approximate RD of an integer matrix and its extension to the exact evaluation of the IRD. Algorithm 5.1 is intended as matrix inversion block in the algorithms of sections 3–4 for computing the ERD, IRD, and the associated determinants of a c.-d.d. integer matrix V. Then, the matrices V_α and the values of $\det V^{(q)}$ and $\det V_\alpha$ for all q and α are computed approximately, even where we still perform all arithmetic operations over the rationals, with infinite precision and no errors. Our next goal is to yield the exact IRD, assuming that we fixed a sufficiently large b and defined X_0 and c according to Remark 5.1.

Let us write $\tilde{V}_\alpha$, $\tilde{\det}\ V^{(q)}$, and $\tilde{\det}\ V_\alpha$ for the computed approximations to V_α, $\det\ V^{(q)}$, and $\det\ V_\alpha$, respectively. Then we extend the relations (3.1), (3.2), (5.1), (5.4), and (5.5) by writing

$$\tilde{V}_\beta = \begin{pmatrix} \tilde{B}_\beta & \tilde{C}_\beta \\ \tilde{E}_\beta & \tilde{G}_\beta \end{pmatrix}, \tag{6.1}$$

$$\tilde{V}_{\beta 0} = \tilde{B}_\beta\ , \qquad \tilde{V}_{\beta 1} = \tilde{G}_\beta - \tilde{E}_\beta X_\beta \tilde{C}_\beta\ , \tag{6.2}$$

for all binary strings β of length less than h, where the policy of defining $X_{\beta,0}$ (in accordance with Remark 5.1) will be specified later on, and where

$$c(\beta) = -\log \parallel \tilde{B}_\beta X_{\beta,0} + I \parallel, \tag{6.3}$$

$$g(\beta) = \log(b/c(\beta)), \tag{6.4}$$

$$X_{\beta,i} = X_{\beta,i-1}(2I + \tilde{B}_\beta X_{\beta,i-1})\ , \qquad i = 1, \dots, g(\beta), \tag{6.5}$$

$$X_\beta = -X_{\beta,g(\beta)}, \tag{6.6}$$

and I is the identity matrix of an appropriate size.

Algorithm 6.1. *Approximating the RD of a c.-d.d. matrix.*

Input: a positive integer h, $n = 2^h$, a positive b, positive integers $g(\alpha)$ for all binary strings α of length at most h, and an $n \times n$ c.-d.d. matrix $V = \tilde{V}_\Lambda$.

Output: a set of matrices $\tilde{V}_\alpha$ for all binary strings α of length at most h, satisfying (6.1)–(6.6), and the values $\tilde{\det} V_\alpha$ for all α and $\tilde{\det} V^{(q)}$, $q = 1, \dots, n$, defined according to Proposition 4.5 with det and v replaced by $\tilde{\det}$ and $\tilde{v}$, respectively.

Computations: Starting with $V = \tilde{V}_\Lambda$ for the empty string Λ, recursively apply (6.1)–(6.6) and a fixed policy of defining $X_{\beta,0}$, in order to compute $\tilde{V}_{\beta 0}$ and $\tilde{V}_{\beta 1}$ for all binary strings β of length less than h. (For binary strings β of length $h-1$, the matrices $\tilde{V}_{\beta 0}$ and $\tilde{V}_{\beta 1}$ have size 1×1, so $\tilde{V}_{\beta 0}$ is inverted immediately, and then the matrix $\tilde{V}_{\beta 1}$ is computed based on (2.1) and (2.2) for $W = \tilde{V}_\beta$.) Finally, compute $\tilde{\det} V_\alpha$ and $\tilde{\det} V^{(q)}$ for all α and q by applying Propositions 4.4 and 4.5, under the above modification of the notation.

Correctness of the algorithm is immediately verified, provided that the value b is chosen sufficiently large so that all matrices $\tilde{B}_\beta$—which approximate the c.-d.d. matrices B_β—still have the property of being c.-d.d. and that the matrices $X_0 = X_{\beta,0}$ are chosen satisfying (5.8) for $B = \tilde{B}_\beta$ for all binary strings β. We note that (5.6) and (6.4) together imply that $||I + \tilde{B}_\alpha X_\alpha|| \le 2^{-b}$ and, consequently,

$$||X_\alpha - \tilde{B}_\alpha^{-1}|| \le 2^{-b} ||\tilde{B}_\alpha^{-1}||, \tag{6.7}$$

which extends (5.3).

The computational cost is bounded by $O_A((\log n)^2 g, n^\omega)$ for $g = \max_{|\beta|<h} g(\beta)$ and for ω of (4.5). (Recursively apply (4.3)–(4.5) and Proposition 5.1.) A desired upper bound on g (and, consequently, on the parallel time) will be ensured by (5.8), (6.3), (6.4), and appropriate choice of b. Such a choice and its analysis will be shown in the next sections. $g(\beta)$ will in fact be independent of β, that is, we will choose $g(\beta) = g$ for all β.

Next, for a c.-d.d. matrix V filled with integers, we will apply Algorithm 6.1 for a sufficiently large b, and then we will apply the techniques of *integer rounding* (compare [P85], [P87], and [BP94, p. 252]), to extend the resulting approximate RD of V to the evaluation of the IRD of V. To yield this extension, we will choose b in Algorithm 6.1 sufficiently large to ensure the following bounds:

$$|\tilde{\det}\, V^{(q)} - \det V^{(q)}| < 1/2\ , \quad q = 1, \dots, n\ , \tag{6.8}$$

$$\| \tilde{V}_{\beta 1} - V_{\beta 1} \| < \| V \|^{-n}/2 \ \text{ for all binary strings } \beta, \ \ |\beta| < h, \tag{6.9}$$

where $\tilde{\det}\, V^{(q)}$ and $\tilde{V}_{\beta 1}$ denote the approximations to $\det V^{(q)}$ and $V_{\beta 1}$, respectively, computed by Algorithm 6.1 for the fixed value of b.

Under (6.8) and (6.9), we recover the IRD of V as follows.

Algorithm 6.2.

Input: a set $\{\tilde{\det}\, V^{(q)},\ q = 1, \dots, n\}$ of approximations to $\det V^{(q)}$ for all q and an approximate RD of an $n \times n$ matrix V filled with integers, such that (6.8) and (6.9) hold.

Output: the IRD of V.

Computations:

1. Round the values $\det \tilde{V}^{(q)}$ to the closest integers; output the resulting integer values of $\det V^{(q)}$, $q = 1, \dots, n$.

2. Compute the matrices $\tilde{W}_{\beta 1} = \tilde{V}_{\beta 1} \det V^{(Q(\beta 1))}$ and round their entries to the closest integers; output the resulting integer matrices $W_{\beta 1} = V_{\beta 1} \det V^{(Q(\beta 1)}$ for all binary strings β of length less than h.
3. For all binary strings β of length less than h and all binary strings γ filled with zero bits and satisfying $|\beta 1 \gamma| \le h$, output the matrices $W_{\beta 1\gamma} = V_{\beta 1\gamma} \det V^{(Q(\beta 1))}$ (cf. (4.1) and Definition 4.1).

Correctness of Algorithm 6.2 follows since, clearly, the values $\det V^{(q)}$ are integers for all q and since the matrices $W_{\beta 1} = V_{\beta 1} \det V^{(Q(\beta 1))}$ are filled with integers for all β (due to Proposition 4.3). The *computational cost* of performing the algorithm is bounded by $O_A(1, n^2)$.

REMARK 6.1. *Instead of choosing the multipliers $m_{\beta 1}$ based on Proposition* 4.3, *one may follow the more straightforward recipe of* [R95] *and recursively define m_β by using induction on $|\beta|$ and by writing $m_{\beta 1} = m_\beta \det V_{\beta 0}$. Then, however, the order of $\log |m_\beta|$ grows from $|\beta|$ (compare our bounds* (4.2)*) to $|\beta|^2$, and the bit-precision and the bit-complexity of the computations grow by the extra factor n. (The statement of Proposition* 5.1 *of* [R95] *is false. Its proof relies on an erroneous claim that if a matrix mA is filled with integers, then so is the matrix m* adj *A; this claim is false, say, for $m = 3$ and the matrix $A =$* diag $(1/3, \dots, 1/3)$.)

7. Errors of the approximation of the RD and the transition from the RD to the IRD for a c.-d.d. matrix. We are going to implement the next step of the outline of section 1.6 by specifying a c.-d.d. matrix V, whose IRD will give us the IRD of A modulo a prime p. We recall that, according to Definition 2.1, we write I to denote the identity matrices of appropriate sizes. We will next specify (in terms of n and $\|V\|$) a choice of the input parameter b of Algorithm 6.1 that will enable us to satisfy the relations (6.8) and (6.9), where V is an $n \times n$ matrix of the following class.

$$V = F - mI. \tag{7.1}$$

F is an $n \times n$ matrix filled with nonnegative integers that are less than a fixed prime $p \ge n$ (we will work with $F = A \bmod p$ for an input matrix A), and

$$m = 10p^2n^2. \tag{7.2}$$

REMARK 7.1. *The choice of a larger m would have made V more strongly diagonally dominant (which is what we would like to have) but would have involved larger integers, which would have increased the Boolean cost of the resulting computations, so we choose only a moderately large m. In fact, our construction allows us to choose even a little smaller m.*

Next, let us prove that the entries of the matrices V of this class and of all matrices V_α of their RDs satisfy the following rough estimate, which will suffice for our purpose.

PROPOSITION 7.1. *The entries of the matrices $V_\alpha + mI$ lie in the range between $-1/2$ and $p - 1/2$ for all binary strings α of length at most $h = \log n$.*

Proof. By the definition of the matrix V, the entries of the matrix $F = V + mI$ range from 0 to $p-1$. By Proposition 4.2, it suffices to prove that the entries of every Schur complement S of an l.p.s. $B = V^{(q)}$ in V range from $-1/2$ to $p - 1/2$. Since $S = G - EB^{-1}C$ (assuming (2.1) for $W = V$) and since the entries of the submatrix $G + mI$ of F range from 0 to $p - 1$, it suffices to prove that the entries of the matrix

$EB^{-1}C$ range between $-1/2$ and $1/2$. Since the matrices $F^{(q)} = B + mI$, C, and E are submatrices of F, their entries also range from 0 to $p-1$. Therefore,

$$||C|| < (p-1)n, ||E|| < (p-1)n,$$

$$-mB^{-1} = (I - F^{(q)}/m)^{-1} = \sum_{i=0}^{\infty} (F^{(q)}/m)^i,$$

$$||B^{-1}|| \le (1/m)(1/(1-a)),\ a = ||F^{(q)}||/m < (p-1)n/m < 0.03.$$

Consequently, $||B^{-1}|| < 2/m$, $||EB^{-1}C|| \le 2(p-1)^2n^2/m < 1/2$. ∎

Hereafter, we will write

$$w = m + (p-1)n. \tag{7.3}$$

Here are three corollaries of Proposition 7.1; the first and the third of them are immediate.

COROLLARY 7.2. *Let $|\alpha| = h$, so that $V_\alpha = (v_\alpha)$ is a 1×1 matrix. Then we have*

$$|v_\alpha| < m + p.$$

COROLLARY 7.3. $\| V_{\beta 0}^{-1} \| < 1.1/m$ *for all binary strings β of length at most $h-1$.*

Proof. Write $F_{\beta 0} = V_{\beta 0} + mI$. Due to Proposition 7.1, we have $\| F_{\beta 0} \| < (p-1)n < m/(10np)$. On the other hand,

$$V_{\beta 0}^{-1} = \frac{1}{m}(I - F_{\beta 0}/m)^{-1} = \left(\frac{1}{m}\right) \sum_{i=0}^{\infty} \left(\frac{F_{\beta 0}}{m}\right)^i.$$

Therefore,

$$\| V_{\beta 0}^{-1} \| \le \left(\frac{1}{m}\right) \sum_{i=0}^{\infty} \left(\frac{\| F_{\beta 0} \|}{m}\right)^i < \left(\frac{1}{m}\right) \sum_{i=0}^{\infty} \frac{1}{(10np)^i} = \frac{10np}{(10np-1)m} < \frac{1.1}{m}\ .$$ ∎

COROLLARY 7.4. $2^{c_\beta} = ||V_{\beta 0}X_{\beta,0} + I|| < 1/(10pn) \le 1/(10n^2)$ *for $X_{\beta,0} = -I/m$ and for all binary string β of length at most $h-1$.*

Hereafter, we will assume that $n > 1$ and that the matrices $X_{\beta,0}$ for all β are chosen as in Corollary 7.4, so that the relations (5.8) and even (5.7) hold. Our next task is to estimate the desired range for b, which would enable us to recover the IRD. In this section we will prove the following basic proposition.

PROPOSITION 7.5. *Under* (7.1)–(7.3), *both requirements* (6.8) *and* (6.9) *are satisfied if the matrices $V_{\beta 1}$ for all binary strings β of length less than h are approximated by Algorithm* 6.1 *within an error norm bound*

$$\sigma = 0.5/w^n. \tag{7.4}$$

Proof. We deduce from (7.1) and (7.3) that

$$\| V \| \le m + (p-1)(n-1) < w.$$

Therefore, we will satisfy (6.9) if we approximate the matrices $V_{\beta 1}$ within the error norm bound (7.4). To prove that (6.8) is satisfied too, we need the next lemma.

LEMMA 7.6. *The requirement* (6.8) *is satisfied if the values* v_α *for all binary strings* α *of length* h *are approximated within an error bound*

$$\delta \le \overline{w}^{1-n}/(2n+2),\ \overline{w} = m+p < w. \tag{7.5}$$

Proof of the lemma. By the virtue of Proposition 4.5, $\det V^{(q)}$ is the product of exactly q values v_α for α denoting binary strings of length h. Under the assumptions of Lemma 7.6, the maximum error of computing $\det V^{(q)}$ may only increase if we assume that $q = n$, that $v_\alpha = \overline{w}$, and that the approximations to v_α equal $\overline{w} + \delta$ for all α. Then, $\det V = \overline{w}^n$ is approximated by $(\overline{w}+\delta)^n$, with an approximation error

$$E = (\overline{w}+\delta)^n - \overline{w}^n = \overline{w}^n((1+(\delta/\overline{w}))^n - 1) = \overline{w}^{n-1}\delta\sum_{i=1}^{n}(\delta/\overline{w})^{i-1}\begin{pmatrix} n \\ i \end{pmatrix}.$$

We have $\binom{n}{1} = n$, $\binom{n}{i} < 2^n$ for all i, and $\delta/w < 1$. Therefore, $E < (n + (\delta/\overline{w})(n-1)2^n)\delta\overline{w}^{n-1}$.

Equations (7.2) and (7.5) together imply that

$$\overline{w} > (n-1)2^n\delta\ ,$$

and we may rewrite our bound on E as follows:

$$E < (n+1)\delta\overline{w}^{n-1}\ .$$

Substitute (7.5) and obtain that $E < 1/2$. ∎

To complete the proof of Proposition 7.5, we observe that

$$\sigma = 0.5/w^n < w^{1-n}/(2n+2) < \overline{w}^{1-n}/(2n+2)$$

(compare (7.5)), and the values v_α for all binary strings α of length h (except for the string $\alpha(0)$ consisting of h zeros) are among the entries of the matrices $V_{\beta 1}$ for $|\beta| \le h-1$. $v_{\alpha(0)}$ is an entry of V and is known exactly without any computation. Therefore, the assumptions of Lemma 7.6 and, consequently, the requirement (6.8) are satisfied too. ∎

8. Estimating the error accumulation and the precision of the approximation of the matrix inverse. In this section, we will extend Proposition 7.5 by estimating the parameter b of Algorithm 6.1 (which expresses the precision of the matrix inversion) to ensure (7.4) and, consequently, (6.8) and (6.9).

PROPOSITION 8.1. *Under some choice of* $b = O(n \log\ p)$, *the bounds* (6.8) *and* (6.9) *can be satisfied in all applications of Newton's iteration* (6.5) *within Algorithm* 6.1, *which is in turn applied to approximate the RD of a matrix* V *satisfying* (7.1) *and* (7.2).

The remainder of this section is devoted to the proof of Proposition 8.1. Due to the bound (5.6), it is actually quite clear that we would ensure (7.4) if we choose sufficiently large values $b = O(n\log p)$, $g(\alpha)$ of (6.4), and m of (7.1), but we will deduce (7.4) already for m of (7.2) and $g(\alpha) = O(\log(n\log p))$ for all α. As usually in the proofs involving error analysis, some tedious estimates are required. The idea of our proof is to condense estimating the error propagation into a single step, which will allow its recursive extension to cover all the nodes V_α of the tree representing the

RD. This basic step of our analysis will be given in the form of Proposition 8.2. (We will first give some preliminaries, then will state and prove this proposition, and then will show that its conclusion enables us to extend recursively its assumptions (and, consequently, its conclusion too) and thus to extend the error estimates recursively to all the descendants of the current node V_α of the tree.)

Proof of Proposition 8.1. Consider a path in the tree T from the root V to a leaf $V_\alpha = (v_\alpha)$, $|\alpha| = h$. Algorithm 6.1 follows such a path by recursively proceeding from matrices V_β to $V_{\beta 0}$ and $V_{\beta 1}$. For given V_β and $V_{\beta 0}$, the algorithm approximates the matrices $V_{\beta 0}^{-1}$ within the error norm bounds $2^{-b} \parallel V_{\beta 0}^{-1} \parallel$ and then extends such an approximation to approximating $V_{\beta 1}$. The errors of the computed approximations to V_β are accumulated in computation of all descendands of V_β along the paths in T. We need to estimate the resulting overall errors in all the output matrices along all such paths, assuming that b is large though of order $O(n \log\ p)$.

We will next analyze a single recursive step along such a path; that is, we will first bound the matrix $\Delta(V_\beta)$ of the initial errors of the approximation of $W = V_\beta$, and then we will estimate the propagated errors of the approximation of $S = V_{\beta 1}$ caused by the combined errors due to the initial ones, given by $\Delta(V_\beta)$, and ones of Newton's iterates for the inversion of $V_{\beta 0}$.

Hereafter, we will write $\tilde{M}$ to denote the approximations to matrices M computed by Algorithm 6.1, for M denoting V_α (for any binary string α), a submatrix of V_α, or any other auxiliary matrix involved. We will also write

$$\Delta(M) = \tilde{M} - M \ . \tag{8.1}$$

We will first estimate $\parallel \Delta(V_{\alpha 1}) \parallel$ in terms of $\parallel \Delta(V_\alpha) \parallel$. For convenience, we write $W = V_\alpha$, $S = V_{\alpha 1}$, recall (2.1), (2.2), and estimate the error propagation in the transition from W to B and S. From Proposition 7.1 and Corollary 7.3, we obtain that

$$\max\{\parallel B \parallel, \parallel C \parallel, \parallel E \parallel, \parallel G \parallel\} \leq \parallel W \parallel \leq w \ , \tag{8.2}$$

$$\parallel B^{-1} \parallel < 1.1/m \ . \tag{8.3}$$

We also write (cf. (8.1))

$$\tilde{W} = \begin{pmatrix} \tilde{B} & \tilde{C} \\ \tilde{E} & \tilde{G} \end{pmatrix}, \quad \Delta(W) = \begin{pmatrix} \Delta(B) & \Delta(C) \\ \Delta(E) & \Delta(G) \end{pmatrix},$$

$$\tilde{S} = \tilde{G} - \tilde{E}L\tilde{C},$$

where L denotes the computed approximation to $\tilde{B}^{-1}$. Then, clearly,

$$\max\{\parallel \tilde{B} \parallel, \parallel \tilde{C} \parallel, \parallel \tilde{E} \parallel, \parallel \tilde{G} \parallel\} \leq \parallel \tilde{W} \parallel, \tag{8.4}$$

$$\max\{\parallel \Delta(B) \parallel, \parallel \Delta(C) \parallel, \parallel \Delta(E) \parallel, \parallel \Delta(G) \parallel\} \leq \parallel \Delta(W) \parallel \ . \tag{8.5}$$

We will assume that the errors of the approximations obtained via Algorithm 6.1 are sufficiently small so that the following inequalities hold (also cf. the bound $||B^{-1}|| < 1.03/m$ obtained in the proof of Proposition 7.1):

$$\parallel L - \tilde{B}^{-1} \parallel \leq \nu \parallel \tilde{B}^{-1} \parallel, \quad \nu \leq 1/(4000m^2), \tag{8.6}$$

$$\| \tilde{B}^{-1} \| < 1.11/m, \quad \| L \| \le (1+\nu) \| \tilde{B}^{-1} \| < 1.11(1+\nu)/m. \tag{8.7}$$

REMARK 8.1. *The inequalities of* (8.6) *are reconciled with* (5.3) *for* $\nu \le 2^{-b}$, L *replacing* X, *and* $\tilde{B}$ *replacing* B.

In the next proposition we will bound approximation errors for $V_{\beta 0}$ and $V_{\beta 1}$ in terms of a single positive parameter $\Delta = \Delta(\beta)$ defined by the errors of the approximation of V_β and by the parameter ν, which is in turn defined by the error exponent b of Newton's iteration for the approximation of $\tilde{B}^{-1}$.

PROPOSITION 8.2. *Suppose that the inequalities* (8.2)–(8.7) *hold and that a positive* $\Delta = \Delta(\beta)$ *satisfies the following bounds:*

$$||\Delta(W)|| \le \Delta, \ 40\nu m \le \Delta < 0.01 \ w, \tag{8.8}$$

where $W = V_\beta$ *and* β *is a binary string of length less than* h. *Then, we have*

(a) $\| \Delta(B) \| = \| \Delta(V_{\beta 0}) \| \le \Delta$,

(b) $\| \Delta(S) \| = \| \Delta(V_{\beta 1}) \| \le 5\Delta$.

Proof. Part (a) of the proposition follows immediately since $V_{\beta 0}$ is a submatrix of V_β. To deduce part (b), we will use the following bound (implied by (8.2) and (8.8)):

$$\| \tilde{W} \| < 1.01 \ w, \tag{8.9}$$

as well as the next proposition.

PROPOSITION 8.3. *For any* 4*-tuple of* $k \times k$ *matrices,* X, $\tilde{X}$, Y, *and* $\tilde{Y}$, *we have*

(a) $\| \Delta(X \pm Y) \| \le \| \Delta(X) \| + \| \Delta(Y) \|$,

(b) $\| \Delta(XY) \| \le \| \Delta(X) \| \ \| Y \| + \| \Delta(Y) \| \ \| X \| + \| \Delta(X) \| \ \| \Delta(Y) \|$,

and if X *and* $\tilde{X}$ *are nonsingular matrices, then also*

(c) $\| \Delta(X^{-1}) \| \le \| X^{-1} \| \ \| \tilde{X}^{-1} \| \ \| \Delta(X) \|$.

Proof. The parts (a)–(c) follow immediatley from the next simple equations:

(a) $\Delta(X \pm Y) = \Delta(X) \pm \Delta(Y)$,

(b) $\Delta(XY) = \Delta(X)Y + X\Delta(Y) + \Delta(X)\Delta(Y)$,

(c) $\Delta(X^{-1}) = X^{-1}\Delta(X)\tilde{X}^{-1} = -\tilde{X}^{-1}\Delta(X)X^{-1}$. ∎

Since $S = G - EB^{-1}C$ under (2.2), we will next recursively extend the bound $||\Delta W||$ on the error norms of G, E, B, and C to yield some bounds on the error norms of B^{-1}, EB^{-1}, $EB^{-1}C$, and S.

We first apply part (c) of the latter proposition for $X = B$ and $\Delta(X^{-1}) = \tilde{B}^{-1} - B^{-1}$ and obtain that

$$\| \Delta(B^{-1}) \| \le \| B^{-1} \| \ \| \tilde{B}^{-1} \| \ \| \Delta(B) \| .$$

Substitute (8.3) and (8.7) into the latter bound and obtain that

$$\| \Delta(B^{-1}) \| < 1.221 \ \| \Delta(W) \| / m^2.$$

Combine the relations (8.6) and (8.7) to obtain that $||L - \tilde{B}^{-1}|| \le 1.11\nu/m \le (1.11)\Delta/(4000m^3)$. Combine the latter bounds on the norms, recall (8.8), and deduce that

$$\| L - B^{-1} \| \le \| \Delta(B^{-1}) \| + \| L - \tilde{B}^{-1} \| < (1.221 + (1.11)/(4000m))\Delta/m^2$$

$$< 1.3\Delta/m^2 \ . \tag{8.10}$$

Apply part (b) of Proposition 8.3 for $X = E$, $Y = L$, and deduce that

$$\| \Delta(EL) \| \leq \| \Delta(E) \| \| L \| + \| L - B^{-1} \| (\| E \| + \| \Delta(E) \|) .$$

Recall from (7.2) and (7.3) that $w/m \leq 1.02$. By combining the two latter inequalities with our bounds on $\| L \|$, $\| L - B^{-1} \|$, $\| E \|$, and $\| \Delta(E) \|$ (see (8.4)–(8.10)), obtain that

$$\| \Delta(EL) \| \leq \left(\frac{1.11}{m}(1+\nu) + \left(\frac{1.3}{m^2} \right) 1.01w \right) \Delta \leq \frac{2.5}{m} \Delta . \tag{8.11}$$

Then again, we apply part (b) of Proposition 8.3, this time for $X = EL$, $Y = C$, and obtain that

$$\| \Delta(ELC) \| \leq \| \Delta(EL) \| \| \tilde{C} \| + \| \Delta(C) \| \| EL \| .$$

Substitute our previous estimates (8.2), (8.4)–(8.9), and (8.11) into the latter inequality and deduce that

$$\| \Delta(ELC) \| \leq \left(\left(\frac{2.5}{m} \right) 1.01w + \frac{1.11}{m}(1+\nu)w \right) \Delta \leq 3.7 \Delta w/m.$$

Now, since $w/m \leq 1.02$, we have

$$\| \Delta(ELC) \| \leq 4\Delta.$$

We obtain $\| \Delta(G) \| \leq \Delta$ from (8.5) and (8.8). By applying part (a) of Proposition 8.3 for $X = G$, $Y = ELC$, we deduce that

$$\| \Delta(S) \| \leq \| \Delta(ELC) \| + \| \Delta(G) \| \leq 5\Delta,$$

which proves Proposition 8.2. □

Now, we observe that the assumptions of Proposition 8.2 are satisfied for $W = V$, $\tilde{W} = V$, and $\Delta = 40\nu m$, and we extend them to $W = V_\alpha$ for all α. The extension from $W = V_\beta$ to $W = V_{\beta 0}$ for any β is trivial. We will comment on the extension to $W = V_1$, which will be our sample for the extension from $W = V_\beta$ to $W = V_{\beta 1}$ for any β. We write $\tilde{B} = B = V_0$, $L = -X_{1,g}$, $X_{1,0} = -I/m$, $g \geq 4$, and define by (6.5) the matrices $X_{1,i}$ for all i. Now, observe that $||I + X_{1,0}W|| = ||F||/m$, $||I + X_{1,0}\tilde{W}|| \leq ||F||/m + \Delta/m \leq (\Delta + (p-1)n)/m < 1/(10pn) < 1/m^{1/2}$.

Therefore, (5.6) implies that

$$\| X_i + B^{-1} \| \leq \| B^{-1} \| / m^{2^{i-1}}, \quad i = 1, \ldots, g,$$

and, consequently, since $L = -X_g$ for $g \geq 4$, we have

$$\| L - B^{-1} \| \leq \nu \| B^{-1} \| \quad \text{for} \quad \nu \leq 1/m^8 < 1/(4000m^2n^{12}),$$

thus satisfying (8.6). The remaining assumption (8.7) of Proposition 8.2 is also easily verified (by using (8.5) and (8.8) and by following the line of the proof of Corollary 7.3).

Now, we are ready to extend Proposition 8.2 recursively, which will give us the desired upper bound on $||\Delta(V_\alpha)||$ in terms of b. By applying this proposition recursively, we extend its assumptions to $W = V_0$, $W = V_1$, $\Delta = 40\nu m$. (In the subsequent

recursive extension from $W = V_{\beta 1}$ to $W = V_\beta$ for any binary string β of length at most $h-1$, we will choose ν depending on α but satisfying (8.6) for all α.)

Apply the bounds of parts (a) and (b) of Proposition 8.2 recursively and obtain that

$$\| \Delta(W) \| \leq \Delta \sum_{i=0}^{|\alpha|} 5^i < n^3 \Delta$$

for $W = V_\alpha$ and all α (with $|\alpha| \leq h = \log n$). Let us choose a b that enables us to reconcile the initial choice of $\Delta = 40\nu m$ and the latter bound on $||\Delta(W)||$ with (8.6)–(8.8). Recall Remark 8.1, recall that the choice of g according to (6.4) implies the bound (6.7) on the output approximation to the inverse, substitute $\nu = 2^{-b}$, and obtain the desired estimate:

$$\| \Delta(V_\alpha) \| < (40mn^3\Delta)2^{-b} < 2^{2-b}w^2 n \tag{8.12}$$

for all α.

Let us choose b of order $n \log p$ satisfying the bound

$$b \geq 3 + \log n + (n+2)\log w,$$

which is compatible with the choice of $b = \log(1/\nu)$ and with (8.6). Substitute this bound on b into the preceding upper bound on $\| \Delta(V_\alpha) \|$ and obtain that

$$\| \Delta(V_\alpha) \| < 0.5\ w^{-n}$$

for all α. This satisfies the requirement (7.4) of Proposition 7.5 and completes the proof of Proposition 8.1.

REMARK 8.2. *By Corollary* 7.4, $c(\beta) = O(\log n)$ *for all binary strings* β *of length at most* $h-1$. *Furthermore,* (6.4) *and the above choice of* b *are compatible with the choice of* $g = g(\beta)$ *of order* $\log b = \log(n \log p)$ *for all* β.

9. Computations with rounding-off: Estimates for the finite precision and computational cost. So far, we assumed the infinite precision of computing the RD and IRD by means of Algorithms 6.1 and 6.2. Next, we will show that this is not necessary for obtaining the result of Proposition 8.1; that is, we will prove the following.

PROPOSITION 9.1. *The estimates of Proposition* 8.1 *hold even if the computations by Algorithms* 6.1 *and* 6.2 *are performed with a precision of* $\tilde{b}$ *bits, for some* $\tilde{b} = O(n \log\ p)$ *provided that a single extra Newton's step* (6.5) *is performed in each application of Algorithm* 6.1.

Proof. Let us first assume the computations of Newton's step (6.5) with the infinite precision, but in the transition from $W = V_\alpha$ to $S = V_{\alpha 1}$ for all binary strings α, $|\alpha| < h$, let the computations be performed with rounding to the $\tilde{b}$-bit precision. Assuming B^{-1} available, the latter transition involves two multiplications and a subtraction of $k \times k$ matrices for $k = 2^{h-|\alpha|}$ (compare (2.2)). By applying the techniques of backward error analysis [W65], [BL80], we bound the norm of the matrix $\epsilon(S)$ of the errors of the approximation to S caused by rounding:

$$\| \epsilon(S) \| \leq n^{O(1)} 2^{-\tilde{b}} \| W \| (1 + \| L \| \| W \|)$$

for L denoting the computed approximation to B^{-1} (cf. (8.6), (8.7)). By applying the relations (8.2), (8.7), (7.2), (7.3), and Proposition 7.1, we obtain that

$$\| \epsilon(S) \| \leq m^{O(1)} 2^{-\tilde{b}} .$$

By choosing $\tilde{b}$ of order $n \log p$, we make $\| \epsilon(S) \|$ less than $2 \| \Delta(W) \|$ for $W = V_\alpha$ and for all α. This is less than 40% of the upper bound that we have in part (b) of Proposition 8.2. Combining both of these bounds gives us cumulative upper bound $7 \| \Delta(W) \|$, which shows the overall impact of the above rounding errors. This enables us to preserve the validity of the bound (8.12) (since $\sum_{i=0}^{h} 7^i \leq n^3$ for $h = \log n$) and, consequently, of the entire proof of Proposition 8.1.

It remains to estimate the impact of rounding to $\tilde{b}$-bit precision when we perform Newton's steps (6.5). Then again, we deal with two matrix multiplications (we ignore the errors caused by the simple addition step $2I + (WX_{i-1})$). By applying backward error analysis again, we estimate that

$$\| \epsilon(X_i) \| \leq n^{O(1)} 2^{-\tilde{b}} \| X_{i-1} \| (2 + \| W \| \| X_{i-1} \|) , \tag{9.1}$$

where $\epsilon(X_i)$ denotes the matrix of the errors of approximation of X_i due to rounding in performing iteration (6.5).

Our next goal is to prove the bound

$$\| X_{i-1} \| \leq 1.1(1 + 1/\tilde{m})/m < 1.21/m \quad \text{for } i \geq 1, \tag{9.2}$$

ignoring for simplicity the terms of order $2^{-2\tilde{b}}$ or less.

We have from Corollary 7.4 that $\|I + BX_0\| < \frac{1}{\tilde{m}}$ for $\tilde{m} = 10pn \geq 20p$ and for $X_0 = -I/m$. Then we obtain from (5.6) that

$$\| I + BX_{i-1} \| \leq 1/\tilde{m}^{2^{i-1}}.$$

Therefore,

$$\| X_{i-1} + B^{-1} \| \leq \| B^{-1} \| / \tilde{m}^{2^{i-1}} .$$

Consequently,

$$\| X_{i-1} \| \leq \| B^{-1} \| \left(1 + \frac{1}{\tilde{m}^{2^{i-1}}}\right)$$

for $i = 1, 2, \ldots$. Substitute (8.3) and arrive at inequality (9.2).

Substitute bound (9.2) on $\| X_{i-1} \|$ and the bound $\| W \| \leq w$ of (8.2) into (9.1), recall (7.2) and (7.3), and obtain that

$$\| \epsilon(X_i) \| \leq (np)^{O(1)} 2^{-\tilde{b}} \quad \text{for all } i .$$

By choosing a sufficiently large $\tilde{b}$, though of order $n \log\ p$, we easily ensure that

$$\| \epsilon(X_i) \| < 2^{-b} / \| B \| .$$

Therefore,

$$\| I + B(X_i + \epsilon(X_i)) \| \leq \| I + BX_i \| + \| B \| \, \| \epsilon(X_i) \|$$

$$\leq \| I + BX_{i-1} \|^2 + 2^{-b} .$$

Since Newton's iteration (6.5) stops if $\| I + BX_{i-1} \| \leq 2^{-b}$, we may assume that $\| I + BX_{i-1} \| > 2^{-b}$, so that the rounding may at worst change (6.7) into the bound

$$\| I + B(X_i + \epsilon(X_i)) \| \leq 2 \| I + BX_i \| \leq 2^{\sigma} \| I + BX_0 \|^{2^i} < 2(2 \| I + BX_0 \|)^{2^i},$$

where $\sigma = \sum_{s=0}^{i} 2^s < 2^{i+1}$. Since $\| I + BX_0 \| \leq 1/m$, the impact of the rounding on the residual norm of the output approximation computed by Newton's iteration is more than compensated by a single extra step (5.5), (6.5), and this completes the proof of Proposition 9.1. □

Let us next summarize our current complexity extimates for the computation of the IRD before we improve them slightly in the next section. Choose b and $\tilde{b}$ of order $n \log p$ and choose g of order $\log(n \log p)$, which is consistent with (6.4) under (5.7) or (5.8). Now, by combining the results of Corollaries 4.6 and 7.3 and Propositions 8.1 and 9.1 with Remark 8.2 and the estimates for the arithmetic parallel complexity of performing Algorithms 6.1 and 6.2 and by using the B-principle, obtain the following corollary.

COROLLARY 9.2. *Algorithm* 6.2 *computes the IRD of a matrix* V *satisfying* (7.1) *and* (7.2) *at the cost* $O_A((\log n)^2 \log(n \log p), n^{\omega}/\log n)$ *for* ω *of* (4.5). *Furthermore,* $\tilde{b}$*-bit precision suffices in these computations for some* $\tilde{b}$ *of order* $n \log p$.

10. Pipelined computation of the IRD. Our next goal is a modification of Algorithm 7.1, which, as we claimed in section 1.6, will enable us to improve by factor $\log n$ the asymptotic time-complexity bounds of Corollary 9.2, without increasing the processor bound by more than a constant factor. To achieve this goal, we will incorporate into our construction the techniques of *pipelining* along the lines of [PR91] (where such techniques were called *stream contraction* and applied to computing the RD of a matrix over semirings of a certain class). Here are our informal underlying observations.

Algorithm 6.1 is not fully efficient because it spends substantial time and work on refining the approximations to the inverses of the matrices $\tilde{B}_{\beta} = \tilde{V}_{\beta 0}$ (cf. (6.5)); this delays the subsequent use of such approximations in the inversion of the matrices $\tilde{V}_{\beta 10} = \tilde{B}_{\beta 1}$, which anyway starts with a much cruder approximation $-I/m$. Next, we will modify Algorithm 6.1. We will start the Newton process of the inversion of $\tilde{B}_{\beta 1}$ by relying on the available rough approximations to $\tilde{B}_{\beta}^{-1}$, and then we will recursively produce a stream of better approximations when the process progresses.

In other words, we are going to pipeline the recursion on α (decomposition) and the Newton one (inversion). To approximate the matrix $V_{\beta 1}$ of the RD and ERD, we will start using the intermediate approximations to the inverse $V_{\beta 0}^{-1}$, as soon as they are computed by Newton's iteration. We will update the resulting approximation to $V_{\beta 1}$ as soon as the approximation to $V_{\beta 0}^{-1}$ is refined; that is, in the process of the computation of the matrix $\tilde{V}_{\beta 1}$, we will keep refining every step of the computations as soon as we refine its input.

More precisely, we will initialize this process by fixing some natural g, to be specified later on. As before, α and β will denote binary strings, $|\alpha| \leq h$, $|\beta| < h$, and γ will denote the unary strings consisting of zero bits. $u(\alpha)$ will denote the number of bits one in a binary string α. t will denote integers in the range from t_0 to $g + h$.

Here, $t_0 = u(\alpha)$ in (10.1) and (10.3) (where α is fixed), $t_0 = u(\beta) + 1$ in (10.4)–(10.7) (where β is fixed), and $t_0 = 0$ elsewhere, that is, in (10.2).

We will now define the following matrices whose subscripts α, β, γ, and t range as specifed above:

$$V_{\alpha,t} = \begin{pmatrix} B_{\alpha,t} & C_{\alpha,t} \\ E_{\alpha,t} & G_{\alpha,t} \end{pmatrix}, \tag{10.1}$$

$$V_{\gamma,t} = V_\gamma \tag{10.2}$$

(cf. Definition 2.6),

$$V_{\alpha\gamma,t} = (V_{\alpha,t})^{(q)} \text{ for } |\alpha\gamma| \le h,\ q = 2^{h-|\alpha\gamma|}, \tag{10.3}$$

$$X_{\beta,t,0} = \begin{cases} -I/m \text{ for } t = u(\beta) + 1, \\ -X_{\beta,t-1} \text{ for } t > u(\beta) + 1, \end{cases} \tag{10.4}$$

$$X_{\beta,t,i+1} = X_{\beta,t,i}(2I + V_{\beta 0,t} X_{\beta,t,i}),\ i = 0, 1, 2, 3, 4, \tag{10.5}$$

$$X_{\beta,t} = -X_{\beta,t,4}, \tag{10.6}$$

$$V_{\beta 1,t+1} = G_{\beta,t} - E_{\beta,t} X_{\beta,t} C_{\beta,t}. \tag{10.7}$$

Now, we are ready to specify our pipelined algorithm.

Algorithm 10.1. *Stream contraction for approximating the RD.*

Input: natural g, h, and $n = 2^h$; an $n \times n$ matrix V.

Output: for all binary strings α of length at most h, matrices $V_{\alpha,g+u(\alpha)}$ satisfying the equations (10.1)–(10.7) and approximating the matrices V_α, respectively.

Computations:

Stage 0. Apply (10.2) for $t = 0$ to define the matrices $V_{\gamma,0}$, $|\gamma| = 0, \dots, h$.

Stage t, $\mathbf{t = 1, \dots, g + h}$.

Concurrently in all binary strings β of length less than h with $u(\beta) < t$, compute successively:

(a) $X_{\beta,t,0}$, based on (10.4),

(b) $X_{\beta,t,i+1}$ for $i = 0, 1, 2, 3, 4$, based on (10.5),

(c) $X_{\beta,t}$, by (10.6),

(d) $V_{\beta 1,t+1} = G_{\beta,t} - E_{\beta,t} X_{\beta,t} C_{\beta,t}$, based on (10.1) and (10.7),

(e) $V_{\beta 1\gamma,t+1}$, by (10.3) where $\alpha = \beta 1$.

These rules are complemented by the following.

Stopping criterion: Output the matrices $V_{\alpha,g+u(\alpha)}$ for all binary strings α of length at most h and cancel all the subsequent computations involving these matrices.

For the reader's convenience, we will next list the matrices computed at stages 1, 2, and 3, letting $\gamma_0, \gamma_1, \gamma_2, \gamma_3$ denote unary strings filled with zeros.

Stage 1:

$X_{\gamma_0,1,0} = -I/m$,

$X_{\gamma_0,1,i+1} = X_{\gamma_0,1,i}(2I + V_{\gamma_0 0} X_{\gamma_0,1,i})$, $i = 0, 1, 2, 3, 4$,
$X_{\gamma_0,1} = -X_{\gamma_0,1,4}$,
$V_{\gamma_0 1,2} = G_{\gamma_0} - E_{\gamma_0} X_{\gamma_0,1,} C_{\gamma_0}$ for $G_{\gamma_0}, E_{\gamma_0}, C_{\gamma_0}$ of (3.1), $|\gamma_0| < h$,
$V_{\gamma_0 1\gamma_1,2} = V^{(q)}_{\gamma_0 1,2}$ for $q = 2^{h-1-|\gamma_0\gamma_1|}, |\gamma_0\gamma_1| < h$.
Stage 2: $X_{\gamma_0,2,0} = -X_{\gamma_0,1}$,
$X_{\gamma_0,2,i+1} = X_{\gamma_0,2,i}(2I + V_{\gamma_0 0,2} X_{\gamma_0,2,i})$, $i = 0, 1, 2, 3, 4$,
$X_{\gamma_0,2} = -X_{\gamma_0,2,5}$,
$V_{\gamma_0 1,3} = G_{\gamma_0} - E_{\gamma_0} X_{\gamma_0,2} C_{\gamma_0}$ for $|\gamma_0| < h$,
$V_{\gamma_0 1\gamma_1,3} = V^{(q)}_{\gamma_0 1,3}$ for $q = 2^{h-1-|\gamma_0\gamma_1|}$, $|\gamma_0\gamma_1| < h$,
$X_{\gamma_0 1\gamma_1,2,0} = -I/m$,
$X_{\gamma_0 1\gamma_1,2,i+1} = X_{\gamma_0 1\gamma_1,2,i}(2I + V_{\gamma_0 1\gamma_1 0,2} X_{\gamma_0 1\gamma_1,2,i})$, $i = 0, 1, 2, 3, 4$,
$X_{\gamma_0 1\gamma_1,2} = -X_{\gamma_0 1\gamma_1,2,5}$,
$V_{\gamma_0 1\gamma_1 1,3} = G_{\gamma_0 1\gamma_1,2} - E_{\gamma_0 1\gamma_1,2} X_{\gamma_0 1\gamma_1,2} C_{\gamma_0 1\gamma_1,2}$ for $G_{\gamma_0 1\gamma_1,2}, E_{\gamma_0 1\gamma_1,2}, C_{\gamma_0 1\gamma_1,2}$ of (10.1), with $\alpha = \gamma_0 1\gamma_1, t = 2, |\gamma_0\gamma_1| < h - 1$,
$V_{\gamma_0 1\gamma_1 1\gamma_2,3} = V^{(q)}_{\gamma_0 1\gamma_1 1,3}$ for $q = 2^{h-2-|\gamma_0\gamma_1\gamma_2|}, |\gamma_0\gamma_1\gamma_2| < h - 1$.
Stage 3: $X_{\gamma_0,3,0} = -X_{\gamma_0,2}$,
$X_{\gamma_0,3,i+1} = X_{\gamma_0,3,i}(2I + V_{\gamma_0 0} X_{\gamma_0,3,i})$, $i = 0, 1, 2, 3, 4$,
$X_{\gamma_0,3} = -X_{\gamma_0,3,5}$,
$V_{\gamma_0 1,4} = G_{\gamma_0} - E_{\gamma_0} X_{\gamma_0,3} C_{\gamma_0}$ for $|\gamma_0| < h$,
$V_{\gamma_0 1\gamma_1,4} = V^{(q)}_{\gamma_0 1,4}$ for $q = 2^{h-1-|\gamma_0\gamma_1|}, |\gamma_0\gamma_1| < h$,
$X_{\gamma_0 1\gamma_1,3,0} = -X_{\gamma_0 1\gamma_1,2}$,
$X_{\gamma_0 1\gamma_1,3,i+1} = X_{\gamma_0 1\gamma_1,3,i}(2I + V_{\gamma_0 1\gamma_1 0,3} X_{\gamma_0 1\gamma_1,3,i})$, $i = 0, 1, 2, 3, 4$,
$X_{\gamma_0 1\gamma_1,3} = -X_{\gamma_0 1\gamma_1,3,5}$,
$V_{\gamma_0 1\gamma_1 1,4} = G_{\gamma_0 1\gamma_1,3} - E_{\gamma_0 1\gamma_1,3} X_{\gamma_0 1\gamma_1,3} C_{\gamma_0 1\gamma_1,3}$ for $G_{\gamma_0 1\gamma_1,3}, E_{\gamma_0 1\gamma_1,3}, C_{\gamma_0 1\gamma_1,3}$ of (10.1) with $\alpha = \gamma_0 1\gamma_1, t = 3, |\gamma_0\gamma_1| < h - 1$,
$V_{\gamma_0 1\gamma_1 1\gamma_2,4} = V^{(q)}_{\gamma_0 1\gamma_1 1,4}$ for $q = 2^{h-2-|\gamma_0\gamma_1\gamma_2|}, |\gamma_0\gamma_1\gamma_2| < h - 1$,
$X_{\gamma_0 1\gamma_1 1\gamma_2,3,0} = -I/m$,
$X_{\gamma_0 1\gamma_1 1\gamma_2,3,i+1} = X_{\gamma_0 1\gamma_1 1\gamma_2,3,i}(2I + V_{\gamma_0 1\gamma_1 1\gamma_2 0,3} X_{\gamma_0 1\gamma_1 1\gamma_2,3,i})$, $i = 0, 1, 2, 3, 4$,
$X_{\gamma_0 1\gamma_1 1\gamma_2,3} = -X_{\gamma_0 1\gamma_1 1\gamma_2,3,5}$,
$V_{\gamma_0 1\gamma_1 1\gamma_2 1,4} = G_{\gamma_0 1\gamma_1 1\gamma_2,3} - E_{\gamma_0 1\gamma_1 1\gamma_2,3} X_{\gamma_0 1\gamma_1 1\gamma_2,3} C_{\gamma_0 1\gamma_1 1\gamma_2,3}$ for $G_{\gamma_0 1\gamma_1 1\gamma_2,3}, E_{\gamma_0 1\gamma_1 1\gamma_2,3}, C_{\gamma_0 1\gamma_1 1\gamma_2,3}$ of (10.1), with $\alpha = \gamma_0 1\gamma_1 1\gamma_2$, $t = 3$, $|\gamma_0\gamma_1\gamma_2| < h - 2$,
$V_{\gamma_0 1\gamma_1 1\gamma_2 1\gamma_3,4} = V^{(q)}_{\gamma_0 1\gamma_1 1\gamma_2 1,3}, q = 2^{h-3-|\gamma_0\gamma_1\gamma_2\gamma_3|}, |\gamma_0\gamma_1\gamma_2\gamma_3| < h - 2$.

Correctness of Algorithm 10.1 is immediatley verified. It remains to specify the choice of natural g, which would satisfy the requirements of Proposition 7.5, and then to estimate the resulting *compuational cost*.

As in section 8, we assume infinite precision computations in Algorithm 10.1, but the same techniques of backward error analysis as in section 9 enable relatively simple transition to the case of computations with rounding to finite precision of order $n \log\ p$ bits.

The analysis of the approximation errors and of the computation precision given in section 8 is easily extended. In particular, the extension of Proposition 8.2 and its proof is immediate provided that in its statement $V_\beta, V_{\beta 0}$, and $V_{\beta 1}$ are replaced by $V_{\beta,t}$, $V_{\beta 0,t+1}$, and $V_{\beta 1,t+1}$, $t > u(\beta)$. Furthermore, the assumptions of this proposition are extended recursively with each increase of t and the length $|\beta|$ by 1. Such an extension is analyzed as in section 9. (The transition from $V_{\beta,t}$ to $V_{\beta 1,t+1}$ involves five Newton steps (10.5), versus four steps used in section 8; an extra step compensates us for the impact of the rounding errors; this suffices according to the analysis

in section 9.) The small factor 5 of the error propagation bound of part (b) of Propostion 8.2 (even when it increases to 7 due to the rounding errors) is immediately suppressed by Newton's steps (10.5). To accomodate the factors 5 or 7, we also should increase the upper bound on $||V_{\beta 0,t}^{-1}X_{\beta,t,0}+I||$ obtained in Corollary 7.4; the increase is from $1/(10n^2)$ to $1/(2n^2)$ or to $7/(10n^2)$, respectively. This, however, implies only a nominal increase of g, which we may set equal to

$$g = 1 + \lceil \log(b/(2\log n))\rceil, \tag{10.8}$$

say. ($2\log n$ in the denominator replaces $\log(10n^2)$, which more than compensates us for the error propagation factors 5 or 7.) The computation of every $V_{\alpha,g+u(\alpha)}$ involves at least $5g$ Newton steps (10.5), so that the output error norm bound 2^{-b} is guaranteed under (10.8). Therefore, to satisfy the requirement (7.4) of Proposition 7.5, it is sufficient to choose b of order $n\log p$. Then, by (10.8), we have

$$g = O(\log(n\log p)). \tag{10.9}$$

Now, let us estimate the *computational cost* of performing Algorithm 10.1.

For any t, Stage t amounts essentially to ten steps of multiplication of at most n/k pairs of $k\times k$ matrices for $k=2^\ell$, $\ell=1,2,\dots,h$. All these multiplications for $k=2^\ell$ and for all ℓ are performed concurrently. Their overall cost is bounded by $O_A(\log n, n^\omega)$, $2\le\omega<2.376$ (compare (4.5) and observe that $\sum_{\ell=1}^h 2^\ell(n/2^\ell)^\omega = O(n^\omega)$ for $\omega>1$). Summarizing these bounds for all stages t, $t=1,\dots,g+h$, we obtain the following proposition.

PROPOSITION 10.1. *Algorithm* 10.1 *supports approximating the RD of a c.-d.d.* $n\times n$ *matrix* V *of* (7.1) *within the error norm bound* 2^{-b}*, at the overall cost* $O_A((\log n)(\log n+g), n^\omega)$ *for* g *of* (10.8) *and* ω *of* (4.5).

By combining Algorithms 10.1 and 6.2, summarizing the estimates for the computational cost of their performance, given in particular in (10.9) and Proposition 10.1, and extending the rounding error analysis applied in the proof of Proposition 9.1, we obtain the following corollary.

COROLLARY 10.2. *The IRD of a c.-d.d.* $n\times n$ *matrix* V *of* (7.1) *can be exactly computed at the computational cost* $O_A((\log n)\log(n\log p), n^\omega)$ *for* ω *of* (4.5), $2\le\omega<2.376$*; moreover, this computation can be performed by only involving operations with* $\tilde b$*-bit precision numbers for* $\tilde b = O(n\log\ p)$.

11. Computing modulo a fixed prime of the ERD of an integer matrix. Our next goal is probabilistic extension of Corollary 10.2 from the class of matrices V of (7.1) to the class of all strongly nonsingular integer matrices A. In this section, we will compute the IRD and even the ERD of A modulo a fixed prime p; in the next section we will shift to the IRD of A.

Let $A=(a_{i,j})$ be a strongly nonsingular $n\times n$ matrix filled with integers $a_{i,j}$. Then by virtue of Proposition 3.2, there exists the RD of A. Let p be a fixed prime, let $0\le f_{i,j}=a_{i,j} \bmod p<p$ for all i,j, and let

$$F=(f_{i,j})=A \bmod p\ . \tag{11.1}$$

(Here and hereafter, we assume that $0\le a \bmod p<p$ for any integer a.)

We will compute modulo p the ERD of the matrix F as an auxiliary stage of computing the ERD of A. At first, we should examine if there exists the ERD modulo p of F.

LEMMA 11.1 (see [IR82]). *Let $f(n)$ be a function defined on the set of positive integers such that $f(n) > 0$ and $\lim_{n\to\infty} f(n) = \infty$. Then there exist two positive constants C and n_0 such that, for any $n > n_0$, the interval*

$$J = \{p : f(n)/n < p < f(n)\} \tag{11.2}$$

contains at least $f(n)/(C \log f(n))$ distinct primes.

LEMMA 11.2. *Let $f(n)$, $h_q(n)$, and $k_q(n), q = 1, \dots, Q$ be some functions in n such that $h_q(n)$ are integer valued, $h_q(n) \neq 0$,*

$$0 < (h_q(n))^{1/k_q(n)} \le f(n)/n,\ k_q(n) > 0,\ \lim_{n\to\infty} f(n) = \infty \tag{11.3}$$

for $q = 1, \dots, Q$. Let p be a random prime in the interval J of (11.2). *Then for the positive constants C and n_0 of Lemma* 11.1 *and for any fixed $n > n_0$, we have $h_q(n) \neq 0 \bmod p$ for $q = 1, \dots, Q$ with a probability at least $1-(CK(n)\log f(n))/f(n)$, where $K(n) = \sum_{q=1}^{Q} k_q(n)$.*

Proof. Let $l_q(n)$ primes lying in the interval J divide $h_q(n)$. Then their product also divides $h_q(n)$ and, therefore, cannot exceed $h_q(n)$. As these primes lie in the interval J, each of them exceeds $f(n)/n$, and their product exceeds $(f(n)/n)^{l_q(n)}$. Hence, $(f(n)/n)^{l_q(n)} < h_q(n)$. Compare this inequality with the assumed bound $h_q(n) \le (f(n)/n)^{k_q(n)}$ and obtain that $l_q(n) < k_q(n)$. This holds for all q. Therefore, the number of primes lying in J and dividing at least one of the integers $h_q(n)$ (for any q) is at most $\sum_{q=1}^{Q} l_q(n) < \sum_{q=1}^{Q} k(n) = K(n)$. Compare this number with the overall number of primes in J estimated in Lemma 11.1 and obtain the desired probability estimate. ∎

PROPOSITION 11.3. *Let $\rho > 2$ be a fixed scalar, let A be a strongly nonsingular $n \times n$ integer matrix, where $n > 1$, $\|A\| > 1$, and let p be a prime chosen randomly (under the uniform probability distribution) in the interval $J = \{p :\ n^{\rho-1} \log \| A \| < p < n^{\rho} \log \| A \|\}$. Then $p \ge n$, and the matrix F of* (11.1) *is strongly nonsingular modulo p with a probability at least $1-P_{\rho,n}$ for $P_{\rho,n} < (n+1)Cn^{1-\rho}$ and for a positive constant C of Lemmas* 11.1 *and* 11.2.

Proof. Apply Lemma 11.2 for $f(n) = n^{\rho}\log||A||$, $h_q(n) = |\det A^{(q)}|$, $Q = n$, and

$$k_q(n) = (q \log\ ||A||)/\log\ (n^{\rho-1} \log\ ||A||),$$

$q = 1, \dots, n$. Recall from Proposition 2.4 that $|\det A^{(q)}| \le ||A^{(q)}|| \le ||A||^q$ for all q, $q \le n$, and deduce that (11.3) holds for all $q \le n$. We immediately deduce that $K(n) = \sum_{q=1}^{n} k_q(n) = ((n+1)n \log||A||)/(2\log(n^{\rho-1}\log||A||))$ and $(\log f(n))/f(n) = (\log(n^{\rho}\log||A||))/(n^{\rho}\log||A||)$. Substitute these expressions for $K(n)$ and $(\log f(n))/f(n)$ into Lemma 11.2 and obtain that $(\det A^{(q)}) \bmod p \neq 0$ for $q = 1, \dots, n$ with a probability at least $1 - P_{\rho,n}$, where

$$P_{\rho,n} < \frac{(n+1)nC\log\ (n^{\rho}\log\ ||A||)}{2n^{\rho}\log\ (n^{\rho-1}\log\ ||A||)} = \frac{(n+1)C}{2n^{\rho-1}}\left(1 + \frac{\log\ n}{\log\ (n^{\rho-1}\log\ ||A||)}\right)$$

for all k. By assumption, we have $||A|| \ge 2$, $\rho > 2$, $n \ge 2$, and it follows that $\log(n^{\rho-1}\log||A||) > \log\ n$. Combine this bound with the above bound on $P_{\rho,n}$ and obtain the claimed estimate of Proposition 11.3. ∎

Now, we will assume that a prime p has been chosen in the interval J of Proposition 11.3 and the matrix F of (11.1) is strongly nonsingular modulo p and, therefore, possesses its ERD modulo p.

The next algorithm computes modulo p such an ERD, representing each auxiliary or output rational value as a pair of its numerator and denominator given as two integers reduced modulo p. (This enables us to avoid the costly stage of computing integer reciprocals modulo p.) The RD modulo p of A is computed already at Stage 1 of the algorithm. Subsequent stages yield the extending set of the RD modulo p via the computation of the dual RD modulo p (see the definitions of the extending set and the dual RD in section 3).

Algorithm 11.1. *Computing the ERD modulo a fixed prime.*

Input: a prime p and a pair of strongly nonsingular $n \times n$ matrices A and $F = A \mod p$ filled with integers.

Output: the (common) ERD modulo p of A and F.

Computations:

Stage 0. Compute $m = 10(np)^2$ and the c.-d.d. matrix $V = F - mI$ (cf. (7.1), (7.2)).

Stage 1. Compute modulo p the IRD of V by applying Algorithms 10.1 and 6.2. Then, compute modulo p the RD of V, by dividing modulo p all the computed matrices $m_\alpha V_\alpha$ of the IRD by the computed multipliers m_α for all α; represent the result of each division by a pair of an entry of $m_\alpha V_\alpha$ reduced modulo p and $m_\alpha \mod p$. Output the computed RD modulo p of V, which is also the RD modulo p of $F = V \mod p$.

Stage 2. Recall Proposition 4.5 and compute $\det V$.

Stage 3. Recall from Proposition 2.4 that $|\det V| \le \| V \|^n$ and apply Newton's iteration (5.5) for $B = V$ in order to compute an approximation X to V^{-1} satisfying (5.3) for $B = V$ and for b satisfying

$$2^{-b}/m < \| V \|^{-n} /2.2 \ . \tag{11.4}$$

Then, compute the entries of the matrix $X \det V$ and round them to the closest integers, which gives us $\text{adj } V$.

Stage 4. Compute the matrix $\hat{W} = (\text{adj } V) \mod p - mI$. Apply Algorithms 10.1 and 6.2 to compute modulo p the IRD of $\hat{W}$ (we will prove that this is the dual IRD modulo p of A, V, and F). Then compute modulo p the matrices $\hat{W}_{\beta 0}$ of the RD of $\hat{W}$ for all binary strings β of length less than h. Output this set of matrices, to be denoted $\{(\hat{W}_{\beta 0}/\det V) \mod p\}$. Their entries are the pairs of integers, each reduced modulo p; one integer of each pair is an entry of $\hat{W}_{\beta 0} \mod p$ and another is $(\det V) \mod p$. (This set of matrices defines the extending set $\{V_{\beta 0}^{-1} \mod p\}$ of the RD modulo p of the input matrix F.)

To verify *correctness* of Algorithm 11.1, first extend Corollary 7.3 to obtain that $\| V^{-1} \| \le 1.1/m$. Together with (11.4), this implies the bound

$$\| X \det V - \text{adj } V \| < 1/2$$

for the matrix X computed at Stage 3 of Algorithm 11.1. Therefore, the rounding at this stage correctly defines $\text{adj } V$.

Furthermore, the matrices $\hat{W}_\alpha \mod p$ (see Stage 4) represent the RD modulo p of $\text{adj } V$. Therefore, the set $\{(\hat{W}_\alpha/\det V) \mod p\}$ represents the RD modulo p of V^{-1}. To complete the correctness proof, it remains to observe that the set of matrices $\{(\hat{W}_{\beta 0}/\det V) \mod p, |\beta| < h\}$ is nothing else but the extending set $\{B_{\beta 0}^{-1} \mod p, |\beta| < h\}$ of the (common) RD modulo p of the three matrices A, V, and $F = V \mod p = A \mod p$. This follows from the next simple result.

PROPOSITION 11.4. *Let* $\{V_\alpha\}$ *and* $\{W_\alpha\}$ *denote the RD and the dual RD of a pair of* $n \times n$ *matrices* V *and* $W = V^{-1}$*, respectively. Then,* $V_\alpha^{-1} = W_\alpha$ *for all binary strings* α *of length at most* h*.*

Proof. Compare (2.3) and (2.4) to obtain that $V_0^{-1} = W_0$, $V_1^{-1} = S^{-1} = W_1$. Recursively extend this observation to all binary strings α, to complete the proofs of both of Proposition 11.4 and, consequently, of the correctness of Algorithm 11.1. ∎

Similarly to deducing Corollary 10.2, we estimate the complexity of performing Algorithm 11.1. We arrive at the following proposition.

PROPOSITION 11.5. *The ERD modulo a fixed prime* p *of an* $n \times n$ *matrix* A *filled with integers and strongly nonsingular modulo* p *(that is, such that* $(\det A^{(q)}) \bmod p \neq 0$ *for all* q*), as well as* $\det A^{(q)} \bmod p$ *for all* q *can be computed at the cost* $O_A((\log n)\log(n\log p),\ n^\omega)$ *for* ω *of* (4.5)*,* $2 \le \omega < 2.376$*; moreover, this computation can be performed by computing with the* $\tilde{b}$*-bit precision operands for* $\tilde{b} = O(n \log\ p)$*.*

REMARK 11.1. *One can be tempted to simplify Algorithm* 11.1 *and to compute modulo* p *the extending set* $\{V_{\alpha 0}^{-1}\}$ *of the RD of the matrix* V *via a more straightforward application of the techniques of sections* 3–10*. In particular, one may proceed by following the recipe of* [R95]*: first approximate the matrices* $V_{\alpha 0}^{-1}$ *closely enough, then multiply the approximations by appropriate integer multipliers* M_α *to arrive at approximations (within an error norm bounded by less than* $1/2$*) to integer matrices* $M_\alpha V_{\alpha 0}^{-1}$*, and then recover the matrices* $M_\alpha V_{\alpha 0}^{-1}$ *via rounding and* $V_{\alpha 0}^{-1}$ *via divisions by* M_α*. The problem with this approach is in bounding the size of the multipliers* M_α*. We need to have* $\log |M_\alpha| = \tilde{O}(n)$ *in order to support the bit-precision bounds of Proposition* 11.5*, but if we follow the cited recipe, we would only reach the bounds of order* $\tilde{O}(n^2)$ *on* $\log |M_\alpha|$*, which would imply involving extra factor* n *in the bit-precision and the bit-complexity bounds. Here, the notation* $\tilde{O}(s)$ *should be read as* $O(s \log^c s)$ *for a constant* c *independent of* s*.*

12. *p*-adic lifting of the ERDs and the recovery of the inverses, determinants, and ranks of integer matrices. In the previous section, we computed the ERD modulo p of an integer matrix A, which is strongly nonsingular modulo p. We will now compute its p-adic (Newton–Hensel's) lifting, that is, the ERD modulo p^{2^g} of A for a fixed natural $g \ge h = \log n$. We will achieve this by incorporating the known techniques [MC79] for p-adic lifting of matrix inverses into our Algorithm 10.1. In this application we will slightly simplify the algorithm by replacing the four steps of Newton's iteration of (10.4)–(10.6) by a single step of the computation of the matrix

$$X_{\beta,t} = X_{\beta,t,0}(2I - V_{\beta 0,t} X_{\beta,t,0}), \tag{12.1}$$

where

$$X_{\beta,t,0} = \begin{cases} V_{\beta 0,t}^{-1} \bmod p \text{ for } t = u(\beta) + 1, \\ X_{\beta,t-1} \text{ for } t > u(\beta) + 1, \end{cases} \tag{12.2}$$

and all matrices $V_{\beta 0,t}^{-1} \bmod p$ are supplied as an input to the p-adic lifting algorithm. (The latter expression for $X_{\beta,t,0}$ replaces (10.4).) The only other change versus Algorithm 10.1 is that all the arithmetic operations in (10.7) and (12.1) are performed modulo p^{2^s} for $s = t - 1 - u(\beta)$ and for $u(\beta)$ denoting (as in section 10) the number of bits one in a binary string β. Hereafter we refer to the resulting algorithm as Algorithm 12.1.

Correctness of the resulting algorithm follows because (12.1) and the inductive assumption that $X_{\beta,t-1} = V^{-1}_{\beta 0,t-1} \bmod p^{2^s}$, $s = t-2-u(\beta)$, together imply that

$$(I - V_{\beta 0,t}X_{\beta,t} - (I - V_{\beta 0,t}X_{\beta,t,0})^2) \bmod p^{2^{s+1}} = 0\ ,$$

and, therefore,

$$X_{\beta,t} = V^{-1}_{\beta 0,t} \bmod p^{2^{s+1}} \tag{12.3}$$

(compare [MC79] or [BP94, Fact 3.3.1, p. 244]).

The *arithmetic complexity* estimates $O_A((g+\log n)\log n, n^\omega)$ of Proposition 10.1 are extended to the case of Algorithm 12.1, where g denotes a fixed natural input value, $g \geq h = \log n$.

We will keep assuming that p is a prime fixed in the interval J of Proposition 11.3, $\|A\| > 1$, $n > 1$, and the matrix F of (11.1) is strongly nonsingular. Furthermore, hereafter we will assume that

$$g = 1 + \left\lfloor \log \frac{1 + n\log \|A\|}{\log p} \right\rfloor . \tag{12.4}$$

Then, we have

$$4\|A\|^{2n} \geq p^{2^g} > 2\|A\|^n. \tag{12.5}$$

Therefore, by the virtue of Proposition 2.4, the value $0.5p^{2^g}$ exceeds $|\det A|$ as well as the maximum absolute value of any entry of adj A. We observe that

$$q = \begin{cases} q \bmod p & \text{if } q \bmod p < 0.5q, \\ (q \bmod p) - p & \text{otherwise,} \end{cases}$$

provided that q is an integer and $2|q| < p$. These observations, Corollary 4.6, and relations (12.5) together enable us to recover $\det A$ from $(\det A) \bmod p^{2^g}$ and adj A from $(\text{adj } A) \bmod p^{2^g}$, as the p-adic lifting of the ERD is completed. Then, we may immediately compute $A^{-1} = (\text{adj } A)/\det A$, since A is a nonsingular matrix.

REMARK 12.1. *We may control the computational precision at the last lifting stage (where the precision is the largest) simply by performing this stage modulo p^q, where $q = \lfloor \log(2\|A\|^n) \rfloor + 1$, so that $2\|A\|^n \leq p^q \leq 2p\|A\|^n$.*

Summarizing the algorithms and the complexity estimates of this and the previous sections, we arrive at the following proposition.

PROPOSITION 12.1. *Let A be a strongly nonsingular $n \times n$ matrix filled with integers. Let $n > 1$, let $\|A\| > 1$, and let p be a prime from the interval J of Proposition* 11.3 *for a fixed $\rho > 2$. Furthermore, let the matrix A be strongly nonsingular modulo p too. Then, one may compute A^{-1} and $\det A^{(k)}$, $k = 1, 2, \ldots, n$, in two stages that amount essentially to application of Algorithms* 11.1 *and* 12.1, *respectively, and are performed at the arithmetic cost bounded by $O_A((\log n)\log(n\log p), n^\omega)$, at the first stage (compare Proposition* 11.5*) and $O_A((\log n)\log(n\log ||A||), n^\omega)$, at the second stage, for ω of* (4.5), $2 \leq \omega < 2.376$.

Assuming p chosen from the interval J of Proposition 11.3, we obtain that $\log p = O(\log(n\log ||A||))$, so that the overall arithmetic cost is dominated by the cost of the second stage.

COROLLARY 12.2. *Under the assumptions of Proposition* 12.1, *one may compute A^{-1} and $\det A^{(k)}$ for $k = 1, 2, \ldots, n$, at arithmetic cost $O_A((\log n)\log(n\log \|A\|), n^\omega)$ for ω of* (4.5).

Let us extend Proposition 12.1 and Corollary 12.2 to estimate at first the bit-precision and then the Boolean complexity of the same computations.

We immediately recall the bound $O(n \log\ p)$ on the bit-precision required in Algorithm 11.1, that is, at the first stage of the computations of Proposition 12.1. At the second stage (that is, essentially for Algorithm 12.1), we revisit the derivation of Proposition 10.1, where we estimated the complexity of the stage of numerical approximation of the RD and ERD of V, and recall or estimate again that this stage is essentially reduced to at most $g+h$ substages for g of (12.4) and for $h = \log n$, such that the cost of performing each substage is dominated by the cost of ten steps of multiplication of at most n/k pairs of $k \times k$ matrices for $k = 2^\ell$ and $\ell = 0, 1, \dots, h-1$. At the stage of the application of Algorithm 12.1, only four (instead of ten) steps are needed. At each of such four steps, all the matrix multiplications are performed concurrently, as in the case of the derivation of Proposition 10.1. Furthermore, at every step of Substage t of the second stage, $t = 1, \dots, g+h$, at most $n/2^l$ pairs of matrices of the sizes $2^l \times 2^l$ are encountered for $l = n - |\beta| - 1, u(\beta) < t$. Such matrices are pairwise multiplied together modulo $p^{2^{t-u(\beta)}}$,

$$t - u(\beta) \le \lambda(t) = \min\{t, g\}. \tag{12.6}$$

The above bounds on the modulo imply some bit-precision bounds since computation modulo ℓ can be performed with $2\lceil \log \ell \rceil$-bit-precision. Furthermore, we recall the known estimates $O_B((\log k) \log\log k, k)$ for the Boolean complexity of performing an arithmetic operation modulo $2^k - 1$ (see [AHU74], [BP94], [CK91], [RT90]), which can be extended to our computations whenever we perform them with k-bit-precision.

By combining the latter estimates with estimates for the arithmetic cost and for the bit-precision of our computations, we bound the Boolean cost of performing Algorithm 11.1, that is, the first stage of the computations supporting Proposition 12.1 (cf. Corollary 10.1) by

$$O_B((\log n)(\log(n \log p))^2 \log\log(n \log p), n^{\omega+1} \log\ p),$$

and we bound the Boolean cost of performing the tth stage of Algorithm 12.1 by

$$O_B((\log n)(\log(2^{\lambda(t)} \log p)) \log\log(2^{\lambda(t)} \log p), n^\omega 2^{\lambda(t)} \log p),\ t = 1, \dots, g+h.$$

(Compare (12.6) and recall that the tth stage of Algorithm 12.1 is the tth substage of the second stage of the computations of Proposition 12.1.)

By summarizing all these estimates, for p lying in the interval J of Proposition 11.3 and for g satisfying (12.4), (12.5), we estimate the Boolean complexity of our computations. To simplify the expressions for the resulting estimates, we write

$$A = (a_{i,j}), \quad a = \log \max_{i,j} |a_{i,j}| \tag{12.7}$$

and obtain that $g = O(\log(na))$, $g + h = O(\log(na))$, $2^g \log p = O(na)$ for g of (12.4), $\log p = O(\log(na))$, $\log(n \log p) = O(\log(n \log a))$. Then, we rewrite our Boolean cost bounds as follows:

$$O_B((\log n)(\log(n \log a))^2 \log\log(n \log a), n^{\omega+1} \log(na))$$

for performing Algorithm 11.1,

$$O_B((\log n)(\log(na)) \log\log(na), n^{\omega+1} a)$$

for performing the tth stage of Algorithm 12.1 for $t = g+1, \dots, g+h$, where $\lambda(t, g) = g + 1$, and

$$O_B((\log n)(t + \log\log(na)) \log(t + \log\log(na)), n^\omega 2^t \log p)$$

for performing the tth stage of Algorithm 12.1 for $t = 1, \dots, g$, where $\lambda(t, g) = t$. By applying the B-principle, we bound the overall cost of performing the first $g = O(\log(na))$ stages of Algorithm 12.1 by

$$O_B((\log n)(\log(na))^2 \log\log(na), n^{\omega+1} a/\log(na)),$$

and we bound the overall cost of performing its last $h = \log n$ stages by

$$O_B((\log n)^2(\log(na)) \log\log(na), n^{\omega+1} a).$$

Then again, we apply the B-principle to yield the same parallel Boolean time bound, $O((\log n)(\log(na))^2 \log\log(na))$, in all the three estimates (for Algorithm 11.1, for the first g stages of Algorithm 12.1, and for its last h stages), which gives us the Boolean processor bounds

$$O((n^{\omega+1}(\log(n \log a))^2 \log\log(n \log a))/(\log(na) \log\log(na)))$$

$$= O(n^{\omega+1}(\log(n \log a))^2/\log(na)),$$

$$O(n^{\omega+1} a/\log(na)),$$

and

$$O((n^{\omega+1} a \log n)/\log(na))$$

for these three groups of computations, respectively. We note that the sum of the three latter bounds gives us $O((\log n)(a + \log n)n^{\omega+1}/\log(na))$.

By using the Boolean cost bounds of Proposition 11.5 for computing $\det A^{(k)} \bmod p$ for all k, and by combining the cited Boolean time bound and the latter processor bound, we obtain the following proposition.

PROPOSITION 12.3. *Under the assumptions of Proposition* 12.1, *one may compute the inverse matrix* $A^{-1} \bmod p$ *and* $\det A^{(k)} \bmod p$, $k = 1, \dots, n$, *at the Boolean cost*

$$O_B((\log n)(\log(n \log a))^2 \log\log(n \log a), n^{\omega+1} \log(na)),$$

and one may compute the matrix A^{-1} *and* $\det A^{(k)}$, $k = 1, \dots, n$, *at the Boolean cost* $O_B((\log n)(\log(na))^2 \log\log(na), (\log n)(a + \log n)n^{\omega+1}/\log(na))$ *for* ω *of* (4.5) *and* a *of* (12.7).

REMARK 12.2. *Our choice of a prime* p *and our complexity estimates rely on the bounds of Proposition* 2.4 *on* $|\det W|$. *For a large class of matrices* W, *such bounds can be refined a little (e.g., by using Hadamard's upper bound on* $|\det A|$*) and so can our complexity estimates. Likewise, by expressing the estimates of Proposition* 12.3 *in terms of* $||A||$ *rather than* a, *one may obtain some slightly refined (though more complicated) estimates. Finally, our estimates for parallel Boolean cost can be slightly improved if, instead of the bounds* $O_B((\log k) \log\log k, k)$ *on the cost of an arithmetic operation, we will rely on the bounds* $O_B(\log k, k \log\log k)$, *which hold for the cost of*

an addition, a subtraction and a multiplication (see, e.g., [BP94, p. 297]*). We may rely on the latter bound because the ops of the latter three classes are most numerous among all the ops in our algorithms. Similar observations apply to the estimates of Theorems* 1.1 *and* 1.2.

It remains to work out the strong nonsingularity issue in order to extend the complexity estimates of Corollary 12.2 and Proposition 12.3 to estimates of Theorem 1.1. (Note that, in terms of a, the bounds of Corollary 12.2 turn into $O_A((\log n)\log(na), n^\omega)$, as required in Theorem 1.1.)

We will first assume that A is a nonsingular matrix. In this case, AA^T is an s.p.d. matrix and, consequently, a strongly nonsingular matrix, by Corollary 2.11. Consequently, AA^T is strongly nonsingular modulo p, with a probability $1 - P_{\rho,n}$ for $P_{\rho,n}$ bounded according to Proposition 11.3. Therefore, we may apply the results of this section to compute at first $(AA^T)^{-1}$ and then $A^{-1} = A^T(AA^T)^{-1}$ and $\mathbf{x} = A^{-1}\mathbf{f}$ satisfying $A\mathbf{x} = \mathbf{f}$. (Strong nonsingularity (modulo p) of AA^T is tested as a by-product of computing $(AA^T)^{-1}$.) We may also immediately compute $\det(AA^T) = (\det A)^2$, though this does not give us the sign of $\det A$. The matrix A is singular (that is, $\det A = 0$) if and only if application of the same approach to a matrix A requires us to invert a singular matrix at some step.

Next, we will apply randomization to relax the assumptions about (strong) nonsingularity of A when we compute rank A and the sign of $\det A$. Towards this goal, we fix $\rho > 2$, a sufficiently large finite set of integers, S, and two matrices U and L, as specified in Proposition 2.19; we compute the matrix $\tilde{A} = UAL$ (cf. Remark 12.3 at the end of this section), fix a random prime p in the interval J of Proposition 11.3, and extend Algorithm 11.1 to compute $(\det \tilde{A}^{(k)}) \bmod p$ for $k = 1, \dots, n$, and $r(p) = \max\{k,\ (\det\ \tilde{A}^{(k)}) \bmod p \neq 0\}$. Let us write $\tilde{r} = \max\{k,\ \det\ \tilde{A}^{(k)} \neq 0\}$, so that rank $A \geq \tilde{r} \geq r(p)$. Furthermore, $\tilde{r} = $ rank A with a probability at least $P_r = 1 - (\tilde{r}+1)\tilde{r}/|S|$ (due to Proposition 2.19), and $\tilde{r} = r(p)$, with a probability $1 - P_{\rho,n}$, estimated in Proposition 11.3. Thus, we output $r(p)$ as rank A and arrive at the estimate of Theorem 1.1 for the randomized cost of computing rank A. (Note that in this case, the computations modulo p suffice; thus, in our computation of rank A, we omit the p-adic lifting stage and rely on the first Boolean cost estimate of Proposition 12.3.)

Let us extend this technique to the computation of the sine of $\det A$. If $r(p) < n$, then $(\det A) \bmod p = 0$, and we output $\det A = 0$, which is correct with a probability at least $1 - P_{\rho,n}$. Otherwise, that is, if $r(p) = n$, then we have $n \geq$ rank $A \geq r(p) = n$; that is, A is nonsingular. Furthermore, by using the randomization based on Proposition 2.19, we may compute $\det A = \det(UAL)$, because UAL is strongly nonsingular, with a probability at least $1-(n+1)n/|S|$ if A is nonsingular. By letting $|S| = n^4$, say, and by applying Propositions 11.5 and 12.3 to the matrix UAL, we arrive at the desired algorithm for $\det A$, supporting Theorem 1.1.

Now, assume that $r(p) < n$ and that the $r(p) \times r(p)$ leading principal submatrix $B = \tilde{A}^{(r(p))}$ of $\tilde{A}$ is nonsingular. Let us write $\tilde{A} = \left(\begin{smallmatrix} B & C \\ D & E \end{smallmatrix}\right)$, $G = \left(\begin{smallmatrix} I & -B^{-1}C \\ 0 & I \end{smallmatrix}\right)$, and observe that $\tilde{A}G = \left(\begin{smallmatrix} B & 0 \\ D & Q \end{smallmatrix}\right)$, where $Q = 0$ if and only if $r(p) =$ rank A. (Compare [KP91] and [BP94, pp. 110 and 333].) This gives us an algorithm for verification whether $r(p)$=rank A (at the cost within the asymptotic cost bounds of Theorem 1.1). If so, then the $n-r$ columns of the matrix $LG\left(\begin{smallmatrix} 0 \\ I \end{smallmatrix}\right)$, where I denotes the $(n-r)\times(n-r)$ identity matrix, give us a basis for the null-space, $\mathbf{N}(A)$, of A (compare Definition 2.17). We recall from Fact 2.1 that if there exists a solution $\mathbf{x}$ to a linear system $A\mathbf{x} = \mathbf{f}$, then it can be represented as $\mathbf{x} = \mathbf{x}_0 + \mathbf{z}$, $\mathbf{x}_0$ being a fixed specific solution

and $\mathbf{z}$ being a vector from $\mathbf{N}(A)$.

Let $\mathbf{g}$ be the r-dimensional prefix-subvector of $\mathbf{f}$, made by the first r components of $\mathbf{f}$. Let $\mathbf{y} = B^{-1}\mathbf{g}$ be the solution to the nonsingular system $B\mathbf{y} = \mathbf{g}$. Then, a specific solution $\mathbf{x}_0$ to the system $UA\mathbf{x} = U\mathbf{f}$ is given by $\mathbf{x}_0 = LG\begin{pmatrix}\mathbf{y}\\ \mathbf{0}\end{pmatrix}$ if the latter linear system is consistent, and we have $UA\mathbf{x_0} \neq U\mathbf{f}$ otherwise. This completes our proof of Theorem 1.1. □

REMARK 12.3. *Our computations supporting Theorem* 1.1 *include some* $n \times n$ *matrix multiplications (of* A *by* A^T, L, *and* U*). Their cost bound is dominated by the complexity bounds of Theorem* 1.1, *and a similar argument applies to yield the extension of this theorem to Theorem* 1.2, *to be shown in section* 14 *(cf. Proposition* 14.1*). The increase of the matrix norm in the transition from* A *to* AA^T *and* $\tilde{A} = UAL$ *may cause the increase only by a constant factor in the estimate for the precision of the computations and their Boolean complexity (if we choose, say,* $S = \{1, 2, \dots, |S|\}$ *and* $|S| = n^{O(1)}$*).*

13. Some definitions and auxiliary results on computations with structured matrices. Our next goal is to show that the computational cost of our algorithms supporting Theorem 1.1 decreases dramatically, to the level of the estimates of Theorem 1.2, provided that the input matrix has Toeplitz-like structure. In this section we will recall some definitions and some simple and/or well-known facts on Toeplitz-like matrices, which we will use in the next section towards the stated goal (cf. (1.1) and (1.2) of section 1.1, Definition 2.18, and [BP94], [CKL-A87], [KKM79], [P92]).

PROPOSITION 13.1. *The product of a* $k \times k$ *Toeplitz matrix (cf. Definition* 2.18*) and a vector of dimension* k *can be computed at the cost* $O_A(\log k, k)$ *(via reduction to three FFTs, each on* $O(k)$ *points, or to convolution of two vectors of dimension* $O(k)$*).*

DEFINITION 13.2. *For a* $k \times k$ *matrix* A *and for the matrix* Z *of Definition* 2.18, *write* $F_+(A) = A - ZAZ^T$, $F_-(A) = A - Z^TAZ$. *If* $F(A) = GH^T$ *for a pair of* $k \times \ell$ *matrices* G *and* H *and for* $F = F_+$ *or* $F = F_-$, *then the pair of* G, H *is called an* F-generator *of* A *of length* ℓ. *(Note that, in this case, the pair* H, G *is an* F*-generator of* A^T *of the same length.) The minimum length* ℓ *of an* F-generator *of* A, *for fixed* A *and* F, *is called the* F-rank *of* A, *is denoted by* $r_F(A)$, *and is equal to* rank $F(A)$. *A* $k \times k$ *matrix* A *is called a* Toeplitz-like matrix *if it is given with its* F*-generator (for* $F = F_+$ *or* $F = F_-$*) having a length bounded by a constant independent of* k. F*-generators and* F*-ranks, for both* $F = F_+$ *and* $F = F_-$, *are also called* displacement generators *and* displacement ranks *(following the original definitions of* [KKM79]*).*

PROPOSITION 13.3. $r_F(T) \leq 2$ *if* T *is a Toeplitz matrix, and* $r_F(T) \leq 1$ *if* T *is a triangular Toeplitz matrix for* $F = F_+$ *and* $F = F_-$. *In particular,* $r_F(I) = 1$.

The correlation to (1.2) is given by the following result.

PROPOSITION 13.4. G, H *is an* F_+*-generator (respectively,* F_-*-generator) of* A *having a length* ℓ, $G = (\mathbf{g_1}, \dots, \mathbf{g}_\ell)$, $H = (\mathbf{h_1}, \dots, \mathbf{h}_\ell)$, *if and only if* $A = \sum_{s=1}^{\ell} L(\mathbf{g_s})\, L^T(\mathbf{h_s})$ *(respectively, if and only if* $A = \sum_{s=1}^{\ell} L^T(\mathbf{g_s})\, L(\mathbf{h_s})$*).*

Based on the latter results, we will operate with the F-generators of Toeplitz-like matrices, rather than with the matrices themselves. Such a representation is memory space efficient and also enables us to use less sequential time and fewer processors in Toeplitz-like computations, due to the following corollary (cf. Propositions 13.1 and 13.4).

COROLLARY 13.5. *The product of a* $k \times k$ *Toeplitz-like matrix by a vector of dimension* k *can be computed at the cost* $O_A(\log k, k)$.

The next result gives us more specific estimates—the cost bound of Toeplitz-like matrix multiplication is proportional to the square of the sum of the lengths of the F-generators of the input matrices, and such a length is roughly doubled in a matrix addition or multiplication.

PROPOSITION 13.6. *Given F-generators, G_A, H_A of length ℓ_A and G_B, H_B of length ℓ_B, of $k \times k$ matrices A and B, respectively (for $F = F_+$ or $F = F_-$), one may compute an F-generator G_{AB}, H_{AB} of AB of length at most $\ell_A + \ell_B + 1$ at the cost $O_A(\log k, (\ell_A + \ell_B)^2 k)$, whereas an F-generator of $A + B$ of length at most $\ell_A + \ell_B$ is immediately available cost-free.*

In view of the latter results, we will study various bounds on the F-ranks and the length of F-generators, in particular regarding the matrices involved in the RD and Newton's iteration with Toeplitz-like input.

PROPOSITION 13.7.

(a) *$r_{F+}(A) \le r_{F-}(A) + 2$, $r_{F-}(A) \le r_{F+}(A) + 2$ for any matrix A. Furthermore, an F_+-generator (respectively, F_--generator) of a length ℓ for any matrix A can be immediately transformed (at the cost $O_A(\log n, n)$ of performing $O(1)$ convolutions or FFTs) into an F_--generator (respectively, F_+-generator) of length at most $\ell + 2$ for A.*

(b) *If A is nonsingular, then $r_{F+}(A^{-1}) = r_{F-}(A)$.*

The next result is immediately verified (compare Definition 2.6).

PROPOSITION 13.8. *Let $GH^T = F_+(W)$ for a $k \times k$ matrix W. Then $(GH^T)^{(i)} = F_+(W^{(i)})$ for $i = 1, 2, \dots, k$; furthermore, $r_{F_+}(C) \le r_{F_+}(W) + 1$, $r_{F_+}(E) \le r_{F_+}(W) + 1$, under (2.1), and $r_{F_+}(T) \le r_{F_+}(W) + 2$ for any submatrix T of W formed by contiguous sets of row and columns of W.*

It follows that $r_{F_+}(B) \le r_{F_+}(W)$, under (2.1).

We observe similar relations for trailing principal submatrices and the operator F_-. By Proposition 2.7, S^{-1} is a trailing principal submatrix of W^{-1}. Therefore, $r_{F_-}(S^{-1}) \le r_{F_-}(W^{-1})$. By applying Proposition 13.7 (b) for $A = S$ and $A = W$, we obtain that $r_{F_+}(S) \le r_{F_+}(W)$.

PROPOSITION 13.9. *Let (2.1) and (2.2) hold, where B, S, and W are nonsingular matrices. Then $\max\{r_{F_+}(B), r_{F_+}(S)\} \le r_{F_+}(W)$.*

By applying the latter proposition recursively, we bound the F_+-rank throughout the RD.

COROLLARY 13.10. *Let V_α be a matrix of the RD of a matrix A. Then, $r_{F_+}(V_\alpha) \le r_{F_+}(A)$.*

So far, we have no tools yet to counter the growth of the length of the F-generators in the process of Newton's iteration. Developing such tools (which we call the techniques for *the truncation of a generator (TG)*) is our next task. Namely, we will next (in Proposition 13.11) show how to compute a shorter F-generator of a matrix having small F-rank but given with its longer F-generator. This is our first technique of TG. It will be used to refine p-adic (Newton–Hensel's) lifting to bound the length of the F-generators of the matrices involved there. We will prove easily, based on Propositions 13.7 and 13.9, that such matrices have small F-rank if so has the input matrix. For Newton's iteration of Algorithm 5.1, such a property does not hold, and the F-rank of the computed approximations to the Toeplitz-like inverses may grow quite rapidly. These approximations, however, always have matrices with small F-rank nearby, and we will periodically shift to the latter matrices and then restart Newton's process. Our tool for such a shift will be Algorithm 13.1 (see [PBRZ99] on some alternative tools).

PROPOSITION 13.11. *Let an F-generator of a $k \times k$ matrix A of length ℓ (for $F = F_+$ or $F = F_-$) and an upper bound $r^* < l$ on the F-rank $r_F(A)$ be given. Then an F-generator of A of length at most r^* can be computed at the cost $O_A(l,\ kl)$.*

Proof. Apply the proof of Proposition A.6 of [P92] or the solution of Problem 2.11 of [BP94, pp. 111–112]. Verify that all the computations (including the computation of the LSP factorization or, alternatively, the PLU factorization) can be performed at the claimed overall cost. □

Let us next show the promised alternative algorithm for controlling the length of F-generators of matrices involved in Newton's process. The algorithm relies on the *SVD truncation of F-generator*, which is our second TG technique.

Algorithm 13.1 ([P92b], [P93], [P93a]).

Input: $F = F_+$ or $F = F_-$, an F-generator G, H of a $k \times k$ matrix A of length l, and a natural $r' < l$.

Output: an F-generator G', H' of a $k \times k$ matrix A' of length at most r' such that

$$\|A' - A\|_2 \le 2(1 + 2(r_F(A) - r')k) \min_Y \|Y - A\|_2 \ , \tag{13.1}$$

where the minimum is over all $k \times k$ matrices Y of F-rank at most r'.

Computations:

Stage 1. Compute the singular value decomposition (SVD) of the matrix $GH^T = F(A)$; that is, compute a pair U and V of unitary $k \times l$ matrices and an $l \times l$ diagonal matrix $\Sigma = \mathrm{diag}(\sigma_1, \dots, \sigma_l)$ for positive $\sigma_1, \dots, \sigma_l$ satisfying

$$GH^T = F(A) = U\Sigma V^T.$$

Stage 2. Compute and output an F-generator G', H' of A' of length at most r' as follows:

$$G' = U\Sigma_{r'}, \quad H' = VI_{r',l},$$

where $\Sigma_{r'} = \mathrm{diag}(\sigma_1, \dots, \sigma_{r'}, 0, \dots, 0)$ and $I_{r',l} = \mathrm{diag}(1, \dots, 1, 0, \dots, 0)$ are $l \times l$ matrices of rank r'.

On the correctness proof of this algorithm, on the bound $O_A(\log k, k/\log k)$ for $l = O(1)$, and on the *computational cost* of its performance, see [P92b], [P93], [P93a].

REMARK 13.1. *Bound* (13.1) *is proved in* [P92b], [P93], [P93a], *based on approximate computation of the SVD at Stage* 1 *of the algorithm. Any improvement of the approximation of the SVD would decrease the factor* 2 *of* (13.1), *which turns into* 1 *if the SVD is computed exactly.*

REMARK 13.2. *If $r' \ge r_F(A)$, then* (13.1) *implies that $\|A' - A\|_2 = \min_Y \|Y - A\|_2 = 0$, and then Algorithm* 13.1 *is an alternative to the algorithm supporting Proposition* 13.11, *except that the latter algorithm is rational (it can be performed with no errors over the rational), whereas Algorithm* 13.1 *has a nonrational, though numerically stable stage of computing the SVD. This suggests that the algorithm supporting Proposition* 13.11 *should be applied in Algorithm* 12.1, *at the p-adic lifting stage, whereas Algorithm* 13.1 *is a better candidate to use in numerical applications of Algorithm* 11.1, *performed with rounding.*

14. Improvement of the algorithms for the ERD, IRD, inverse, determinant, and rank in the Toeplitz and Toeplitz-like cases. Let us apply the techniques and the results of the previous section to reexamine the computation of

the ERD and IRD of a strongly nonsingular $n \times n$ matrix A filled with integers in the case where A is a Toeplitz or Toeplitz-like matrix given with its F_+-generator G, H of length $r = r_{F_+}(A) = O(1)$.

We recall that $r_{F_+}(V_\alpha) \le r$ for all matrices V_α of the RD of A (compare Corollary 13.10), and we will apply either the algorithm supporting Proposition 13.11 or Algorithm 13.1 in order to decrease (to a level at most r) the length of the computed F-generators of these matrices, in all cases where this length exceeds r. Likewise, we will obtain from Propositions 13.6–13.8 that the computation of $V_{\beta 1,t+1}$, according to (10.1)–(10.7), only involves matrices whose F-ranks are bounded from above by $3r + r_{F_+}(X_{\beta,t}) + 6$.

According to our analysis, the matrix $X_{\beta,t}$ approximates $B_{\beta 0}^{-1}$ for all binary strings β of length at most $h-1$, and since $r_{F_+}(B_{\beta 0}) \le r$, we have $r_{F_-}(B_{\beta 0}^{-1}) \le r$, $r_{F_+}(B_{\beta 0}^{-1}) \le r+2$ (compare Proposition 13.7). We will apply Algorithm 13.1 in order to compute an F_+-generator of length at most $r+2$ for a matrix $X'_{\beta,t}$ approximating $X_{\beta,t}$ and, therefore, also $V_{\beta 0}^{-1}$. (The approximation of $V_{\beta 0}^{-1}$ by $X'_{\beta,t}$ deteriorates slightly, versus the approximation by $X_{\beta,t}$, but since $X'_{\beta,t}$ still closely approxiamtes the matrix $V_{\beta 0}$, we more than compensate ourselves for such a deterioration by performing an extra Newton step in (10.5).) Then, all matrices involved in the computation of the ERD and the IRD of A will be represented by their F_+-generators of length $O(r)$.

A similar argument is applied to the computation of the p-adic lifting of the ERD of A, except that this argument is simplified since (12.3) and Proposition 13.7 together imply that

$$\begin{aligned} r_{F_+}(X_{\beta,t+1} \bmod p^{2^{t-u(\beta)}}) &\le r_{F_-}(X_{\beta,t+1} \bmod p^{2^{t-u(\beta)}}) + 2 \\ &= r_{F_+}(B_{\beta 0}^{-1} \bmod p^{2^{t-u(\beta)}}) + 2 \le r+2. \end{aligned}$$

Thus, to keep the length of the associated F_+-generators bounded, we just apply the rational algorithm that supports Proposition 13.11, instead of applying Algorithm 13.1. In fact, we may also apply other alternative techniques for bounding the length of an F-generator of $X_{\alpha,i+1}$; such techniques may rely on using distinct operators F, such as $F^+(A) = AZ - ZA$ (see [BP94, p. 189]) or operators using some f-circulant matrices instead of Z (see [PBRZ99], [P00]).

Finally, it is easily verified (cf. [P96b]) that the computation (of section 12) of a basis for the null-space of A also involves only matrices represented by their F_+-generators of length $O(r)$ for a matrix A given with its F_+-generator of length r.

Let us now turn to estimating the computational cost, in the case of Toeplitz or Toeplitz-like input. There are two new features versus the case of a general integer input matrix A.

(1) Performing every matrix multipication, we operate with F_+-generators of Toeplitz-like matrices involved in these multiplications and apply Propositions 13.4, 13.6, and Corollary 13.5.

(2) Some of these matrix multiplications are followed by the application of the algorithms supporting Proposition 13.11 or Algorithm 13.1.

The manipulation with the F_+-generators enables us to decrease the arithmetic processor bound of Corollary 12.2 from n^ω to $n \log n$, because concurrent multiplications of $O(2^t)$ pairs of $(n/2^t) \times (n/2^t)$ Toeplitz-like matrices for $t = 1, \dots, h$, $h = \log n$ are performed at the overall cost bounded by $O_A(\log n, n \log n)$ (versus $O_A(\log n, n^\omega)$ in the case of general integer input matrices). The estimated overall cost of the required computations (of A^{-1}, det A, and so on) is dominated by the estimated cost of

all Toeplitz-like matrix multiplications involved, because, according to section 13, the estimated cost of such a multiplication dominates the estimated cost of the application of both Algorithm 13.1 and the algorithm supporting Proposition 13.11.

Summarizing, we obtain the following result.

PROPOSITION 14.1. *If the $n \times n$ input Toeplitz-like matrix A is strongly nonsingular and is filled with integers, then one may modify the randomized computation of its ERD and IRD according to the algorithms of sections* 6–12 *in order to perform all these computations at the overall cost* $O_A((\log n)\log(n\log\|A\|), n\log n)$.

The cost bounds of Proposition 14.1 are immediatley extended to the solution of all the computational problems listed in Theorem 1.1, where now we assume a Toeplitz-like input matrix A and represent its inverse or the basis matrix for its nullspace by their short F-generators. (Verifying the correctness of the computation of the rank and the inverse, we should also deal with short F-generators and use Proposition 13.11 to avoid processing n^2 entries of $n\times n$ matrices, which would have required order of n^2 ops.)

To obtain a similar extension of the Boolean complexity bounds of Proposition 12.3 and Theorem 1.1, let us examine the precision of the computations by our algorithms simplified in the Toeplitz-like case. We recall that our Toeplitz-like computations can be ultimately reduced to vector convolutions (Propositions 13.1, 13.4, and 13.6). Thus, we will bound the cost of our computations at the p-adic lifting stage based on the following estimate.

PROPOSITION 14.2. *Given two vectors of dimension n filled with integers lying in the range from* 0 *to* 2^k-1, *the convolution of these vectors can be computed at the Boolean cost* $O_B((\log(kn), kn\log\log(kn))$.

Proof. The well-known *binary segmentation* techniques (see, e.g., [BP94, section 3.9]) reduces our convolution problem to the multiplication of two integers lying in the range from 0 to $2^{kn}-1$, and the known algorithms solve this task at the required cost. □

The resulting Boolean cost bounds for performing the p-adic lifting stage will repeat the bounds of section 12, except that the Boolean (like arithmetic) processor bounds will decrease by factor $n^{\omega-1}/\log n$.

Let us show that this holds also for the Boolean cost of the rest of our computation.

When we approximate the ERD of an input Toeplitz-like matrix, we will effectively reduce the computations to performing FFTs (see Propositions 13.1 and 13.4) and will recall Corollary 3.4.1 on pp. 255–256 of [BP94], which shows a numerically stable implementation of FFT. We also recall that the known algorithms for the computation of the SVD of a matrix are numerically stable (see [GL89/96], [P93]). From these observations, we deduce that we may perform the computations with the same bit-precision (up to a constant factor independent of n), no matter whether we apply our original Algorithm 11.1 for an arbitrary $n \times n$ input matrix or its Toeplitz-like modification. Since in the latter case we use by factor $n^{\omega-1}/\log n$ fewer arithmetic processors, we will also use by factor $n^{\omega-1}/\log n$ fewer Boolean processors, thus replacing n^ω for ω of (4.5) by $n\log n$ in the Boolean cost estimates of section 12.

This enables us to extend Theorem 1.1 to arrive at Theorem 1.2. □

REMARK 14.1. *Inspection of our algorithms shows immediately that Proposition* 14.1 *and Theorem* 1.2 *can be extended to the case where the input matrix A is given with its F-generator of length r, provided that both time and processor bound increase by factor r. It is possible to confine the cost increase to processor bound (increasing it*

by factor r^2*). The only nontrivial stage is the decrease of the length of* F*-generators (cf. Proposition* 13.11 *and Algorithm* 13.1*). The algorithm supporting Proposition* 13.11*, however, can be modified by extending the probabilistic techniques of the proof of Theorem* 1.1 *(this would include, in particular, application of Proposition* 2.19 *using* $n + r$ *extra random parameters), whereas Algorithm* 13.1 *should be replaced by an alternative approach of* [PBRZ99].

REMARK 14.2. *It may seem that Theorems* 1.1 *and* 1.2 *can be supported by a substantially simpler construction, and simplified construction has indeed been proposed in* [R95]. *Unfortunately, however, the construction of* [R95] *has no power for supporting the claimed results. In particular, the construction relies on the two "simplifying" recipes cited in our Remarks* 6.1 *and* 11.1*, and each of the recipes invalidates the resulting algorithm. (See* [P96c] *for more details on these and some other of the many mishaps of* [R95], and note also that the main result of the paper [R93], cited in [R95], is a rediscovery of some results of [BT90] and [BP91].) *It is instructive, for getting better insight, to discuss two other major gaps of the construction of* [R95] *and of its analysis presented in* [R95]. *Both gaps are in area of Toeplitz-like computations, where* [R95] *becomes particularly prone to serious errors. In* [R95]*, an algorithm of* [BA80] *is used in order to decrease the length of an* F*-generator of a matrix* A *to the level* $r = \text{rank } F(A)$. *Unlike our Algorithm* 13.1 *for the SVD truncation and our algorithm supporting Proposition* 13.11*, the algorithm of* [BA80] *only works if* $(F(A))^{(r)}$*, the* $r \times r$ *l.p.s of* $F(A)$*, is nonsingular. Furthermore, to support the algorithm of* [R95]*, one must have matrix* $(F(A))^{(r)}$ *well-conditioned. Actually, to salvage the algorithm of* [R95] *at this point, one would have had to use some techniques that are absent from* [R95] *and are substantially more advanced than ones used in* [R95]. *Likewise, some techniques are required to prevent the* F*-ranks of the computed approximations to* A^{-1} *from their disturbing growth (from the desired constant level to the level* n*) in less than* $\log n$ *Newton's steps, and then again, such techniques are absent from* [R95] *and are substantially more advanced than ones used in* [R95]. *The growth of the* F*-ranks immediately implies the growth by the extra factor* $n^{\omega-1} \log n$ *(for* ω *of* (4.5)*) of both arithmetic and Boolean processor complexity bounds, versus the ones claimed in* [R95].

15. Discussion. Our paper leaves as a major open question of theoretical importance whether the level of our parallel complexity estimates of Theorem 1.2 for Toeplitz and Toeplitz-like computations can be reached by means of purely algebraic approach, using no rounding to the closest integers. This question is also of practical interest because the algorithms of this paper involve the exact computation of $\det A$ and, therefore, at some stage require us to use the precision of computation of order $\log |\det A|$, which generally means the order of $n \log ||A||$, even if we only need the output with a much lower precision. Historically, a similar open problem had arisen for computations with general integer matrices, after the appearance of [P85], [P87]. In that case (for general integer matrices), the subsequent works of [KP91], [KP92], [P91], and [P92] gave us an alternative randomized algebraic solution that involved no rounding. Will this be eventually done also in the Toeplitz-like case or at least in the Toeplitz case?

Acknowledgments. Detailed and thoughtful comments by a referee and by a reviewer helped me a great deal to improve my original draft and to make it more accessible for the reader. The request by the area editor Joachim von zur Gathen to incorporate the appendix into the body of the paper also served the same goal.

REFERENCES

[AGr88] G. S. Ammar and W. B. Gragg, *Superfast solution of real positive definite Toeplitz systems*, SIAM J. Matrix Anal. Appl., 9 (1988), pp. 61–76.

[AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[B68] E. H. Bareiss, *Sylvester's identity and multistep integer-preserving Gaussian elimination*, Math. Comp., 22 (1968), pp. 565–578.

[BA80] R. R. Bitmead and B. D. O. Anderson, *Asymptotically fast solution of Toeplitz and related systems of linear equations*, Linear Algebra Appl., 34 (1980), pp. 103–116.

[Be68] E. R. Berlekamp, *Algebraic Coding Theory,* McGraw-Hill, New York, 1968.

[Be84] S. Berkowitz, *On computing the determinant in small parallel time using a small number of processors*, Inform. Process. Lett., 18 (1984), pp. 147–150.

[BGH82] A. Borodin, J. von zur Gathen, and J. Hopcroft, *Fast parallel matrix and GCD computation*, Inform. and Control, 52 (1982), pp. 241–256.

[BGY80] R. P. Brent, F. G. Gustavson, and D. Y. Y. Yun, *Fast solution of Toeplitz systems of equations and computation of Padé approximations*, J. Algorithms, 1 (1980), pp. 259–295.

[BK87] A. Bruckstein and T. Kailath, *An inverse scattering framework for several problems in signal processing*, IEEE Acoustics, Speech and Signal Processing (ASSP) Magazine, January 1987, pp. 6–20.

[BL80] D. Bini and G. Lotti, *Stability of fast algorithms for matrix multiplication*, Numer. Math., 36 (1980), pp. 63–72.

[BMP98] D. Bondyfalat, B. Mourrain, and V. Y. Pan, *Controlled iterative methods for solving polynomial systems*, in Proceedings of the Annual ACM International Symposium on Symbolic and Algebraic Computation, ACM, New York, 1998, pp. 252–259.

[BP91] D. Bini and V. Y. Pan, *Parallel complexity of tridiagonal symmetric eigenvalue problem*, in Proceedings of the 2nd Annual ACM-SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1991, pp. 384–393.

[BP93] D. Bini and V. Y. Pan, *Improved parallel computation with Toeplitz-like and Hankel-like matrices*, Linear Algebra Appl., 188/189 (1993), pp. 3–29.

[BP94] D. Bini and V. Y. Pan, *Polynomial and Matrix Computations*, *Fundamental Algorithms* 1, Birkhäuser, Boston, 1994.

[BT71] W. S. Brown and J. F. Traub, *On Euclid's algorithm and the theory of subresultants*, J. ACM, 18 (1971), pp. 505–514.

[BT90] M. Ben-Or and P. Tiwari, *Simple algorithm for approximating all roots of a polynomial with real roots*, J. Complexity, 6 (1990), pp. 417–442.

[Bun85] J. R. Bunch, *Stability of methods for solving Toeplitz systems of equations*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 349–364.

[C47/48] S. Chandrasekhar, *On the radiative equilibrium of a stellar atmosphere*, Astrophys. J., 106 (1947), pp. 152–216, 107 (1948), pp. 48–72.

[C74] R. W. Cottle, *Manifestation of the Schur complement*, Linear Algebra Appl., 8 (1974), pp. 189–211.

[Ch85] A. L. Chistov, *Fast parallel calculation of the rank of matrices over a field of arbitrary characteristics*, in Fundamentals of Computation Theory (Cottbus, 1985), Lecture Notes in Comput. Sci. 199, Springer, Berlin, 1985, pp 63–69.

[CK91] D. G. Cantor and E. Kaltofen, *On fast multiplication of polynomials over arbitrary rings*, Acta Inform., 28 (1991), pp. 697–701.

[CKL-A87] J. Chun, T. Kailath, and H. Lev-Ari, *Fast parallel algorithm for QR-factorization of structured matrices*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 899–913.

[Cs76] L. Csanky, *Fast parallel matrix inversion algorithms*, SIAM J. Comput., 5 (1976), pp. 618–623.

[dH87] F. R. de Hoog, *On the solution of Toeplitz systems*, Linear Algebra Appl., 88/89 (1987), pp. 123–138.

[E67] J. Edmonds, *Systems of distinct representatives and linear algebra*, J. Res. Nat. Bur. Standards, 71B (1967), pp. 241–245.

[EG88] D. Eppstein and Z. Galil, *Parallel algorithmic techniques for combinatorial computation*, Annual Rev. Comput. Sci., 3 (1988), pp. 233–283.

[EP97] I. Z. Emiris and V. Y. Pan, *The structure of sparse resultant matrices*, in Proceedings of the Annual ACM International Symposium on Symbolic and Algebraic

Computation, ACM, New York, 1997, pp. 189–196.

[F64] L. Fox, *An Introduction to Numerical Linear Algebra*, Oxford University Press, Oxford, UK, 1964.

[G84] J. von zur Gathen, *Parallel algorithms for algebraic problems*, SIAM J. Comput., 13 (1984), pp. 802–824.

[G86] J. von zur Gathen, *Parallel arithmetic computations: A survey*, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 233, Springer, Berlin, 1986, pp. 93–112.

[GKO95] I. Gohberg, T. Kailath, and V. Olshevsky, *Fast Gaussian elimination with partial pivoting for matrices with displacement structure*, Math. Comp., 64 (1995), pp. 1557–1576.

[GL89/96] G. H. Golub and C. F. Van Loan, *Matrix Computations,* Johns Hopkins University Press, Baltimore, MD, 1989 (2nd ed.), 1996 (3rd ed.).

[H91] S. Haykin, *Adaptive Filter Theory*, 2nd ed., Prentice-Hall, Englewood Cliffs, NJ, 1991.

[H95] G. Heinig, *Inversion of generalized Cauchy matrices and other classes of structured matrices*, in Linear Algebra for Signal Processing, IMA Vol. Math. Appl. 69, Springer, New York, 1995, pp. 95–114.

[IR82] K. Ireland and M. Rosen, *A Classical Introduction to Modern Number Theory*, Springer, Berlin, 1982.

[J92] J. Jà Jà, *An Introduction to Parallel Algorithms,* Addison-Wesley, Reading, MA, 1992.

[K74] T. Kailath, *A view of three decades of linear filtering theory*, IEEE Trans. Inform. Theory, 20 (1974), pp. 146–181.

[K87] T. Kailath, *Signal processing applications of some moment problems*, in Moments in Mathematics, Proc. Sympos. App. Math. 37, AMS, Providence, RI, 1987, pp. 71–100.

[K95] E. Kaltofen, *Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems*, Math. Comput., 64 (1995), pp. 777–806.

[KAGKA89] R. King, M. Ahmadi, R. Gorgui-Naguib, A. Kwabwe, and M. Azimi-Sadjadi, *Digital Filtering in One and Two Dimensions: Design and Applications,* Plenum Press, New York, 1989.

[KKM79] T. Kailath, S.-Y. Kung, and M. Morf, *Displacement ranks of matrices and linear equations*, J. Math. Anal. Appl., 68 (1979), pp. 395–407.

[KLM78] T. Kailath, L. Ljung, and M. Morf, *A new approach to the dertemination of Fredholm resolvents of nondisplacement kernels*, in Topics in Functional Analysis, I. Gohberg and M. Kac, eds., Academic Press, New York, 1978, pp. 169–184.

[KP91] E. Kaltofen and V. Y. Pan, *Processor efficient parallel solution of linear systems over an abstract field*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, ACM, New York, 1991, pp. 180–191.

[KP92] E. Kaltofen and V. Y. Pan, *Processor-efficient parallel solution of linear systems* II. *The positive characteristic and singular cases*, in Proceedings of 33rd Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 714–723.

[KP94] E. Kaltofen and V. Y. Pan, *Parallel solution of Toeplitz and Toeplitz-like linear systems over fields of small positive characteristic*, in Proceedings of the First International Symposium on Parallel Symbolic Computation, Lecture Notes Ser. Comput. 5, World Scientific, Singapore, 1994, pp. 225–233.

[KR90] R. Karp and V. Ramachandran, *A survey of parallel algorithms for shared memory machines*, in Handbook for Theoretical Computer Science, J. van Leeuwen, ed., North-Holland, Amsterdam, 1990, pp. 869–941.

[KS91] E. Kaltofen and B. D. Saunders, *On Wiedemann's method for solving sparse linear systems*, Proc. AAECC-9, Lecture Notes in Comput. Sci. 539, Springer, Berlin, 1991, pp. 29–38.

[KVM78] T. Kailath, A. Vieira, and M. Morf, *Inverses of Toeplitz operators, innovations, and orthogonal polynomials*, SIAM Rev., 20 (1978), pp. 106–119.

[L-AK84] H. Lev-Ari and T. Kailath, *Lattice filter parametrization and modelling of nonstationary processes*, IEEE Trans. Inform. Theory, IT-30 (1984), pp. 2–16.

[L-AKC84] H. Lev-Ari, T. Kailath, and J. Cioffi, *Least squares adaptive lattice and transversal filters; a unified geometrical theory*, IEEE Trans. Inform. Theory, IT-30 (1984), pp. 222–236.

[Le92] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays,*

Trees and Hypercubes, Morgan Kaufmann, San Mateo, CA, 1992.

[LRT79] R. J. Lipton, D. Rose, and R. E. Tarjan, *Generalized nested dissection*, SIAM J. Numer. Anal., 16 (1979), pp. 346–358.

[Ma75] J. Makhoul, *Linear prediction: A tutorial review*, Proc. IEEE, 63 (1975), pp. 561–580.

[MC79] R. T. Moenck and J. H. Carter, *Approximate algorithms to derive exact solutions to systems of linear equations*, in Symbolic and Algebraic Computation, Lecture Notes in Comput. Sci. 72, Springer, Berlin, 1979, pp. 63–73.

[Morf74] M. Morf, *Fast Algorithms for Multivariable Systems*, Ph.D. Thesis, Stanford University, Stanford, CA, 1974.

[Morf80] M. Morf, *Doubling algorithms for Toeplitz and related equations*, in Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE Computer Society, Los Alamitos, CA, 1980, pp. 954–959.

[MP98] B. Mourrain and V. Y. Pan, *Asymptotic acceleration of solving multivariate polynomial systems of equations*, in Proceedings of the ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 488–496.

[MRK88] G. L. Miller, V. Ramachandran, and E. Kaltofen, *Efficient parallel evaluation of straight-line code and arithmetic circuits*, SIAM J. Comput., 17 (1988), pp. 687–695.

[Mu81] B. R. Musicus, *Levinson and Fast Choleski Algorithms for Toeplitz and Almost Toeplitz Matrices*, Internal Report, Lab. of Electronics, M.I.T., Cambridge, MA, 1981.

[OP98] V. Olshevsky and V. Y. Pan, *A unified superfast algorithm for boundary rational tangential interpolation problem and for inversion and factorization of dense structured matrices*, in Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 192–201.

[OP99] V. Olshevsky and V. Y. Pan, *Polynomial and rational interpolation and multipoint evaluation (with structured matrices)*, in Proceedings of the 26th International Colloquium on Automata, Languages and Programming (ICALP 99), Lecture Notes in Comput. Sci. 1644, J. Wiedermann, P. van Emde Boas, and M. Nielsen, eds., Springer, Berlin, 1999, pp. 585–594.

[P85] V. Y. Pan, *Fast and efficient parallel algorithms for the exact inversion of integer matrices*, in Proceedings of the 5th Annual Conference on Foundations of Software Technology and Theoretical Compututer Science, Lecture Notes in Comput. Sci. 206, Springer-Verlag, New York, 1985, pp. 504–521.

[P87] V. Y. Pan, *Complexity of parallel matrix computations*, Theoret. Comput. Sci., 54 (1987), pp. 65–85.

[P90] V. Y. Pan, *Computations with dense structured matrices*, Math. Comp., 55 (1990), pp. 179–190.

[P91] V. Y. Pan, *Complexity of algorithms for linear systems of equations*, in Computer Algorithms for Solving Linear Algebraic Equations (The State of the Art), E. Spedicato, ed., NATO Adv. Sci. Inst. Ser. F Comput. and Systems Sci. 77, Springer, Berlin, 1991, pp. 27–56.

[P92] V. Y. Pan, *Parametrization of Newton's iteration for computations with structured matrices and applications*, Comput. Math. Appl., 24 (1992), pp. 61–75.

[P92a] V. Y. Pan, *Complexity of computations with matrices and polynomials*, SIAM Rev., 34 (1992), pp. 225–262.

[P92b] V. Y. Pan, *Parallel solution of Toeplitz-like linear systems*, J. Complexity, 8 (1992), pp. 1–21.

[P93] V. Y. Pan, *Decreasing the displacement rank of a matrix*, SIAM J. on Matrix Anal., Appl. 14 (1993), pp. 118–121.

[P93a] V. Y. Pan, *Concurrent iterative algorithm for Toeplitz-like linear systems*, IEEE Trans. Parallel and Distributed Systems, 4 (1993), pp. 592–600.

[P93b] V. Y. Pan, *Parallel solution of sparse linear and path systems*, in Synthesis of Parallel Algorithms, J.H. Reif, ed., Morgan Kaufmann, San Mateo, CA, 1993, pp. 621–678.

[P95] V. Y. Pan, *Optimal (up to polylog factors) sequential and parallel algorithms for approximating complex polynomial zeros*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 741–750.

[P96] V. Y. Pan, *A new approach to parallel computation of polynomial GCD and to*

related parallel computations over abstract fields, in Proceedings of the Seventh Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, PA, 1996, pp. 518–527.

[P96a] V. Y. Pan, *Optimal and nearly optimal algorithms for approximating polynomial zeros*, Comput. Math. Appl., 31 (1996), pp. 97–138.

[P96b] V. Y. Pan, *Parallel computation of polynomial GCD and some related parallel computations over abstract fields*, Theoret. Comput. Sci., 162 (1996), pp. 173–223.

[P96c] V. Y. Pan, *Effective parallel computations with Toeplitz and Toeplitz-like matrices filled with integers*, in The Mathematics of Numerical Analysis (Park City, Utah, 1995), Lectures in Appl. Math. 32, J. Renegar, M. Shub, and S. Smale, eds., Amer. Math. Soc., Providence, RI, 1996, pp. 591–641.

[P97] V. Y. Pan, *Solving a polynomial equation: Some history and recent progress*, SIAM Rev., 39 (1997), pp. 187–220.

[P00] V. Y. Pan, *Nearly optimal computations with structured matrices*, in Proceedings of the 11th Annual ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 2000, pp. 953–962.

[P00a] V. Y. Pan, *Matrix structure, polynomial arithmetic, and erasure-resilient encoding/decoding*, to appear in Proceedings of the ACM International Symposium on Symbolic and Algebraic Computation, ACM, New York, 2000.

[PBRZ99] V. Y. Pan, S. Branham, R. Rosholt, and A. Zheng, *Newton's iteration for structured matrices*, in Fast Reliable Algorithms for Matrices with Structure, SIAM, Philadelphia, PA, 1999, pp. 189–210.

[PP95] V. Y. Pan, F. P. Preparata, *Work-preserving speed-up of parallel matrix computations*, SIAM J. Comput., 24 (1995), pp. 811–821.

[PR89] V. Y. Pan and J. Reif, *Fast and efficient parallel solution of dense linear systems*, Comput. Math. Appl., 17 (1989), pp. 1481–1491.

[PR91] V. Y. Pan and J. Reif, *The parallel computation of the minimum cost paths in graphs by stream contraction*, Inform. Process. Lett., 40 (1991), pp. 79–83.

[PR93] V. Y. Pan and J. Reif, *Fast and efficient parallel solution of sparse linear systems*, SIAM J. Comput., 22 (1993), pp. 1227–1250.

[PSLT93] V. Y. Pan, A. Sadikou, E. Landowne, and O. Tiga, *A new approach to fast polynomial interpolation and multipoint evaluation*, Comput. Math. Appl., 25 (1993), pp. 25–30.

[PZHY97] V. Y. Pan, A. Zheng, X. Huang, and Y. Yu, *Fast multipoints polynomial evaluation and interpolartion via computations with structured matrices*, Ann. Numer. Math., 4 (1997), pp. 483–510.

[Q94] M.J. Quinn, *Parallel Computing: Theory and Practice,* McGraw-Hill, New York, 1994.

[R93] J. Reif, *An $O(n \log^3 n)$ algorithm for the real root problem*, in Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1993, pp. 626–635.

[R95] J. Reif, *Work efficient parallel solution of Toeplitz systems and polynomial GCD*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, ACM, New York, 1995, pp. 751–761.

[RT90] J. Reif and S.R. Tate, *Optimal size integer division circuits*, SIAM J. Comput., 19 (1990), pp. 912–924.

[St69] V. Strassen, *Gaussian elimination is not optimal*, Numer. Math., 13 (1969), pp. 354–356.

[Ste94] W. F. Stewart, *Introduction to the Numerical Solution of Markov Chains,* Princeton University Press, Princeton, NJ, 1994.

[VSBR83] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, *Fast parallel computation of polynomials using few processors*, SIAM J. Comput., 12 (1983), pp. 641–644.

[W65] J. H. Wilkinson, *The Algebraic Eigenvalue Problem,* Clarendon Press, Oxford, UK, 1965.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1126–1155

FROM STATIC TO DYNAMIC ROUTING: EFFICIENT TRANSFORMATIONS OF STORE-AND-FORWARD PROTOCOLS*

CHRISTIAN SCHEIDELER† AND BERTHOLD VÖCKING‡

Abstract. We investigate how static store-and-forward routing algorithms can be transformed into efficient dynamic algorithms, that is, how algorithms that have been designed for the case that all packets are injected at the same time can be adapted to more realistic scenarios in which packets are continuously injected into the network. Besides describing specific transformations for well-known static routing algorithms, we present a black box transformation scheme applicable to every static, oblivious routing algorithm. We analyze the performance of our protocols under a stochastic and an adversarial model of packet injections.

One result of our specific transformations is the first dynamic routing algorithm for leveled networks that is stable for arbitrary admissible injection rates and that works with packet buffers of size depending solely on the injection rate and the node degree, but not on the size of the network. Furthermore, we prove strong delay bounds for the packets. Our results imply, for example, that a throughput of 99% can be achieved on an n-input butterfly network with buffers of constant size while each packet is delivered in time $O(\log n)$, with high probability.

Our black box transformation ensures that if the static algorithm is pure (i.e., no extra packets apart from the original packets are routed), its dynamic variant is stable up to a maximum possible injection rate. Furthermore, in the stochastic model, the routing time of a packet depends on local parameters such as the length of its routing path, rather than on the maximum possible path length, even if the static algorithm chosen for the transformation does not provide this locality feature and is not pure. In the adversarial model, the delay bound of the packets is closely related to the time bound given for the static algorithm.

Key words. communication networks, store-and-forward routing, packet scheduling

AMS subject classifications. 68Q22, 68Q25, 68M20, 68R10, 90B35

PII. S0097539799353431

1. Introduction.

Many static routing protocols have been developed in recent years (see, e.g., [6, 8, 9, 10, 12, 13]). These protocols aim to route as quickly as possible some initially given set of packets along predetermined paths in a network. In practice, however, networks are rarely used in this static fashion, but packets are injected dynamically into the network. Since much less is known in the area of dynamic routing (see, e.g., [5, 15, 17]) than in the area of static routing, it would be highly desirable to transfer the results gathered for static routing to dynamic routing.

In this paper we present transformations for *oblivious* algorithms; i.e., the path of a packet is already fixed when the packet is injected into the system. We investigate how static, oblivious routing algorithms can be transformed into dynamic routing algorithms that are stable and efficient under a stochastic or adversarial model of packet injections. In particular, we will show that the ghost packet protocol [8, 14] and the growing rank protocol [10, 11] can be transformed into dynamic routing protocols

*Received by the editors March 22, 1999; accepted for publication (in revised form) October 24, 1999; published electronically September 7, 2000. A preliminary version of this paper appears in *Proceedings of the* 31*st ACM Symposium on the Theory of Computing*, Atlanta, GA, 1999, pp. 215–224.

http://www.siam.org/journals/sicomp/30-4/35343.html

†Department of Mathematics and Computer Science, and Heinz Nixdorf Institute, Paderborn University, Paderborn, Germany (chrsch@uni-paderborn.de). The research of this author was supported in part by the DFG-Sonderforschungsbereich 376 and the EU ESPRIT Long Term Research Project 20244 (ALCOM-IT).

‡Max-Planck-Institut für Informatik, Saarbrücken, Germany (voecking@mpi-sb.mpg.de).

that are stable up to a maximum possible injection rate. Furthermore, we will present a simple and elegant scheme that transforms almost any static protocol into an efficient dynamic protocol that is also stable up to a maximum possible injection rate. Besides showing the stability of these protocols, we will prove bounds on the routing time of the packets. For the protocols derived by the black box transformation we further prove that they recover quickly from any worst-case scenario; that is, packets generated a certain amount of time after a bad event are not influenced by this event anymore.

1.1. Models and problems. We consider arbitrary network topologies modeled as undirected graphs of n nodes. The nodes represent switches, and the edges represent bidirectional communication links, unless otherwise stated, with buffers for incoming packets on either side. These buffers are called *edge buffers*, and bounds on the buffer size always refer to the maximum number of packets that these buffers can hold. Additionally, every node contains an *injection buffer* in which the initial packets, in the case of static routing, or the newly injected packets, in the case of dynamic routing, are stored. Routing is performed in a synchronous "store-and-forward" manner; that is, in every step, each edge can be crossed by at most one packet in each direction. (For simplicity, we assume that time proceeds in discrete time steps.) Once a packet reaches its destination, it is discarded.

We present routing protocols in which the nodes *locally* decide which packets to move forward in each step; i.e., a decision only depends on the routing information carried by the packets and on the local history of the execution. These algorithms are called *local control* or *on-line* algorithms. In general, a packet routing scheme consists of two (not necessarily independent) parts: the first part is responsible for selecting a path for each packet, and the second part is responsible for scheduling the packets along their chosen paths. We assume that some suitable strategy for the path selection is given. Hence, in the following we concentrate only on the question of how to schedule the packets along their fixed paths. We use the following models.

1.1.1. Static packet routing. Here we assume that a fixed collection of paths is given with *congestion* C and *dilation* D; that is, C denotes the maximum number of paths crossing an edge, and D denotes the maximum length of a path. Along each of these paths a packet has to be sent. Further, let M denote the *complexity* of the routing problem; i.e., M is defined to be the maximum of the number of edges, the number of paths, and the dilation. Let us give some examples of known results on the *routing time* for static packet routing, i.e., the time needed to deliver all packets.

- $C \cdot D$: trivial upper bound for any greedy protocol in case of unlimited buffers, i.e., protocols in which a packet is only delayed because other packets move along the next edge on the packet's routing path;
- $(1+\kappa) \cdot C + O(D \cdot \log M)$, w.h.p.[1], for any constant $\kappa > 0$: upper bound for arbitrary paths in arbitrary networks with unbounded buffers [9];
- $O(C + D + \log M)$, w.h.p.: upper bound in leveled networks with bounded buffers of constant size, where D is the depth of the network [14, 8], and upper bound for routing along shortest paths in arbitrary networks with unbounded buffers [10, 11];
- $(1+\kappa) \cdot C + (\log^* M)^{O(\log^* M)} D + O((\log M)^6)$, w.h.p., for any constant $\kappa > 0$: upper bound for routing along *simple paths*, i.e., paths without cycles, in

[1] Throughout this paper, the term "w.h.p." means "with high probability," i.e., with probability at least $1 - M^{-\alpha}$, where $\alpha > 0$ is an arbitrary constant term and M denotes the complexity of the routing problem.

arbitrary networks with unbounded buffers [13];
- $O(C+D+\log^{1+\kappa} M)$, w.h.p., for any constant $\kappa > 0$: upper bound for routing along simple paths in arbitrary networks with unbounded buffers [12].

We will come back to some of these results when using our black box transformations.

1.1.2. Dynamic packet routing. The most commonly used injection models in the dynamic setting are the *stochastic* and the *adversarial* injection model.

The stochastic model. Here the packets are injected by a set of *generators*, each of them mapped to one of the nodes in the network. We allow any relationship between the number of generators and the number of nodes in the network. Furthermore, we place no restrictions on the distribution of the generators among the nodes. That is, one node could have several generators, whereas another node may have none. So a generator may represent a user thread or process, whereas a node may represent a processor. In each time step, each generator g placed on a node v injects a packet with some probability p_g. This probability is called the *injection rate* of g. For each packet, the generator randomly selects a destination and a routing path from v to this destination according to an arbitrary, fixed probability distribution. We assume that each generator is operating independently of other generators, and the injection of a packet and its routing path is independent of injections in previous time steps. Note that we do not demand that the destinations are chosen uniformly from the set of all nodes, or that packets with the same source and destination node follow the same routing path. Finally, we define the *(overall) injection rate*, which is denoted by λ. Define λ_e to be the expected number of messages generated in a time step that contain the edge e in their routing paths. Then λ is defined to be the maximum λ_e over all edges.

The adversarial model. An adversary is allowed to demand network bandwidth up to a prescribed injection rate. For any $w, \lambda > 0$, an adversary is called a *bounded adversary of rate* (w, λ) if for all edges e and all time intervals I of length w, it injects no more than $\lambda \cdot w$ packets during I that contain edge e in their routing paths. As in the stochastic model, λ is defined to be the *injection rate*. (We use the adversarial model as defined by Andrews et al. [1] rather than the original model introduced by Borodin et al. [3] because this model avoids calculating with floors and ceilings. Apart from minor changes in constants, however, all our results hold in the original model of Borodin et al. too.)

For both injection models, a protocol is called *stable* for a given injection rate λ if the number of packets stored in injection or edge buffers does not grow unboundedly with time. We are interested in protocols that are stable for high injection rates. Of course, since an edge can transport at most one packet per step, λ can be at most 1. Our aim is to construct algorithms that are stable under injection rates that are close to 1. Additionally, we are interested in short delays for the packets; i.e., we aim to minimize the time from injection to service for each packet.

Apart from the stability and the routing time, we will consider another property of dynamic routing protocols, the recovery from worst-case scenarios. Although our bounds on the routing time guarantee that bad configurations are very unlikely, they eventually occur from time to time when the routing protocol runs for an infinite number of time steps. Let a *worst-case scenario* denote an arbitrarily bad configuration of the network. Then the *recovery time* with regard to some property $\mathcal{P}$ of the routing protocol is defined as the time that has to pass by after the occurrence of a worst-case scenario until $\mathcal{P}$ holds again. (In our case, we are interested in properties such as the expected routing time of a packet and time bounds that hold w.h.p.)

As in the static model we define the complexity of a dynamic routing problem to be a value capturing all relevant parameters. In particular, the *complexity* is defined to be the maximum of the number of edges, the number of generators, the maximal possible length of a routing path, and $1/(1-\lambda)$. (The number of generators in the adversarial model is defined to be w times the number of edges, which corresponds to the maximum number of packets that can be injected in a single step.)

In the following sections, we will prove results for both the stochastic and the adversarial injection model.

1.2. Previous results. In the last two years a new model called *adversarial queuing theory* emerged. This approach was introduced by Borodin et al. in [3]. Most research in this model focuses on the stability of routing protocols and networks. For example, Borodin et al. [3] show several stability results for greedy protocols on directed acyclic graphs (DAGs) and directed cycles. Andrews et al. [1] extend their results by showing that there exist simple greedy protocols (such as longest-in-system, shortest-in-system, and farthest-to-go protocols) that are stable against any adversary for all networks. However, the delay of the packets and the number of packets stored in a queue might get exponential in the length of the longest path.

Furthermore, Andrews et al. [1] present a transformation of the static protocol presented in [9] into a dynamic protocol that is stable for any injection rate < 1 and fulfills the following constraint on the buffer size: For any fixed time step t, at most $(D \cdot \log m)^k$ packets are stored in any queue at time t, w.h.p., where D denotes the longest routing path, m the number of edges, and k is a suitable constant. Note that this result implies that the delay of the packets is also bounded by $(D \cdot \log m)^k$, w.h.p. However, as the bound on the buffer size does not hold deterministically, any buffer of fixed size will overflow eventually.

Rabani and Tardos [13] present a transformation scheme which yields much better routing times. Assuming there is a static algorithm that delivers all packets in $(1+\kappa)C + g(M)D + f(M)$ steps for some constant $\kappa > 0$, their tansformation yields a dynamic algorithm that delivers each packet to its destination, w.h.p., in $O(w + g(N)D + f(N) + \log N)$ against an adversary of rate $(w, \Theta(\kappa))$, where M and N denote the complexity of the static and dynamic routing problems, respectively. The stability of the dynamic algorithms, however, is not shown. In fact, although most packets will be delivered fast, some packets will never reach their destination and queues will grow to infinity assuming either the stochastic model or the adversarial model in combination with a randomized, static algorithm.

Broder, Frieze, and Upfal [4] introduce a general approach to dynamic packet routing with bounded buffers in the stochastic and adversarial models. They show sufficient conditions for the stability of dynamic packet routing algorithms and investigate how some well-known static routing protocols for the butterfly network can be transformed into dynamic algorithms that fulfill these conditions. In particular, they present a dynamic routing algorithm for the butterfly that is stable for a small constant injection rate, and they show that the expected routing time for each packet is $O(\log n)$, with n denoting the number of nodes on a level.

Andrews et al. [2] investigate another, more restrictive dynamic routing model. In contrast to the stochastic and adversarial models, the packets are injected regularly in "sessions." For each session i, packets are injected at a rate r_i to follow a fixed path of length d_i. They describe a schedule that delivers each packet in a time depending only on local parameters; that is, each packet reaches its destination in time $O(d_i + 1/r_i)$, which is worst-case optimal. We will see that similar local properties, i.e., the routing

time depends on d_i, can be achieved also in the stochastic model.

1.3. New results. In this paper, we present specific transformations of well-known routing protocols and introduce a powerful black box transformation scheme applicable to every static, oblivious routing protocol.

In the following, N denotes the complexity of the routing problem (as defined in section 1.1), D denotes the maximum length of a routing path, and $\epsilon = 1 - \lambda$, where λ denotes the injection rate. Further, we define $\epsilon^* = a \cdot \epsilon^b$, for suitable constants $a, b > 0$. For simplicity, we state our results only for networks of constant degree. For more detailed results the reader is referred to the following sections.

In this paper, we give three specific transformations of well-known routing protocols.

- In section 2, we present a dynamic variant of the ghost packet protocol [8, 14] for leveled networks that is stable for any $\lambda < 1$ in the stochastic model and $\lambda \leq 1$ in the adversarial model, given a sufficiently large but fixed buffer size of $1/\epsilon^*$ in the stochastic model and $2\lambda \cdot w + 2$ in the adversarial model. In the stochastic model, each individual packet is delivered in expected time L/ϵ^*, and in time $(L + \log N)/\epsilon^*$, w.h.p., where L denotes the depth of the network. In the adversarial model, each packet reaches its destination in at most $L + \lambda \cdot w \cdot L - 1$ time steps.
  For example, the tuned ghost packet protocol achieves a throughput of $1 - \epsilon$, for any $\epsilon > 0$, on an n-input butterfly network with buffers of size $1/\epsilon^*$ if we place two generators on each node of level 0, each of which injects packets that are sent to randomly selected nodes of level $\log n$, using a rate of $\lambda = 1 - \epsilon$. Furthermore, the algorithm delivers each individual packet in time $\log n/\epsilon^*$, w.h.p.
  Previous results on routing with bounded buffers in leveled networks obtain stability only for constant injection rates $\lambda \ll 1$ [4, 16] (stochastic model) or require buffers whose size is exponential in the depth of the network [1] (adversarial model).
- In section 3, we present a dynamic routing protocol for arbitrary networks that is stable for any injection rate $\lambda < 1$, assuming buffers of fixed size D/ϵ^*. We prove an expected routing time of $(D^2+w)/\epsilon^*$, and $(D^2+D\cdot\log N+w)/\epsilon^*$, w.h.p., for every individual packet. These bounds hold both for the adversarial model and for the stochastic model (with $w = 0$).
  To the best of our knowledge, this is the first protocol that is stable for buffers of small fixed size under any injection rate $\lambda < 1$. Previous results in the stochastic model with bounded buffers require $\lambda \ll 1$ and besides assume that packets can be dropped and reinjected in later time steps [16]. Previous results in the adversarial model require buffers whose size is exponential (or polynomial, w.h.p.) in D [1]. Note that a bound on the buffer size that does not hold with certainty leaves open the question of what to do in the rare case of a buffer overflow (e.g., dropping or blocking incoming packets), and hence does not guarantee stability for networks with a fixed buffer size.
- In section 4, we describe a dynamic variant of the growing rank protocol [10] for shortest paths in arbitrary networks. The dynamic protocol is stable for any $\lambda < 1$ if unbounded buffers are given. In the stochastic model, each individual packet p with a routing path of length d_p is delivered in time d_p/ϵ^*, expected, and $(d_p + \log N)/\epsilon^*$, w.h.p. In the adversarial model, the d_p in the time bound has to be replaced by $D + w$.

Previously, similar results have been shown only for $\lambda < 1/\mathrm{e}$ in the stochastic model [16].

Furthermore, in section 5, we present a powerful black box transformation scheme that is applicable to every static, oblivious routing algorithm in networks with unlimited buffers. Basically, we combine the ideas of Rabani and Tardos [13] for the fast delivery of packets with the universal stability of the shortest-in-system protocol originally shown by Andrews et al. [1] for the adversarial model. The major problem that we solve is merging these two approaches so that we obtain dynamic protocols that are stable up to some injection rate depending on the static protocol without any significant slowdown due to the inefficiency of the shortest-in-system protocol.

Let $\mathcal{S}$ denote any set of paths, e.g., the set of all simple or all shortest paths in the network. Suppose we are given a static routing algorithm that routes all packets in $\gamma \cdot C + \delta \cdot D + O(\log^\alpha M)$ steps, w.h.p. (or even with certainty), for any collection of paths or subpaths in $\mathcal{S}$ with congestion C, dilation D, and complexity M. Assume that in the dynamic setting only paths in $\mathcal{S}$ are allowed to be generated. Then our black box transformation yields a dynamic variant of this protocol with the following properties. (In the following overview, we describe only the results for the case that γ and δ do not depend on C or D and $\alpha \geq 1$ is a constant. Similar results will be shown for other choices of α, γ, and δ.)

- If the given static protocol is *pure* (i.e., no control messages or copies of packets are allowed), the dynamic algorithm is stable for any injection rate $\lambda < 1$. Otherwise, it is stable for any injection rate $\lambda < 1/\gamma$.
- If $\lambda < 1/\gamma$, then the algorithm guarantees that any packet p that has to travel a distance of d is delivered in time $O(\delta \cdot d + \log^\alpha N)$, w.h.p., in the stochastic injection model, and in time $O(\delta \cdot D + w + \log^\alpha N)$, w.h.p., in the adversarial injection model.
- The algorithm recovers from any worst-case scenario in $O(\delta \cdot D + w + \log^\alpha N)$ time steps (with $w = 0$ in the stochastic model).

The bound on the routing time implies that it might be important for static routing protocols to know the exact factor γ in front of the C since this can be decisive for the performance of their dynamic counterparts. Interestingly, in the stochastic injection model, the dynamic variant is able to exploit locality, whereas the static algorithm does not need to provide this feature. For example, the transformation of a well-known static routing algorithm (see, e.g., [9]) that delivers all packets in time $(1+\kappa) \cdot C + O(D \cdot \log M)$, w.h.p., for any constant $\kappa > 0$ yields a dynamic algorithm that delivers each packet p in time $O(d \cdot \log N)$, w.h.p., for any constant injection rate $\lambda < 1/(1+\kappa)$, where d is the length of p's path.

1.4. Tools. We will frequently apply the following Chernoff bounds.

LEMMA 1.1 (Chernoff). *Let $X_1, \ldots, X_n$ be n independent random variables with $X_i \in \{0,1\}$ for all $i \in \{1,\ldots,n\}$. Furthermore, let $X = \sum_{i=1}^n X_i$ and $\mu \geq \mathrm{E}[X]$. Then it holds for all $\epsilon \geq 0$ that*

$$\Pr[X \geq (1+\epsilon)\mu] \leq \left(\frac{\mathrm{e}^\epsilon}{(1+\epsilon)^{1+\epsilon}}\right)^\mu.$$

This can be simplified to

$$\Pr[X \geq (1+\epsilon)\mu] \leq \begin{cases} \mathrm{e}^{-\epsilon^2\mu/3} & \text{if } \epsilon \in [0,1], \\ \mathrm{e}^{-\epsilon\mu/3} & \text{otherwise.} \end{cases}$$

2. Routing in leveled networks with bounded buffers. In this section, we consider the problem of routing packets in a leveled network with bounded buffers. In a *leveled network*, the nodes can be partitioned into levels $0, \ldots, L$ such that each link in the network leads from some node at level i to some node at level $i + 1$ for $0 \leq i \leq L - 1$. L is called the *depth of the network*.

The routing proceeds in discrete time steps, starting with step 0. In each step, each link can forward at most one packet. The links are assumed to be directed; that is, packets can cross them only in the direction leading to the higher level. Packet injections and arrivals are assumed to happen at the beginning of a time step, so that a packet may leave a node at the end of the time step in which it is injected or arrives at the node. The packets' routing paths may start on any level $k \geq 0$ and end on any level k' with $k < k' \leq L$. Each node has a buffer for each of its incoming edges and a buffer for newly injected packets. Each of the edge buffers has space for storing at most q packets.

Static batches of packets can be routed efficiently on leveled networks by a protocol known as Ranade's, or the ghost packet, protocol [7, 8, 14]. The disadvantage of the static ghost packet protocol is that each node is allowed to forward only one data packet at each time step, rather than forwarding data packets along all outgoing edges in parallel. All edges that are not passed by a data packet in a step are used to exchange control packets that are called *ghost packets*. As a consequence, most of the transmitted packets are ghost packets, which shows that a simple transformation of the static ghost packet protocol into a dynamic protocol cannot yield stability for injection rates close to 1. In order to achieve stability for any injection rate $\lambda < 1$, we introduce a tuned variant of the ghost packet protocol that uses only a very limited number of ghost packets.

The tuned ghost packet protocol. The packets are assigned ranks in order to decide which packet is preferred in case of contention. For each packet p, let $\text{birth}(p)$ denote the time step at which p was injected. The rank of p is set to $\text{birth}(p)$ plus some small value x from the interval $[0, \kappa)$ for some $\kappa < 1$, where x is chosen such that each packet has its own unique rank (e.g., based on the identification number of the generator that injected the packet). Packets with smaller ranks, i.e., older packets, are always preferred against packets with higher rank, i.e., younger packets. As in the static ghost packet protocol, special ghost packets help the algorithm to maintain the following invariant: A packet is routed along an edge only after all the other packets with lower ranks that must pass through the edge have done so. The nodes on level k start working in step k for $0 \leq k \leq L$. In order to give time for initializing the network, we assume that packet injections on level k do not start before time step k. Figure 1 describes the rules for contention resolution in detail.

Ghost packets are discarded as soon as they are delayed in a step. Thus, they never block the buffer for following packets. The role of the ghost packets is to slow down packets that are too fast in order to avoid that a relatively old packet is blocked because younger packets occupy the slots in the next buffer. Note that each outgoing link on level k transmits one packet in each time step $t \geq k$: either a ghost or a real packet. This mechanism ensures that each link transmits packets and ghost packets in the order of increasing rank. (Obviously, this property holds for the links on level 0. For higher levels the property follows by induction.) We will see that this property is crucial for the analysis. Analyzing the performance of a variant of the protocol described above that does not use ghost packets is an interesting open problem, even in the static case.

The following algorithm is executed for each outgoing link e of a node v on level k in each time step $t \geq k$. Each edge buffer can hold up to q packets.

- Let r denote the minimum rank of a packet that is stored in one of v's buffers and aims to pass edge e. If there is no such packet, then $r = \infty$.
- Let g denote the minimum over all ranks of packets or ghost packets that arrived on v at the beginning of step t. If there is no such packet (as v is a node without incoming edges, e.g., on level 0), then g is set to $t + \kappa$.
- If $r < g$, then

  if the buffer of e contains less than q packets at the beginning of step t, then

    forward the (unique) packet with rank r along e;

  else

    send a ghost packet with rank r along e;

  else

    send a ghost packet with rank g along e.

FIG. 1. *Contention resolution in the tuned ghost packet protocol.*

The following analysis shows that the tuned ghost packet protocol is stable for any injection rate $\lambda \leq 1$ in the adversarial model, and $\lambda < 1$ in the stochastic model, provided that the edge buffer size is sufficiently large.

THEOREM 2.1. *Let L denote the depth of the network, Δ the maximum node degree, and q the size of the edge buffers.*

- *Suppose the packets are injected according to the adversarial model with injection rate (w, λ) for any $w \leq (q-2)/(2\lambda)$ and $0 \leq \lambda \leq 1$. Then the tuned ghost packet protocol is stable, and each packet reaches its destination in at most $L + \lambda \cdot w \cdot L$ time steps.*
- *Suppose the packets are injected according to the stochastic model with injection rate $\lambda \leq 1 - \epsilon$ for a suitably chosen $\epsilon = \Theta(\log(q\Delta)/q)$. Then the tuned ghost packet protocol is stable, and the routing time for each individual packet is $O(L \cdot \log(\Delta/\epsilon)/\epsilon + \log(1/\epsilon)/\epsilon^2)$, expected, and $O(L \cdot \log(\Delta/\epsilon)/\epsilon + (\log N)/\epsilon^2)$, w.h.p., where the probability is with respect to the stochastic packet injections.*

Proof. We use a "delay sequence argument" to analyze the tuned ghost packet protocol. Our analysis is similar to the one for the static ghost packet protocol given in [8]. A delay sequence witnesses that a packet needs many time steps to reach its destination. For the adversarial model, we will show that a delay sequence witnessing a long routing time does not exist, so that every packet reaches its destination within

the time bound given in the theorem. For the stochastic model, we will show that "large" delay sequences are very unlikely so that each packet needs only limited time, w.h.p., to reach its destination.

The ghost packet protocol uses fractional ranks. The only reason for the fractional additive is to define a total order among all packets such that a packet p or a ghost packet corresponding to p (i.e., a ghost packet that has the same rank as p) that delays a packet p' in a step cannot be delayed by packet p' or a ghost packet corresponding to p' in another time step. In the following, however, we mainly use *integral ranks*, i.e., the integral values of the fractional ranks, which, in the case of the ghost packet protocol, are equal to the birth date of the corresponding packet.

DEFINITION 2.2 ((p, s, ℓ, r)-delay sequence). *Let p denote a packet, and let $s, \ell, r \geq 1$ denote integers. Then a (p, s, ℓ, r)-delay sequence consists of the following:*

- *A path of length ℓ starting at the destination node of packet p. This path is called the* delay path. *Let $v_0, \ldots, v_\ell$ denote the nodes on the delay path. The delay path may include edges in both directions and, hence, follows a course going up and down the levels of the network.*
- *ℓ* delay edges *$e_1, \ldots, e_\ell$ such that e_i is incident to v_i for $1 \leq i \leq \ell$. These edges are not necessarily included in the delay path.*
- *ℓ nonempty* intervals of integral ranks *$r_1, \ldots, r_\ell$ such that $\sum_{i=1}^{\ell} |r_i| = r$, where $|r_i|$ denotes the length of interval r_i. The maximum integral rank in r_1 is equal to the birth date of packet p, and, for $2 \leq i \leq \ell$, the maximum integral rank in r_i is equal to the minimum in r_{i-1}.*

A (p, s, ℓ, r)-delay sequence is called active *if the adversary or the stochastic generators inject s packets $p_1, \ldots, p_s$ (different from p) such that, for every $1 \leq i \leq s$, packet p_i has an integral rank in r_j and its routing path includes edge e_j for some $1 \leq j \leq \ell$. These packets are called* delay packets.

The following lemma shows that a long routing time of a packet p_0 is always accompanied by an active (p_0, s, ℓ, r)-delay sequence with relatively large s and small ℓ and r.

LEMMA 2.3. *Suppose a packet p_0 takes $L+T$ or more steps to reach its destination for any $T > 0$. Then there is an active (p_0, s, ℓ, r)-delay sequence with $s \geq T + r + (\frac{q}{2} - 2) \cdot \ell - (\frac{q}{2} - 1) \cdot L'$, where L' denotes the difference between the level of the first node, v_0, and the level of the last node, v_s, of the delay path.*

Proof. We will construct a sequence of packets or ghost packets $p_0, \ldots, p_{s'}$ and nodes $v'_0, v'_1, \ldots, v'_{s'}$ such that v'_0 denotes the destination of p_0, and packet p_i or a ghost packet corresponding to this packet delays packet p_{i-1} at node v'_i for $1 \leq i \leq s'$. (Ultimately, s will be set either to s' or to $s' - 1$.) There are two reasons why a packet may be delayed: It is delayed either by a packet or ghost packet with lower rank that wants to traverse the same link, or it is delayed by q other packets with lower ranks occupying the next edge buffer on its path. The first kind of delay is called *m-delay*; the second kind of delay is called *f-delay*.

The active delay sequence is constructed incrementally. Suppose we have already fixed the packets $p_0, \ldots, p_{j-1}$ and nodes $v'_1, \ldots, v'_{j-1}$. Starting from the time step in which p_{j-1} delayed p_{j-2} on node v'_{j-1} or, if $j = 1$, the time step in which $p_{j-1} = p_0$ reached its destination node v'_0, we follow the course of the packet p_{j-1} backwards in time. If p_{j-1} is a ghost packet whose generation was caused by the arrival of another packet, then we identify p_{j-1} with that packet and continue the trace. We stop following when we reach a node on which p_{j-1} either was delayed, was injected

as a nonghost packet, or was injected as a ghost packet on a node without incoming edges. The node on which this event happened is called v'_j. If we stop because of an m-delay, then the packet that caused the delay is called p_j. If we stop because of an f-delay, then the q packets that occupy the corresponding buffer are called $p_j, \dots, p_{j+q-1}$, in decreasing order of ranks. Moreover we set $v'_{j+1}, \dots, v'_{j+q-1} = v'_j$. In both of these cases we can continue our construction. In the other cases, however, our construction ends with a packet p_j that was injected at node v'_j, and we define $s' = j$.

The path from the destination of p_0 to the source of $p_{s'}$ recorded by this process is called the *delay path*. The nodes on this path are defined to be $v_0, \dots, v_\ell$. Note that these nodes are not necessarily identical to $v'_0, \dots, v'_{s'}$, but $\{v'_0, \dots, v'_{s'}\} \subseteq \{v_0, \dots, v_\ell\}$. The edges on which the recorded delays of the packets $p_0, \dots, p_{s'-1}$ take place are defined to be the delay edges. We have to show that the number of these edges is at most ℓ. (Note that the delay edges are not necessarily included in the delay path. Consider, for example, the following scenario. Suppose packet p_j is delayed in step t on level k by an f-delay caused by the packets $p_{j+1}, \dots, p_{j+q}$ stored in a buffer on level $k+1$. Suppose the next event recorded by the delay sequence is an m-delay of packet p_{j+q} caused by packet p_{j+q+1} moving along an edge e to level $k+2$ in step $t-1$, and suppose this packet arrived on level $k+1$ coming from level k in step $t-2$. Then the delay path goes from level k to level $k+1$ and then back to level k, and hence skips the delay edge e.)

Although not every delay edge is included in the delay path, each of these edges is incident to a node of the delay path, and the number of different delay edges is at most ℓ, which can be shown as follows. All delay events in a sequence of consecutive m-delays recorded by the construction above in consecutive time steps, i.e., not separated by packet movements or f-delays, take place at the same edge. Further, an f-delay following immediately after a sequence of m-delays takes place at the same edge, too. (Note that these properties are not given for the original ghost packet protocol.) Hence, considering the incremental construction of the delay sequence, the delay edge changes only in those incremental steps in which the delay path is increased by at least one node. Consequently, every node v_i of the delay path can be assigned one delay edge, which is called e_i for $1 \le i \le \ell$.

The ghost packet protocol ensures that the ranks in the delay sequence are decreasing. In particular, the ranks of the packets traversing edge e_i are larger than the ranks of the packets traversing edge e_{i+1} for $1 \le i \le \ell - 1$. Hence, we can define consecutive intervals of integral ranks $r_1, \dots, r_\ell$ in such a way that every neighboring pair of intervals has one integral rank in common (i.e., overlap by one) and the packets that are delayed at edge e_i have an integral rank in r_i for $1 \le i \le \ell$. We define $r' = \sum |r_i| = \text{birth}(p_0) - \text{birth}(p_{s'}) + \ell$. (Ultimately, r will be set either to r' or to $r' - 1$.)

Next we investigate the relationship between the parameters of the delay sequence. Define t to be the number of time steps covered by the construction, i.e., the time from the birth of the packet $p_{s'}$ to the arrival of the packet p_0 at its destination. Then

$$\begin{aligned} t &\ge \text{birth}(p_0) + L + T - \text{birth}(p_{s'}) \\ &= L + T + r' - \ell. \end{aligned} \tag{1}$$

The delay path is enlarged by one node in each of those time steps that do not represent an m-delay. Let m denote the number of m-delays. Then we can conclude

that $t = m + \ell$ or $m = t - \ell$. Applying the bound in (1) to this equation yields

$$m \geq L + T + r' - 2\ell. \tag{2}$$

Let f be the number of f-delays. Since each f-delay enlarges the delay path by two edges, we have $\ell = L' + 2 \cdot f$, where L' denotes the difference between the level of v_0 and the level of v_s. Hence,

$$f = \frac{\ell - L'}{2}. \tag{3}$$

Further, each m-delay adds one packet to the active delay sequence, and each f-delay adds q packets. As a consequence,

$$\begin{aligned} s' &= m + q \cdot f \\ &\overset{(2,3)}{\geq} L + T + r' - 2\ell + \tfrac{q}{2} \cdot \ell - \tfrac{q}{2} \cdot L' \\ &\geq T + r' + \left(\tfrac{q}{2} - 2\right) \cdot \ell - \left(\tfrac{q}{2} - 1\right) \cdot L', \end{aligned}$$

where the last estimation holds because $L' \leq L$.

Finally, we fix the parameters s and r. For $0 \leq i \leq s' - 1$, every packet p_i is delayed by packet p_{i+1}. As ghost packets are never delayed, $p_1, \ldots, p_{s'-1}$ must be nonghost packets. The definition of s and r depends on whether or not $p_{s'}$ is a ghost packet. If it is not, then we set $s = s'$ and $r = r'$ so that $p_1, \ldots, p_{s'}$ are the packets of the active (p_0, s, ℓ, r)-delay sequence. Otherwise, if $p_{s'}$ is a ghost packet, then we set $s = s' - 1$. In this case, the packets $p_1, \ldots, p_{s'-1}$ are the delay packets. Since $p_{s'}$ is a ghost packet with rank $\text{birth}(p_{s'}) + \kappa$ and $p_{s'}$ is preferred against $p_s = p_{s'-1}$, we have $\text{birth}(p_s) \geq \text{birth}(p_{s'}) + 1$. (Recall that the increment in the rank for ghost packets, i.e., κ, is larger than the increment in the rank for other packets, i.e., some $x \in [0, \kappa)$.) As a consequence, we can set $r = r' - 1$, and the constructed delay sequence fulfills all desired requirements. Hence, Lemma 2.3 is proven. □

Analysis for the adversarial model. We assume an adversary that injects packets at rate (w, λ) for any $w \leq (q-2)/(2\lambda)$ and $0 \leq \lambda \leq 1$; that is, the adversary injects no more than $\lambda \cdot w$ packets for the same edge during every time interval of length w.

Suppose the routing time of a packet p_0 is $L + T$ or more. Then we can construct a (p_0, s, ℓ, r)-delay sequence with parameters as described in Lemma 2.3. The delay sequence specifies edges $e_1, \ldots, e_\ell$ such that e_i is traversed by packets with integral ranks from the interval r_i. On the one hand, the adversary is allowed to inject at most $\lambda \cdot (|r_i| + w - 1)$ packets with an integral rank in r_i that traverse e_i for $1 \leq i \leq \ell$. (Recall that the integral rank of a packet corresponds to its birth date.) Hence, the number of packets that traverse one of the delay edges and have the corresponding integral rank is at most

$$\sum_{i=1}^{\ell} \lambda \cdot (|r_i| + w - 1) = \lambda \cdot (r + \ell \cdot (w - 1)).$$

On the other hand, we conclude from Lemma 2.3 that the total number of packets needed for an active delay sequence corresponding to a routing time of $L + T$ steps is at least $s \geq T + r + \left(\frac{q}{2} - 2\right) \cdot \ell - \left(\frac{q}{2} - 1\right) \cdot L'$. This yields the constraint

$$\lambda \cdot (r + \ell \cdot (w - 1)) \geq T + r + \left(\tfrac{q}{2} - 2\right) \cdot \ell - \left(\tfrac{q}{2} - 1\right) \cdot L'$$

and, therefore,

$$\begin{aligned} T &\le \lambda \cdot (r + \ell \cdot (w-1)) - \left(r + \left(\tfrac{q}{2} - 2\right) \cdot \ell - \left(\tfrac{q}{2} - 1\right) \cdot L'\right) \\ &= (\lambda - 1) \cdot (r - \ell) + \left(\lambda \cdot w - (\tfrac{q}{2} - 1)\right) \cdot \ell + \left(\tfrac{q}{2} - 1\right) \cdot L'. \end{aligned}$$

This upper bound on T can be simplified as follows. First, $(\lambda - 1) \cdot (r - \ell) \le 0$ because $\lambda \le 1$ and $r \ge \ell$. Second, $\left(\lambda \cdot w - (\frac{q}{2} - 1)\right) \cdot \ell \le \left(\lambda \cdot w - (\frac{q}{2} - 1)\right) \cdot L'$ because $\ell \ge L'$ and $w \le (q-2)/(2\lambda)$, so that the factor in front of the ℓ is negative or 0. Applying these two equations to the above bound on T yields

$$\begin{aligned} T &\le \left(\lambda \cdot w - (\tfrac{q}{2} - 1)\right) \cdot L' + \left(\tfrac{q}{2} - 1\right) \cdot L' \\ &\le \lambda \cdot w \cdot L' \ \le \ \lambda \cdot w \cdot L. \end{aligned}$$

Consequently, each packet takes at most $L + \lambda \cdot w \cdot L$ time steps to reach its destination and the tuned ghost packet protocol is stable for any injection rate $\lambda \le 1$.

Analysis for the stochastic model. Now we assume that the packets are injected at random by independent generators at a rate of $\lambda = 1 - \epsilon$ with $\epsilon > 0$. Again we use a delay sequence argument. The major difference between our analysis and the analysis for the static ghost packet protocol given in [8] is that we have to deal with an arbitrarily long history of packet delays. Further, we have to use Chernoff bounds (see Lemma 1.1) instead of simple counting methods in order to prove stability results for injection rates arbitrarily close to 1.

Let L denote the depth of the network, Δ the maximum node degree, and q the size of the edge buffers. We will show that each individual packet reaches its destination within $L + T$ time steps with probability $1 - 2^{-\Omega(\epsilon^2 \cdot T)}/\epsilon^2$, provided that q and T are chosen sufficiently large, i.e., $q \ge 2(k+1)$ and $T \ge k \cdot L$ for some $k = \Theta(\log(\Delta/\epsilon)/\epsilon)$ which will be specified during the proof.

Fix an arbitrary packet p_0. Suppose that p_0 needs more than $L + T$ time steps to reach its destination. Then Lemma 2.3 yields that a (p_0, s, ℓ, r)-delay sequence is active, with

$$\begin{aligned} s &\ge T + r + \left(\tfrac{q}{2} - 2\right) \cdot \ell - \left(\tfrac{q}{2} - 1\right) \cdot L' \\ &\ge k \cdot L + r + (k-1) \cdot \ell - k \cdot L' \\ &\ge r + (k-1) \cdot \ell. \end{aligned} \tag{4}$$

We assume $k \ge 1$. Then at least r packets are needed for an active (p_0, s, ℓ, r)-delay sequence. However, we will show that the expected number of packets that pass the delay edges and have the corresponding integral rank is at most $(1 - \epsilon) \cdot r$. Hence, this event is very unlikely.

Suppose that the delay edges and the range of integral ranks for each delay edge are fixed. Define $R = r_1 \cup \cdots \cup r_\ell$. For each integral rank $j \in R$, let the binary random variable X_j^g be one if and only if generator g generates a packet that has integral rank $j \in r_i$ and traverses delay edge e_i. (The integral rank j may fall in two or more rank intervals corresponding to different delay edges.) Let $X = \sum_{j,g} X_j^g$. Every active (p_0, s, ℓ, r)-delay sequence of length s with fixed delay edges and a fixed rank assignment has at least one choice for the X_j^g's such that $X \ge s$, because each of the packets $p_1, \ldots, p_s$ traverses one of the delay edges e_i and has an integral rank in r_i. (Note that s distinct packets are needed to have an active sequence, regardless of whether a packet traverses more than one delay edge responsible for its integral rank.) Hence, if a delay sequence of length s can be constructed, then $X \ge s$.

Consequently, the probability that a fixed delay sequence becomes active is bounded above by $\Pr[X \geq s]$.

Because the integral rank of a packet corresponds to its birth date and the expected number of packets injected in a time step that include delay edge e_i in their path is at most $\lambda = 1 - \epsilon$,

$$\begin{aligned} \mathrm{E}[X] &\leq \sum_{i=1}^{\ell} (1-\epsilon) \cdot |r_i| \\ &\leq (1-\epsilon) \cdot r \qquad (5) \\ &\overset{(4)}{\leq} (1-\epsilon) \cdot s \qquad (6) \end{aligned}$$

for $k \geq 1$. The binary random variables in the sum of X are stochastically independent, because each generator is operating independently of other generators and previous time steps. Therefore, the probability for a deviation from the expectation can be estimated by using a Chernoff bound (see Lemma 1.1). We define $\delta = s/((1-\epsilon)\cdot r) - 1$. Then

$$\begin{aligned} \Pr[X \geq s] &= \Pr\left[X \geq (1+\delta) \cdot (1-\epsilon) \cdot r\right] \\ &\overset{(5)}{\leq} \mathrm{e}^{-\min\{\delta, \delta^2\} \cdot (1-\epsilon) \cdot r/3}. \end{aligned}$$

A further bound, solely depending on s and ϵ, can be derived as follows:

$$\begin{aligned} \Pr[X \geq s] &= \Pr\left[X \geq \left(1 + \frac{\epsilon}{1-\epsilon}\right) \cdot (1-\epsilon) \cdot s\right] \\ &\overset{(6)}{\leq} \mathrm{e}^{-\min\left\{\frac{\epsilon}{1-\epsilon}, \left(\frac{\epsilon}{1-\epsilon}\right)^2\right\} \cdot (1-\epsilon) \cdot s/3} \\ &\leq \mathrm{e}^{-\epsilon^2 \cdot s/3}. \end{aligned}$$

Up to now we have calculated only the probability that a fixed delay sequence becomes active. It remains to sum over all delay sequences that possibly caused the delay of packet p_0. The number of different (p_0, s, ℓ, r)-delay sequences can be bounded as follows. The maximum node degree in the network is Δ. Hence, the number of possibilities to determine the delay path starting at the destination of p_0 is at most Δ^ℓ. Once the path is fixed, the number of possibilities to choose the delay edges is at most Δ^ℓ, too, because edge e_i is incident to node v_i for $1 \leq i \leq \ell$. Further, the number of different ways of specifying the rank intervals is equivalent to the number of binary strings of length r with exactly $\ell - 1$ ones. This number is $\binom{r}{\ell-1}$. As a consequence, the probability that there exists an active (p_0, s, r, ℓ)-delay sequence for a fixed set of parameters s, ℓ, and r fulfilling the equation in Lemma 2.3 is at most

$$\begin{aligned} \binom{r}{\ell-1} \cdot \Delta^{2\ell} \cdot \Pr[X \geq s] &\leq \underbrace{\left(\frac{\mathrm{e}\Delta^2 \cdot r}{\ell-1}\right)^{\ell} \cdot \sqrt{\mathrm{e}^{-\min\{\delta,\delta^2\}\cdot(1-\epsilon)\cdot r/3}}}_{=:\,Y} \cdot \sqrt{\mathrm{e}^{-\epsilon^2 \cdot s/3}} \\ &\leq \sqrt{\mathrm{e}^{-\epsilon^2 \cdot s/3}}. \qquad (7) \end{aligned}$$

The last inequality assumes $Y \leq 1$, which holds if k is chosen sufficiently large in $\Theta(\log(\Delta/\epsilon)/\epsilon)$. We defer the corresponding calculations to the end of this proof.

Let $t(p_0)$ denote the routing time of packet p_0. If $t(p_0) > L+T$ for any $T > 0$, then we can construct an active delay sequence as described in Lemma 2.3. Consequently, the probability that $t(p_0) > L + T$ is bounded above by the probability for the existence of an active (p_0, s, ℓ, r)-delay sequence whose parameters fulfill the following constraints. Equation (4) yields $r \le s$ and $\ell \le s/(k-1) \le s$, assuming $k \ge 2$. Lemma 2.3 yields $s \ge T$. Now applying the bound in (7), which holds for $T \ge k \cdot L$, we obtain

$$\begin{aligned} \Pr[t(p_0) > L + T] &\le \sum_{s=T}^{\infty} \sum_{\ell=1}^{s} \sum_{r=1}^{s} \sqrt{\mathrm{e}^{-\epsilon^2 \cdot s/3}} \\ &\le \frac{2^{-\beta \cdot \epsilon^2 \cdot T}}{\epsilon^2} \end{aligned} \tag{8}$$

for some suitable constant β. This term is at most $N^{-\alpha}$ for any constant $\alpha > 0$ if $T \ge \alpha' \log N/\epsilon^2$ for a suitably large constant α'. (Recall that $N \ge 1/\epsilon$.) Therefore,

$$t(p_0) \;\le\; \max\left\{L \cdot (k+1), \frac{\alpha' \log N}{\epsilon^2}\right\} \;=\; O\left(\frac{L \cdot \log(\Delta/\epsilon)}{\epsilon} + \frac{\log N}{\epsilon^2}\right),$$

w.h.p. It remains to prove the bound on the expected routing time of p_0. In general, for any integer $Z \ge 0$,

$$\mathrm{E}[t(p_0)] \;=\; \sum_{i=1}^{\infty} \Pr[t(p_0) \ge i] \;\le\; Z + \sum_{i=Z+1}^{\infty} \Pr[t(p_0) \ge i].$$

Applying (8), we obtain that $\sum_{i=Z+1}^{\infty} \Pr[t(p_0) \ge i] \le 2 \cdot Z$ for $Z \ge \max\{L \cdot (k+1), L + 4 \cdot \log(1/\epsilon)/\epsilon^2\}$. In this case,

$$\mathrm{E}[t(p_0)] \;\le\; 3 \cdot Z \;=\; O\left(\frac{L \cdot \log(\Delta/\epsilon)}{\epsilon} + \frac{\log(1/\epsilon)}{\epsilon^2}\right),$$

which corresponds to the bound on the expected routing time given in Theorem 2.1.

Deferred calculations. Finally, we show that $Y \le 1$ if k is chosen sufficiently large in $\Theta(\log(\Delta/\epsilon)/\epsilon)$. First, we estimate δ.

$$\delta = \frac{s}{(1-\epsilon) \cdot r} - 1 \;\overset{(4)}{\ge}\; \frac{r + (k-1) \cdot \ell}{(1-\epsilon) \cdot r} - 1 \;\ge\; \epsilon + \frac{(k-1) \cdot \ell}{r}.$$

Depending on the value of δ, we distinguish two cases. Consider the case $\delta \le 1$. Assume $Y > 1$. Then

$$\left(\frac{\mathrm{e}\Delta^2 \cdot r}{\ell - 1}\right)^{\ell} \cdot \mathrm{e}^{-\delta^2 (1-\epsilon) \cdot r/6} > 1.$$

Applying $\delta \ge \epsilon$, that is, substituting ϵ for δ, and solving the resulting equation for r/ℓ yields $r/\ell = O\left(\log(\Delta/\epsilon)/\epsilon^2\right)$. We assume that k is chosen in such a way that

$$k \ge \sqrt{\frac{r}{\ell} \cdot \frac{6}{(1-\epsilon)} \cdot \log\left(\frac{\mathrm{e}\Delta^2 \cdot r}{\ell - 1}\right)} + 1 \;=\; O\left(\frac{\log(\Delta/\epsilon)}{\epsilon}\right)$$

for all possible choices of r and ℓ. Now applying $\delta \ge (k-1) \cdot \ell/r$ yields

$$Y \le \left(\frac{\mathrm{e}\Delta^2 \cdot r}{\ell - 1}\right)^{\ell} \cdot \mathrm{e}^{-\left(\frac{(k-1) \cdot \ell}{r}\right)^2 \cdot (1-\epsilon) \cdot r/6} \;\le\; 1.$$

Similarly, we get $Y \le 1$ for the case $\delta > 1$, too, already for $k = O(\log(\Delta/\epsilon))$. This completes the proof of Theorem 2.1. □

3. Dynamic routing in arbitrary networks with bounded buffers. The tuned ghost protocol can be used to construct an efficient routing algorithm for arbitrary paths in an arbitrary (nonleveled) network G with bounded buffers. In this section, we consider only the stochastic model. In section 6, we will show how the results obtained in this model can be adapted to the adversarial model. In the stochastic model, our approach yields a dynamic routing protocol that is stable for any injection rate $\lambda < 1$ and requires only small edge buffers. The dynamic protocol uses the following simulation technique.

Suppose the maximum length of a routing path is D. Define $L = \lceil D \cdot (1 + 1/\epsilon) \rceil$ with $\epsilon = 1 - \lambda$. G simulates the tuned ghost protocol on a leveled network G' of depth L under a maximum injection rate of $\lambda' \leq 1 - \epsilon^2$. G' is defined as follows. On each level, it contains a node for every node in G. A node u from level i and a node v from level $i+1$ for $0 \leq i \leq L-1$ are connected by an edge if and only if the corresponding nodes in G are connected by an edge.

Each edge of the leveled network is simulated by its respective counterpart in G. Hence, every edge in G has to simulate L edges of G' and, therefore, the buffer size in G has to be L times the buffer size of the simulated network G'. The simulation works in a round-robin fashion; that is, in each time step t, the edges of G simulate the edges of G' between the nodes of level i and level $i+1$ with $i = t \bmod L$. For each injected packet p, the generator chooses an *offset* κ_p uniformly at random from the range 0 to $\lceil D/\epsilon \rceil - 1$. The routing path in the leveled network starts from level κ_p and simply follows the course prescribed by the original path until it reaches the packet's destination on level $\kappa_p + d_p \leq L$, where d_p denotes the length of the routing path.

Next we calculate the virtual injection rate λ' in the simulated network G'. On the one hand, the virtual rate at which each generator injects packets into G' is L times larger than the actual rate in G, because each edge in G is activated only every Lth step. On the other hand, the probability that an injected packet that traverses an edge e in G also traverses a fixed edge e' in G' that corresponds to e is at most $1/\lceil D/\epsilon \rceil$ because of the randomly selected offset. Therefore,

$$\lambda' \leq \frac{\lambda \cdot L}{\lceil D/\epsilon \rceil} = \frac{(1-\epsilon) \cdot \lceil D \cdot (1+1/\epsilon) \rceil}{\lceil D/\epsilon \rceil} \leq (1-\epsilon) \cdot (1+\epsilon) = 1 - \epsilon^2.$$

Substituting this injection rate into Theorem 2.1 and applying $L = \Theta(D/\epsilon)$ yields the following result.

COROLLARY 3.1. *The simulation of the tuned ghost packet protocol yields a dynamic routing algorithm that is stable for any injection rate $\lambda \leq 1 - \epsilon$ for any $\epsilon > 0$, provided that buffers of sufficiently large size $O(D \cdot \log(\Delta/\epsilon)/\epsilon^3)$ are used. Furthermore, each packet is delivered in time $O(D^2 \cdot \log(\Delta/\epsilon)/\epsilon^4 + D \cdot \log(1/\epsilon)/\epsilon^5)$, expected, and $O(D^2 \cdot \log(\Delta/\epsilon)/\epsilon^4 + D \cdot (\log N)/\epsilon^5)$, w.h.p.*

4. Routing along shortest paths in arbitrary networks. In this section, we assume that each buffer has space for an unlimited number of packets. The goal is to achieve a better routing time than in the previous section, in which we assumed buffers of limited size. We assume that packets do not make detours, that is, all routing paths are shortest paths. Initially, we investigate the stochastic model. In section 6, we will show how the results obtained in this model can be adapted to the adversarial model.

We investigate a dynamic variant of the growing rank protocol that was introduced for static routing in [10, 11]. We have introduced this dynamic variant before in [16],

but the analysis we give there holds only for injection rates $\lambda < 1/e$. In the following, we show that the results can be extended to hold for any constant injection rate $\lambda < 1$.

The dynamic growing rank protocol works as follows. Define the initial rank of a packet to be the time step in which the packet is injected. Whenever the packet traverses a link, its rank is increased by some fixed integer $m \geq 1$, which will be specified later on. If several packets want to traverse the same link at the same time, then the packet with minimal rank is chosen. (In order to break ties, if there are several packets with the same rank, the packet with minimum generator id is taken.)

The following theorem summarizes the results of our analysis of the dynamic growing rank protocol. Note that neither the maximum injection rate for which the protocol is stable nor the routing time depends on the degree of the network.

THEOREM 4.1. *Suppose all routing paths are shortest paths. Then the growing rank protocol is stable for any injection rate λ up to some $1-\epsilon$ with $\epsilon = \Theta((\log m)/\sqrt{m})$. Furthermore, the routing time for each individual packet p that has to travel along a routing path of length d_p is $O(m \cdot d_p)$, expected, and $O(m \cdot (d_p + \log N))$, w.h.p.*

We point out that similar results can be obtained by applying the black box transformation scheme presented later in section 5. The direct transformation of the growing rank protocol, however, is more natural and more elegant, as it does not require to partition the time into fixed size blocks in which the static protocol is executed. Further, the expected routing time guaranteed by the specific transformation is slightly better, i.e., $O(d_p)$ rather than $O(d_p + \log N)$.

Proof. We use a delay sequence argument that is similar to the one for the ghost packet protocol. A (p, s, ℓ, r)-delay sequence is defined as in Definition 2.2 except for the following changes. The delay edges are the edges on the delay path rather than edges that are only incident on that path. The intervals of ranks do not overlap. Instead, the smallest rank in r_i is equal to the maximum rank in r_{i+1} plus m; that is, neighboring intervals are separated by a gap of $m-1$ (integral) ranks. As an additional component, the delay sequence includes ℓ integers $s_1, \ldots, s_\ell$ such that $s = (\sum_{i=1}^{\ell} s_i) - \ell + 1$. In an active delay sequence, each delay edge e_i must be traversed by s_i packets with a rank from r_i.

LEMMA 4.2. *Suppose a packet p_0 that has a routing path of length d_{p_0} takes $m \cdot d_{p_0} + T$ or more steps to reach its destination. Then there is an active (p_0, s, ℓ, r)-delay sequence with $s \geq T + r + (m-2) \cdot \ell$.*

Proof. The construction of the active delay sequence is analogous to the one used in section 2 for the ghost packet protocol. In fact, the construction becomes slightly simpler as we have to consider neither delays because of blocked edges nor delays due to ghost packets. The shortest path restriction ensures that all recorded delay packets are distinct. (For a proof, see [11, Lemma 2.3.]) Neighboring intervals of ranks are separated by a gap of $m-1$ ranks because the rank of a packet increases by m whenever the packet moves along an edge.

It remains to prove the bound on the parameters of the delay sequence. Recall that $r = \sum_{i=1}^{\ell} |r_i|$ with $|r_i|$ denoting the length of the interval of ranks assigned to delay edge e_i. The highest rank in the recorded sequence is $\text{birth}(p_0) + (d_{p_0} - 1) \cdot m$, which is the rank of p_0 on the last edge of its routing path, and the lowest rank is $\text{birth}(p_s)$, which is the rank of p_s on the first edge of its routing path. As neighboring intervals of ranks are separated by a gap of $m-1$ ranks, we get

$$\begin{aligned} r &= \text{birth}(p_0) - \text{birth}(p_s) + (d_{p_0} - 1) \cdot m - (\ell - 1) \cdot (m-1) + 1 \\ &= \text{birth}(p_0) - \text{birth}(p_s) + d_{p_0} \cdot m - \ell \cdot (m-1). \end{aligned} \tag{9}$$

Define t to be the number of time steps covered by the construction, i.e., the time from the birth of the packet p_s to the arrival of the packet p_0 at its destination. Then

$$\begin{aligned} t &\geq \text{birth}(p_0) - \text{birth}(p_s) + d_{p_0} \cdot m + T \\ &\overset{(9)}{=} r + (m-1) \cdot \ell + T. \end{aligned} \tag{10}$$

Each of the t time steps recorded in the delay sequence is either one of s delays or one of ℓ packet movements. Therefore, $t = s + \ell$, and we can conclude

$$\begin{aligned} s &= t - \ell \\ &\overset{(10)}{\geq} r + (m-2) \cdot \ell + T, \end{aligned}$$

which completes the proof of Lemma 4.2. □

Now fix an arbitrary packet p_0, and suppose that p_0 needs more than $d_{p_0} \cdot m + T$ time steps to reach its destination. Then Lemma 4.2 yields that a (p_0, s, ℓ, r)-delay sequence is active with

$$s \geq r + (m-2) \cdot \ell \;\geq\; r + \ell \tag{11}$$

for $m \geq 3$. Hence, at least $r + \ell$ packets are needed for an active (p_0, s, ℓ, r)-delay sequence. However, we will show that the expected number of packets that pass the delay edges and have the corresponding rank is at most $(1-\epsilon) \cdot r$. Hence, this event is very unlikely.

Suppose that the delay edges and the range of ranks for each delay edge are fixed. For $j \in r_1 \cup \cdots \cup r_\ell$ and $1 \leq i \leq \ell$, let the binary random variable $X_{j,i}^g$ be one if and only if generator g generates a packet that has rank $j \in r_i$ at delay edge e_i. Let $X = \sum_{j,i,g} X_{j,i}^g$. Then the probability that a fixed delay sequence becomes active is bounded by $\Pr[X \geq s]$.

The expected number of packets traversing a fixed edge e with some fixed rank r is at most $\lambda = 1 - \epsilon$ because the rank of a packet p at edge e corresponds to its injection time plus an offset that depends on the distance between e and the source node of p. Therefore,

$$\begin{aligned} \mathrm{E}[X] &\leq \sum_{i=1}^{\ell} (1-\epsilon) \cdot |r_i| \\ &\leq (1-\epsilon) \cdot r \\ &\overset{(11)}{\leq} (1-\epsilon) \cdot s. \end{aligned} \tag{12}$$

Thus, applying a Chernoff bound (see Lemma 1.1) yields

$$\begin{aligned} \Pr[X \geq s] &= \Pr\left[X \geq \left(1 + \frac{\epsilon}{1-\epsilon}\right) \cdot (1-\epsilon) \cdot s\right] \\ &\overset{(12)}{\leq} \mathrm{e}^{-\min\left\{\frac{\epsilon}{1-\epsilon}, \left(\frac{\epsilon}{1-\epsilon}\right)^2\right\} \cdot (1-\epsilon) \cdot s/3} \\ &\leq \mathrm{e}^{-\epsilon^2 \cdot s/3}. \end{aligned} \tag{13}$$

So far, we have only calculated the probability that a fixed delay sequence becomes active. It remains to multiply this probability with the number of possible delay sequences. The number of possibilities to choose the r_i's is $\binom{r}{\ell-1}$. The number of

possibilities to choose the s_i's is $\binom{s+\ell-1}{\ell-1}$. Enumerating explicitly all possible delay paths, as we have done for the ghost packet protocol, would give us another factor of Δ^ℓ. However, we can avoid this factor in the case of the growing rank protocol because the delay path is fixed when the delay packets and the s_i's are specified.

The technical problem with the last assumption is that we supposed before, when estimating the probability for $X \geq s$, that the delay path is fixed whereas we assume here that the delay packets and, hence, the random variables $X^g_{j,i}$ are fixed. This dilemma can be solved by constructing the delay path iteratively. Suppose we have specified the delay path up to the dth edge for some $d \geq 0$ starting from the destination of packet p_0. At this point only some of the $X^g_{j,i}$ random variables above are well defined, namely, those variables with $i \leq d$. The specification of the outcome of these variables, however, gives us the delay packets $p_0, \ldots, p_k$ for $k = \sum_{i=1}^{d} s_i$, and following the path of packet p_k backwards for one edge gives us the next edge on the delay path. In this way, the delay path and the outcome of the $X^g_{j,i}$ variables can be determined alternately. Note that the fact that the definition of some random variables depends on the outcome of other variables does not affect the applicability of the Chernoff bounds because their outcome remains independent.

Combining all these results, the probability that a (p_0, s, ℓ, r)-delay sequence for a fixed set of parameters p_0, s, ℓ, and r is active is bounded above by

$$\binom{r}{\ell-1} \cdot \binom{s+\ell-1}{\ell-1} \cdot \Pr[X \geq s] \overset{(11)(13)}{\leq} \left(\frac{\mathrm{e}s}{\ell}\right)^{2\ell} \cdot \mathrm{e}^{-\epsilon^2 \cdot s/3}.$$

Now we can bound the probability that packet p_0 has a long routing time. If this packet takes $d_{p_0} \cdot m + T$ steps, then a (p_0, s, ℓ, r)-delay sequence is active with $s \geq T + r + (m-2) \cdot \ell$. This constraint yields $s \geq T$, $\ell \leq s/(m-2)$, and $r \leq s$. Hence, the probability for this event is bounded above by

$$\sum_{s=T}^{\infty} \sum_{\ell=1}^{s/(m-2)} \sum_{r=1}^{s} \left(\frac{\mathrm{e}s}{\ell}\right)^{2\ell} \cdot \mathrm{e}^{-\epsilon^2 \cdot s/3} \overset{(11)}{\leq} \sum_{s=T}^{\infty} s^2 \cdot (\mathrm{e}(m-2))^{2s/(m-2)} \cdot \mathrm{e}^{-\epsilon^2 \cdot s/3}$$
$$= \frac{2^{-\Omega(\epsilon^2 \cdot T)}}{\epsilon^2}$$

if m is chosen appropriately, that is, if we set $m \geq k \cdot \log(1/\epsilon)/\epsilon^2 + 2$ for a sufficiently large constant k. From this result the bounds on the routing time given in Theorem 4.1 can be derived analogously to the calculations for the ghost packet protocol in section 2. □

5. The black box transformation. In this section, we present a black box transformation scheme for arbitrary static, oblivious routing protocols using the stochastic injection model. In section 6, we will show that the results for the stochastic model can be adapted to the adversarial model, too.

In the adversarial model, the bounds on the routing time obtained by our transformation are almost equivalent to the results achieved by Rabani and Tardos [13]. However, in addition we prove the stability of the dynamic protocols. Furthermore, in the stochastic model, we show an interesting extra feature: the dynamic protocol delivers each packet in a time corresponding to its individual path length even if the transformed static protocol does not provide that property. Moreover, we show that our black box transformation ensures a fast recovery from any worst-case scenario.

Our transformation scheme is especially simple and efficient if the given static protocol is pure. A protocol is called *pure* if it does not use any control packets and

does not duplicate any of its packets; that is, every packet crosses one edge after the other on its routing path, and no other messages are sent. The main results of this section are listed in the following theorem.

Recall that the congestion C of a static routing problem is the maximum number of paths in a path collection crossing the same edge, the dilation D is the maximum length of a path in a path collection, and the complexity M of a static routing problem is the maximum of the number of edges in the network and paths in the path collection. Further, the complexity N of a dynamic routing problem is defined to be the maximum of the number of generators, the number of edges, and $1/(1-\lambda)$.

THEOREM 5.1. *Let $\mathcal{S}$ denote any set of paths in an arbitrary network. Consider any static routing protocol $\mathcal{P}$ that sends packets along every collection of paths or subpaths from $\mathcal{S}$ in at most $\gamma C + \delta D + O(\log^\alpha M)$ time steps, w.h.p. Then $\mathcal{P}$ can be transformed into a dynamic routing protocol $\mathcal{P}'$ possessing the following properties in the stochastic model. Suppose packets are injected at rate λ and are to be routed solely along paths or subpaths from $\mathcal{S}$. Then the expected routing time of a packet following a path of length d is at most*

(1) $O(\epsilon^{-2}(\delta d + \log^\alpha N))$ *if $\alpha \geq 1, \gamma \geq 1$ are constants, $\delta = \Theta(\log^\beta N)$ for some constant $0 \leq \beta \leq 1$ and $\lambda = (1-\epsilon)/\gamma$ for some $\epsilon > 0$;*

(2) $O((\epsilon/2)^{-\frac{2}{1-\beta}}(d \log^{\alpha\beta} N + \log^\alpha N))$ *if $\alpha \geq 1, \gamma \geq 1$ are constants, $\delta = C^\beta$ for some constant $0 < \beta < 1$ and $\lambda = (1-\epsilon)/\gamma$ for some $\epsilon > 0$;*

(3) $O(\epsilon^{-2}(d \log N + \log^2 N))$ *if the bound on the runtime of $\mathcal{P}$ is $C \cdot D$ and $\lambda = (1-\epsilon)/\log N$ for some $\epsilon > 0$.*

All time bounds also hold w.h.p. Furthermore, the recovery time in all three cases is equal to the respective time bound with d replaced by $D(\mathcal{S})$, the length of the longest path in $\mathcal{S}$. If $\mathcal{P}$ is pure, then $\mathcal{P}'$ is stable for any $\lambda < 1$.

Notice that the bounds on the expected routing time of a packet imply stability up to the specified injection rate, depending on the parameters of the static protocol. The stability of pure protocols, however, is independent of these parameters, although the delay of a packet might become exponential in its path length if the injection rate is too high.

Certainly, more than the three cases stated in the theorem can be solved with the techniques below, but we believe that these cases are the most natural ones. Case 1 covers the case in which γ, δ, and α are constants, which we believe is the most important case. Let us apply Theorem 5.1 to some of the static protocols mentioned in section 1.1.1.

- *Routing along simple paths in arbitrary networks.* The static protocol of Ostrovsky and Rabani [12] with runtime $O(C + D + \log^{1+\kappa} M)$, w.h.p., for any constant $\kappa > 0$ can be transformed into a dynamic protocol that guarantees a routing time of $O(d + \log^{1+\kappa} N)$, w.h.p., for any λ up to some constant < 1.
- *Routing along arbitrary paths in arbitrary networks.* The simple static protocol with runtime $(1+\kappa) \cdot C + O(D \log M)$, w.h.p., for any constant $\kappa > 0$, presented by Leighton, Maggs, and Rao in [9], can be used to obtain a dynamic protocol that guarantees a routing time of $O(d \log N)$, w.h.p., for any constant $\lambda < 1$.
- *Greedy routing along simple paths in arbitrary networks.* Any greedy routing protocol can be transformed into a dynamic protocol that delivers each packet in time $O\left(d \log N + \log^2 N\right)$, w.h.p., for any constant $\lambda < 1/\log N$. We point out that the dynamic protocol is greedy, too.

The first and second examples follow from case 1 of the theorem, whereas the last

example follows from case 3. Since all of these protocols are pure, their dynamic counterparts are stable for any $\lambda < 1$.

In the following we show how to transform $\mathcal{P}$ into a dynamic protocol $\mathcal{P}'$ so that Theorem 5.1 holds.

5.1. Description of $\mathcal{P}'$. Let $\mathcal{P}$ be any static, oblivious routing protocol, and let the injection rate λ be $1 - \epsilon$ for some $\epsilon > 0$. Consider the time to be partitioned into consecutive intervals of length T. Let c_T and d_T denote suitable integers. (These parameters will be specified later.) Every newly injected packet waits until the beginning of the next T-interval. Afterwards it tries to traverse d_T edges of its path in each T-interval until it reaches its destination. Whenever it manages to traverse d_T edges within a T-interval, it waits for the next T-interval. If it fails to traverse d_T edges in some T-interval, it is declared a *failed* packet for the rest of the routing.

$\mathcal{P}'$ now works as follows: At the beginning of each T-interval, $\mathcal{P}$ is started with parameters c_T, used as a bound for the congestion, and d_T, used as a bound for the dilation. All packets that have not failed yet are allowed to participate in $\mathcal{P}$. After $(1-\epsilon^2/2)T$ time steps of the T-interval, $\mathcal{P}$ is halted. (All packets that did not manage to traverse d_T edges up to this point fail.) The remaining $T \cdot \epsilon^2/2$ time steps are reserved for the failed packets. To the failed packets, a contention resolution rule called shortest-in-system (SIS) is applied. SIS always gives precedence to the packet most recently injected (i.e., of youngest age) in case several packets contend for the same edge.

If $\mathcal{P}$ is pure, then we can improve $\mathcal{P}'$ so that it becomes greedy; i.e., a packet only has to wait because the next edge on its path is used by another packet. In the pure case, we use SIS as a primary contention resolution rule but we manipulate the age of the nonfailed packets as follows. (As before, a packet that has not traversed $i \cdot d_T$ edges by the end of its ith T-interval is declared a *failed* packet for the rest of its life.) Every newly injected packet gets an initial age of ∞ until the beginning of the next T-interval. Afterwards, as long as it has not failed or reached its destination yet, its age is set to 0 at the beginning of each T-interval. Whenever it traverses $i \cdot d_T$ edges before the end of the ith T-interval in which it participates, its age is set back to ∞ for the rest of that interval. The age of a failed packet is defined by its injection time. In each time interval, the packets with age 0 are scheduled according to protocol $\mathcal{P}$. Notice that packets that participate in $\mathcal{P}$ always have a lower age than the other packets. Hence, if we use SIS as primary contention resolution rule, we can allow the failed packets to be routed together with the other packets without disturbing the schedule of $\mathcal{P}$.

In order to analyze the performance of $\mathcal{P}'$, we need to study the behavior of SIS.

Bounding the routing time for shortest-in-system. In this section we bound the routing time for SIS, given the following model.

Suppose that packets are injected with rate λ, using our standard stochastic model. A packet is said to be *old* if the difference between the actual time step and its injection time is more than K. Otherwise it is called *young*. As long as a packet is young, it is allowed to set its age to an arbitrary value in each step. The age of an old packet is determined by its injection time. The probability that a packet becomes old before it reaches its destination is assumed to be at most p_f (which denotes its *failure probability*). The event that a packet is old may influence the probability that another packet may also be old. We model these dependencies via a *dependency graph* $G = (V, E)$. Each node V represents a (generator, time step)-pair. Edges are chosen in G such that for any independent set $\{(g_1, t_1), \ldots, (g_k, t_k)\} \in V$ with $k \in \mathbb{N}$, under

the assumption that g_i injects a packet P_i at step t_i, the probability that P_i becomes old for all $i \in \{1, \ldots, k\}$ is at most p_f^k. That is, for a proof of an upper bound on the number of old packets, any independent set of (g,t)-pairs can be viewed as having independent failure probabilities.

For this model we show the following lemma.

LEMMA 5.2. *For any $0 < \epsilon < 1$ and $K \geq 0$, the SIS protocol ensures that, under the above model with injection rate $\lambda = 1 - \epsilon$, failure probability p_f and a dependency graph G of maximum degree b, the expected routing time of a packet following a path of length d is at most*

(1) $O((d/\epsilon) \cdot (d(\epsilon^{-1} + b^2) + K))$ *if* $p_f \leq \epsilon/(2(4d+1))$, *and*
(2) $O((K+d) \cdot d/\epsilon^{2d})$ *otherwise.*

Proof. We start by proving item 2. Suppose that there is a packet P that has a routing time of more than $\sum_{i=1}^{d} \tau/\epsilon^{2i}$ steps for some $\tau \geq 2K$. In this case there must exist an $i \in \{1, \ldots, d\}$ for which P was delayed for at least τ/ϵ^{2i} steps at the ith edge of its path. Consider the minimum i for which this holds. Let e be the ith edge on P's path. Then the time interval I from the injection of P till the time when P waited at e for the τ/ϵ^{2i}th time consists of at most

$$\begin{aligned} t &= \sum_{j=1}^{i} \tau/\epsilon^{2j} \leq \tau \cdot \frac{1}{\epsilon^2} \cdot \frac{(1/\epsilon)^{2i} - 1}{(1/\epsilon)^2 - 1} \\ &= \tau \cdot \frac{(1/\epsilon)^{2i} - 1}{1 - \epsilon^2} \end{aligned}$$

time steps. Let the random variable X denote the number of (young and old) packets generated during I with paths containing e. Since P is allowed to choose its age in an arbitrary way only during the first K steps of its life, it holds that if P had to wait for at least τ/ϵ^{2i} steps at e, then $X \geq \tau/\epsilon^{2i} - K$. Clearly,

$$\begin{aligned} \mathrm{E}[X] &\leq (1-\epsilon) \cdot t = \frac{1}{1+\epsilon} \cdot \tau((1/\epsilon)^{2i} - 1) \\ &\leq \left(1 - \frac{\epsilon}{1+\epsilon}\right)(\tau/\epsilon^{2i} - K). \end{aligned}$$

This is less than the number of packets generated in I that have to delay P at e. Because of the independence assumptions in our stochastic injection model we can apply Chernoff bounds (see Lemma 1.1) to show that for any $0 < \epsilon \leq 1$ the probability that p is delayed by at least τ/ϵ^{2i} packets at e is at most

$$\begin{aligned} \Pr\left[X \geq (1+\epsilon)\left(1 - \tfrac{\epsilon}{1+\epsilon}\right)(\tau/\epsilon^{2i} - K)\right] &\leq \mathrm{e}^{-\epsilon^2 \cdot \left(1 - \frac{\epsilon}{1+\epsilon}\right) \cdot (\tau/\epsilon^{2i} - K)/3} \\ &\leq \mathrm{e}^{-\epsilon^2 \cdot (1/2) \cdot (\tau - K)(1/\epsilon^{2i})/3} \leq \mathrm{e}^{-\tau \cdot (1/\epsilon)^{2(i-1)}/12}. \end{aligned}$$

Hence the probability that the routing time of P exceeds $\sum_{i=1}^{d} \tau/\epsilon^{2i}$ is at most

$$\sum_{i=1}^{d} \mathrm{e}^{-\tau \cdot (1/\epsilon)^{2(i-1)}/12} \leq d \cdot \mathrm{e}^{-\tau/12}.$$

Thus the expected routing time of P is at most

$$\begin{aligned}&\sum_{i=1}^{d} 2K/\epsilon^{2i} + \sum_{\tau \geq 2K} \left(\sum_{i=1}^{d} (\tau+1)/\epsilon^{2i} \right) \cdot d \cdot \mathrm{e}^{-\tau/12} \\ &\leq d \cdot 2K/\epsilon^{2d} + \sum_{\tau \geq 2K} (d(\tau+1)/\epsilon^{2d}) \cdot d \cdot \mathrm{e}^{-\tau/12} \\ &= O\left((K+d) \cdot d/\epsilon^{2d}\right).\end{aligned}$$

Next we prove item 1. Suppose that there is a packet P that has a routing time of more than $K + d \cdot 4\tau/\epsilon$ steps for some $\tau \geq K$. In this case there must exist an $i \in \{1, \ldots, d\}$ for which P was delayed for at least $4\tau/\epsilon$ time steps at the ith edge of its path while it was already old. Let I be a time interval covering $4\tau/\epsilon$ of these steps. Furthermore, let the random variable X denote the number of young packets that delayed P in I, and let the random variable Y denote the number of old packets that delayed P in I. It clearly holds that

$$\Pr[X + Y = 4\tau/\epsilon] \leq \Pr[X \geq \tau(4/\epsilon - 1)] + \Pr[Y \geq \tau].$$

We first bound the probability that $X \geq \tau(4/\epsilon - 1)$. It holds that

$$\mathrm{E}[X] \leq \lambda(|I| + K) = (1-\epsilon)(4\tau/\epsilon + K).$$

Because of the independence assumptions in our stochastic injection model, it follows that

$$\begin{aligned}\Pr[X \geq (1 + \tfrac{\epsilon}{2(1-\epsilon)})(1-\epsilon)(4\tau/\epsilon + K)] &\leq \mathrm{e}^{-\min\{(\frac{\epsilon}{2(1-\epsilon)})^2, \frac{\epsilon}{2(1-\epsilon)}\}(1-\epsilon)(4\tau/\epsilon+K)/3} \\ &\leq \mathrm{e}^{-c_1 \cdot \epsilon \cdot \tau}\end{aligned}$$

for some constant $c_1 > 0$. Since $(1 + \frac{\epsilon}{2(1-\epsilon)})(1-\epsilon) = 1 - \epsilon/2$ and

$$\frac{4\tau}{\epsilon} - (1 - \tfrac{\epsilon}{2}) \left(\frac{4\tau}{\epsilon} + K \right) \geq 2\tau - K \geq \tau\ ,$$

we also have

$$\Pr[X \geq \tau(4/\epsilon - 1)] \leq \mathrm{e}^{-c_1 \cdot \epsilon\tau}\ .$$

Next we bound the probability that $Y \geq \tau$. Since P can be delayed in I only by old packets that are younger than P, we get with $p_f \leq \epsilon/(2(4d+1))$ that

$$\mathrm{E}[Y] \leq \left(K + \frac{d \cdot 4\tau}{\epsilon} \right) \cdot p_f \leq \frac{\tau}{2}.$$

In order to prove a tail estimate for Y, we need the following claim which is due to Rabani and Tardos [13]. (We present their result in a slightly improved form.)

Claim 1. *If $X_1, \ldots, X_n$ are binary random variables with a dependency graph of degree at most d and $\mathrm{E}[X_i] \leq p$ for all i, then for any $\delta > 0$,*

$$\Pr[S \geq (1+2\delta)pn] \leq 4d \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{1+\delta}} \right)^{p \cdot n/(2d)}.$$

Proof. Any graph of degree $d \geq 1$ can be partitioned into at most $d+1$ independent sets, and therefore into $m \leq c \cdot d + (d+1)$ independent sets, each of size at most $n/(c \cdot d)$ for any $c \in \mathbb{N}$. Let us choose $c = 2$, and let the corresponding set sizes be $n_1, \ldots, n_m$. Let S_i denote the sum of the variables in set i. Since set i is independent and $\mathrm{E}[X_i] \leq p$ for all i,

$$\Pr[S_i \geq p \cdot n_i + \delta pn/(2d)] \leq \Pr[S_i \geq (1+\delta)pn/(2d)] \leq \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{1+\delta}}\right)^{p \cdot n/(2d)}$$

according to the Chernoff bounds. Thus we obtain

$$\begin{aligned}\Pr[S \geq (1+2\delta)pn] &\leq \Pr\left[\bigvee_{i \in [m]} S_i \geq p \cdot n_i + \delta pn/(2d)\right] \\ &\leq 4d \left(\frac{\mathrm{e}^{\delta}}{(1+\delta)^{1+\delta}}\right)^{p \cdot n/(2d)}. \qquad \square\end{aligned}$$

Using this claim, we obtain that

$$\Pr[Y \geq \tau] \leq 4b \cdot \mathrm{e}^{-c_2 \cdot \tau/b}$$

for some constant $c_2 > 0$.

The probability bounds for X and Y and the fact that there are d possibilities to select an edge where P experiences a high delay imply that the probability that the routing time of P exceeds $K + d \cdot 4\tau/\epsilon$ is at most

$$d\left(\mathrm{e}^{-c_1 \cdot \epsilon\tau} + 4b \cdot \mathrm{e}^{-c_2 \cdot \tau/b}\right).$$

Thus the expected routing time of P is at most

$$\begin{aligned}&\left(K + \sum_{i=1}^{d} 4K/\epsilon\right) + \sum_{\tau \geq K} (K + d \cdot 4(\tau+1)/\epsilon) \cdot d\left(\mathrm{e}^{-c_1 \cdot \epsilon\tau} + 4b \cdot \mathrm{e}^{-c_2 \cdot \tau/b}\right) \\ &\leq d \cdot 5K/\epsilon + \sum_{\tau \geq K} d \cdot 5(\tau+1)/\epsilon \cdot d \cdot \mathrm{e}^{-c_1 \cdot \epsilon\tau} + \sum_{\tau \geq K} d \cdot 5(\tau+1)/\epsilon \cdot d \cdot 4b \cdot \mathrm{e}^{-c_2 \cdot \tau/b} \\ &= O\left(\frac{K \cdot d}{\epsilon} + \frac{d^2}{\epsilon^2} + \frac{d^2 \cdot b^2}{\epsilon}\right) = O\left(\tfrac{d}{\epsilon}(d(\tfrac{1}{\epsilon} + b^2) + K)\right). \qquad \square\end{aligned}$$

5.2. Analysis of $\mathcal{P}'$ for nonpure $\mathcal{P}$. First let us introduce some notation. Given a time interval I, $|I|$ denotes its *size*, i.e., the time range it includes. A packet p is called a *$\mathcal{P}$-packet* in a T-interval I if $\mathcal{P}$ is applied to p in I, i.e., p was generated before I and has not failed nor reached its destination yet.

Bounding the routing time of successful packets. To prove bounds on the routing time of successful packets, we show the following lemma.

LEMMA 5.3. *Let $\mathcal{S}$ denote any set of paths, and let ϕ be any constant greater than 0. Consider any static routing protocol $\mathcal{P}$ that sends packets along any path collection (consisting of paths from some specified set) with congestion C and dilation D in at most $\gamma C + \delta D + O(\log^{\alpha} N)$ time steps, with probability at least $1 - N^{-\phi}/2$, where $\alpha, \gamma, \delta \geq 1$. Then the following hold for each T-interval I.*

(1) *If* $\gamma = \Theta(1)$, $\delta = \Theta(\log^\beta N)$ *for some constant* $0 \le \beta \le 1$, $\lambda = (1-\epsilon)/\gamma$ *for some* $\epsilon > 0$, $d_T = (\log N)^{\alpha-\beta}$, *and* $T = \Theta(\epsilon^{-2}(\phi \log N + \log^\alpha N))$ *is sufficiently large, then with probability at least* $1 - N^{-\phi}$, $\mathcal{P}'$ *requires at most* $(1-\epsilon^2/2)T$ *time steps in* I *to send any fixed* $\mathcal{P}$*-packet along* d_T *edges.*

(2) *If* $\gamma = \Theta(1)$, $\delta = C^\beta$ *for some constant* $0 < \beta < 1$, $\lambda = (1-\epsilon)/\gamma$ *for some* $\epsilon > 0$, $d_T = (\log N)^{\alpha(1-\beta)}$, *and* $T = \Theta((\epsilon/2)^{-\frac{2}{1-\beta}}(\phi \log N + \log^\alpha N))$ *is sufficiently large, then with probability at least* $1 - N^{-\phi}$, $\mathcal{P}'$ *requires at most* $(1-\epsilon^2/2)T$ *time steps in* I *to send any fixed* $\mathcal{P}$*-packet along* d_T *edges.*

(3) *If the runtime of* $\mathcal{P}$ *is at most* $C \cdot D$, $\lambda = (1-\epsilon)/\log N$ *for some* $\epsilon > 0$, $d_T = \log N$, *and* $T = \Theta(\epsilon^{-2}\phi \log^2 N)$ *is sufficiently large, then with probability at least* $1 - N^{-\phi}$, $\mathcal{P}'$ *requires at most* $(1-\epsilon^2/2)T$ *time steps in* I *to send any fixed* $\mathcal{P}$*-packet along* d_T *edges.*

Proof. First, we bound the expected number of $\mathcal{P}$-packets that participate in a T-interval. Consider any fixed edge e. Let the random variable X_t^e denote the number of packets generated at time step t that intend to cross e. Clearly, $\mathrm{E}[X_t^e] \le \lambda$. Furthermore, let the binary random variable $X_{t,k}^g$ be 1 if and only if generator g generates at time step t a packet that has to cross e as kth edge. Then it holds that $X_t^e = \sum_{g,k} X_{t,k}^g$. Now, let us consider some fixed T-interval I that starts at time t_0. Since we require each packet to traverse $i \cdot d_T$ edges till the end of the ith T-interval in which it participates, the expected number of $\mathcal{P}$-packets that intend to cross e in I is given by

$$\begin{aligned}\sum_g \sum_{i\ge 0} \sum_{j=0}^{T-1} \sum_{k=0}^{d_T-1} \mathrm{E}[X^g_{t_0-(i+1)T+j,i\cdot d_T+k}] &= \sum_g \sum_{i\ge 0} \sum_{k=0}^{d_T-1} T \cdot \mathrm{E}[X^g_{t_0,i\cdot d_T+k}] \\ &= \sum_g \sum_d T \cdot \mathrm{E}[X^g_{t_0,d}] \\ &\le T \cdot \lambda,\end{aligned}$$

because $\Pr[X^g_{t_1,d} = 1] = \Pr[X^g_{t_2,d} = 1]$ for any t_1, t_2, since the injection of packets is independent of the time step.

Now we bound T so that with probability at least $1 - N^{-\phi}$ the $\mathcal{P}$-packets participating in some T-interval I are successful in I. Let us assume that $\lambda = (1-\epsilon)/\gamma$ for some $\epsilon > 0$ (resp., $\lambda = (1-\epsilon)/\log N$ in case 3). We now consider the three cases for the choice of γ and δ given in Lemma 5.3.

Case 1. $\gamma = \Theta(1)$ and $\delta = \Theta(\log^\beta N)$ for some constant $0 \le \beta \le 1$.

From above we know that the expected number of $\mathcal{P}$-packets participating in a T-interval that cross some fixed edge is at most λT.

Suppose that $\epsilon \le 7/8$. Since the injection of a packet is independent of the injection of other packets, we can use a Chernoff bound to show that the probability that the congestion caused by $\mathcal{P}$-packets exceeds $(1+\epsilon)\lambda T$ at some fixed edge is at most $\mathrm{e}^{-\epsilon^2 \lambda T/3}$. Hence, for every $\phi > 0$ there is a $t(\epsilon, \phi) = \Theta(\frac{\phi}{\epsilon^2} \log N)$ such that for all $T \ge t(\epsilon, \phi)$, the probability is at least $1 - N^{-\phi}/2$ that the congestion caused by $\mathcal{P}$-packets in I is at most $(1+\epsilon)\lambda T$. Assume in the following that $T \ge t(\epsilon, \phi)$. Set $c_T = (1+\epsilon)\lambda T$ and $d_T = \log^\psi N$. According to the assumptions of Lemma 5.3, the runtime of $\mathcal{P}$, given a path collection with congestion c_T and dilation d_T, is at most $\gamma c_T + \delta d_T + O(\log^\alpha N)$ with probability at least $1 - N^{-\phi}/2$. Hence, the time required by any fixed $\mathcal{P}$-packet participating in I to be successful is at most

$$\gamma \cdot (1+\epsilon)\lambda T + \delta \log^\psi N + O(\log^\alpha N) = (1-\epsilon^2)T + \delta \log^\psi N + O(\log^\alpha N)$$

with probability at least $1 - N^{-\phi}$. This time bound is at most $T(1 - \epsilon^2/2)$ if $T \geq \frac{2}{\epsilon^2}(\delta \log^\psi N + O(\log^\alpha N))$. (The remaining $T \cdot \epsilon^2/2$ time steps will be important when considering the failed packets.)

Now suppose that $\epsilon > 7/8$. In this case, $\lambda < 1/(8\gamma)$. Thus, according to the Chernoff bounds the probability that the congestion caused by $\mathcal{P}$-packets exceeds $T/(4\gamma)$ can be made as small as $N^{-\phi}$ for any constant $\phi > 0$ if $T = \Theta(\phi \log N)$ is chosen large enough. Set $c_T = T/(4\gamma)$ and $d_T = \log^\psi N$. Analogous to above, the routing time required by the $\mathcal{P}$-packets participating in I to be successful can be made as small as $T(1-\epsilon^2/2)$ with probability of at least $1-N^{-\phi}$ if $T = \Theta(\delta \log^\psi N + \log^\alpha N)$ is chosen large enough.

In both cases for ϵ, the bound for T is (asymptotically) minimal for $\beta + \psi = \alpha$. So altogether $T = \Theta(\frac{1}{\epsilon^2}(\phi \log N + \log^\alpha N))$ suffices to guarantee that any fixed $\mathcal{P}$-packet successfully manages a T-interval with probability at least $1 - N^{-\phi}$.

Case 2. $\gamma = \Theta(1)$ and $\delta = C^\beta$ for some constant $0 < \beta < 1$.

If $\epsilon \leq 15/16$, we set $c_T = (1+\epsilon)\lambda T$ and $d_T = \log^\psi N$. As in Case 1, the probability is at most $N^{-\phi}/2$ that the congestion in I exceeds c_T if $T = \Theta(\frac{\phi}{\epsilon^2} \log N)$ is sufficiently large. In this case, the time required by any fixed $\mathcal{P}$-packet participating in I to be successful is at most

$$\gamma \cdot (1+\epsilon)\lambda T + \delta \log^\psi N + \log^\alpha N \leq (1-\epsilon^2)T + T^\beta \log^\psi N + \log^\alpha N$$

with probability at least $1 - N^{-\phi}$. It holds that

$$\log^\alpha N \leq \frac{\epsilon^2}{4} \cdot T \quad \text{if} \quad T \geq \frac{4}{\epsilon^2} \cdot \log^\alpha N$$

and

$$T^\beta \log^\psi N \leq \frac{\epsilon^2}{4} \cdot T \quad \text{if} \quad \psi = \alpha(1-\beta) \text{ and } T \geq (\epsilon/2)^{-2/(1-\beta)} \log^\alpha N.$$

Thus, if $T = \Theta((\epsilon/2)^{-2/(1-\beta)} \log^\alpha N)$ is sufficiently large, then

$$(1-\epsilon^2)T + T^\beta \log^\psi N + \log^\alpha N \leq (1 - \epsilon^2/2)T.$$

Hence $T = \Theta((\epsilon/2)^{-2/(1-\beta)}(\phi \log N + \log^\alpha N))$ suffices to ensure that any fixed $\mathcal{P}$-packet successfully manages a T-interval with probability at least $1 - N^{-\phi}$.

If $\epsilon > 15/16$, we set $c_T = T/(8\gamma)$ and $d_T = \log^\psi N$. Similar to Case 1, the probability is at most $N^{-\phi}/2$ that the congestion in I exceeds c_T if $T = \Theta(\phi \log N)$ is sufficiently large. In this case, the time required by any fixed $\mathcal{P}$-packet participating in I to be successful is at most

$$\gamma \cdot T/(8\gamma) + \delta \log^\psi N + \log^\alpha N \leq T/8 + T^\beta \log^\psi N + \log^\alpha N,$$

with probability at least $1 - N^{-\phi}$. This is at most $(1 - \epsilon^2/2)T$ if

$$T^\beta \log^\psi N \leq T/4 \quad \text{and} \quad \log^\alpha N \leq T/8,$$

which is true if $\psi = \alpha(1-\beta)$ and $T = \Theta((\epsilon/2)^{-2/(1-\beta)} \log^\alpha N)$ is sufficiently large. Hence, if $T = \Theta((\epsilon/2)^{-2/(1-\beta)}(\phi \log N + \log^\alpha N))$, then also in this case any fixed $\mathcal{P}$-packet successfully manages a T-interval with probability at least $1 - N^{-\phi}$.

Case 3. The runtime of $\mathcal{P}$ is at most $C \cdot D$.

Assume that $\lambda = \frac{1-\epsilon}{\log N}$. If $\epsilon \leq 7/8$, we set $c_T = (1+\epsilon)\lambda T$ and $d_T = \log N$. As above, it can be shown that the probability is at most $N^{-\phi}$ that the congestion in I exceeds c_T if $T = \Theta(\frac{\phi}{\epsilon^2} \log^2 N)$ is sufficiently large. In this case, the time required by any fixed $\mathcal{P}$-packet participating in I to be successful is at most

$$(1+\epsilon)\lambda T \cdot \log N = (1-\epsilon^2)T$$

with probability at least $1 - N^{-\phi}$.

If $\epsilon > 7/8$, then we set $c_T = T/(2 \log N)$ and $d_T = \log N$. If $T = \Theta(\phi \log^2 N)$ is sufficiently large, the probability is at most $N^{-\phi}$ that the congestion in I exceeds c_T. In this case, the time required by any fixed $\mathcal{P}$-packet participating in I to be successful is at most

$$\frac{T}{2 \log N} \cdot \log N \leq (1 - \epsilon^2/2)T$$

with probability at least $1 - N^{-\phi}$. □

It follows for Lemma 5.3(1) that if a packet following a path of length d successfully manages all T-intervals, its runtime is bounded by

$$O\left(\left(\frac{d}{\log^\psi N} + 1\right) \frac{1}{\epsilon^2} \log^\alpha N\right) = O\left(\frac{1}{\epsilon^2} \left(\delta d + \log^\alpha N\right)\right).$$

Similarly, for Case 2, the runtime of a packet following a path of length d that successfully manages all T-intervals is bounded by

$$O\left(\left(\frac{d}{\log^\psi N} + 1\right) (\epsilon/2)^{-\frac{2}{1-\beta}} \log^\alpha N\right) = O\left((\epsilon/2)^{-\frac{2}{1-\beta}} \left(d \log^{\alpha\beta} N + \log^\alpha N\right)\right),$$

and for Case 3 the runtime of the packet is bounded by

$$O\left(\left(\frac{d}{\log N} + 1\right) \frac{1}{\epsilon^2} \log^2 N\right) = O\left(\frac{1}{\epsilon^2} \left(d \log N + \log^2 N\right)\right).$$

These time bounds match the time bounds given in Theorem 5.1 if a packet never fails. According to Lemma 5.3, the probability that a packet fails in some T-interval can be made polynomially small in N if λ is sufficiently small. Hence, in this case all time bounds hold w.h.p. It remains to bound the runtimes of failed packets to be able to compute the expected time a packet needs to reach its destination.

Bounding the routing time of failed packets. Next we bound the runtime of failed packets under the assumption that the injection rate is sufficiently small, that is, each individual packet is successful in every T-interval, w.h.p. For this, we will show how to apply Lemma 5.2 to bound the expected routing time of a failed packet by N^c for some constant c. Combined with the upper bound of $N^{-\phi}$ on the probability that a packet fails, this will result in an expected routing time not much larger than the time that the packet requires if it does not fail in any of its T-intervals, provided that $\phi \geq c$. This yields the bounds for the expected routing time in Theorem 5.1 and implies that $\mathcal{P}'$ is stable.

Now we show how to apply Lemma 5.2. First, we change the situation that only every $\frac{2}{\epsilon^2}$th step can be used by a failed packet to a situation in which every time step can be used by a failed packet. For this, we replace each generator g by $2/\epsilon^2$ generators $g_1, \ldots, g_{2/\epsilon^2}$, where g_i is responsible for the simulation of the behavior of

g at time steps t with $(t \bmod 2/\epsilon^2) + 1 = i$. Furthermore, we assume each time step to represent now $2/\epsilon^2$ time steps in the original situation. Let us consider some fixed edge e. For any generator g and time step t in the new situation, let the binary random variable Y_t^g be 1 if and only if g generates a packet at step t that intends to cross e and that fails in some T-interval of the original situation. From Lemma 5.3 we know that $\Pr[Y_t^g = 1]$ can be made as small as $N^{-\phi}$ for any constant $\phi > 0$.

If the probabilities $\Pr[Y_t^g = 1]$ were independent for different Y_t^g, we could model the injection of failed packets as a simple stochastic injection model with injection rate $\lambda = \Pr[Y_t^g = 1] \cdot 2/\epsilon^2$. For $\Pr[Y_t^g = 1] \cdot 2/\epsilon^2 \leq 1/D(\mathcal{S})$, it would follow directly from Lemma 5.2 (choose $K = 0$ and $\lambda = 1/D(\mathcal{S})$) that the routing time of a failed packet is less than N^ϕ if ϕ is sufficiently large.

Coping with the dependencies. Unfortunately, there can be high dependencies among failures of packets. In order to incorporate these dependencies in the model of Lemma 5.2, we construct a dependency graph $G = (V, E)$ that has a node (g, t) for each random variable Y_t^g, and in which two nodes (g, t) and (g', t') are connected in G if and only if $t - 2T \cdot (D(\mathcal{S})+1) \cdot (\epsilon^2/2) \leq t' \leq t + 2T \cdot (D(\mathcal{S})+1) \cdot (\epsilon^2/2)$. Since there are $2N/\epsilon^2$ generators, the maximum degree of G is at most $(4T \cdot (D(\mathcal{S})+1) + 2/\epsilon^2) \cdot N$, which is polynomial in N. In order to apply Lemma 5.2, we need to show that for any independent set $S \subseteq V$, the probability that $Y_t^g = 1$ for all $(g, t) \in S$ is at most $N^{-\phi|S|}$ for any constant $\phi > 0$ (depending on T).

Recall that the proof of Lemma 5.3 bounds the failure probability of a packet within a T-interval I solely by considering the injection events of packets that could have still been successful at the beginning of I and the behavior of $\mathcal{P}$ within I. Further recall that the proof of Lemma 5.3 only uses the congestion (which is upper bounded by the injection events) in I to obtain a probability bound for the success of $\mathcal{P}$. Hence the space $\Omega_{g,t}$ of relevant outcomes that need to be considered to obtain a probability of at most $N^{-\phi}$ for the random variable Y_t^g to be 1 can be limited to contain solely injection events of packets that can participate in $\mathcal{P}$ in some T-interval together with the packet generated by g at step t. Since a packet can be alive without a failure for at most $T \cdot (D(\mathcal{S})+1) \cdot (\epsilon^2/2)$ time steps, the outcome spaces of any two random variables Y_t^g and $Y_{t'}^{g'}$ with either $t' < t - 2T \cdot (D(\mathcal{S})+1) \cdot (\epsilon^2/2)$ or $t' > t + 2T \cdot (D(\mathcal{S})+1) \cdot (\epsilon^2/2)$ must be disjoint (that is, they do not contain a common injection event (g'', t'')). Thus, using the proof of Lemma 5.3, the probability that both of these random variables are 1 can be shown to be at most $N^{-2\phi}$. The same argument extends to any set of random variables that form an independent set in G.

In order to set the remaining parameters in the model of Lemma 5.2, we set λ equal to the given injection rate and $K = 0$. (Successful packets cannot interfere with failed packets.) According to Lemma 5.2, for these parameters the expected routing time of any failed packet is at most N^ϕ if ϕ is large enough. This completes the proof that for a small enough λ, even when considering failures, the expected routing time of any fixed packet is within the time bounds given in Theorem 5.1.

Recovery. We show that $\mathcal{P}'$ recovers very quickly from any worst-case scenario. Clearly, the SIS rule instantly recovers from any worst-case scenario because younger packets are always preferred. Similar, we show for $\mathcal{P}'$ that after a certain amount of time any configuration of the network has *no* influence on the runtime of the newly generated packets anymore, concerning our probability bounds.

Consider any worst-case scenario for the injection of packets that ends at time step t_0. Since each packet has to traverse d_T edges in each T-interval in order to remain successful, $R = (\lceil \frac{D(\mathcal{S})}{d_T} \rceil + 1) \cdot T$ time steps after t_0 there can be no successful packet

anymore that was generated at time step t_0 or earlier. This implies that afterwards the congestion in a T-interval is based only on packets injected after the worst-case scenario, which according to our stochastic injection model is independent of whatever happened during or before a worst-case scenario. Since in the proof of Lemma 5.3 we used the worst-case assumption that all packets have been successful so far to upper bound the congestion in a T-interval, all probability bounds in Lemma 5.3 are again valid R steps after the worst-case scenario. That is, the probability of a packet to fail in a T-interval is again polynomially small in N. This ensures that sufficiently few packets fail. Once a packet fails, its age is determined by its injection time. Since SIS is used for the failed packets, a failed packet generated at a time step $t > t_0$ cannot be blocked by the packets generated during the worst-case scenario. Hence $\mathcal{P}'$ recovers after $(\lceil \frac{D(\mathcal{S})}{d_T} \rceil) + 1) \cdot T$ time steps. Substituting the right T for each of the three cases (see Lemma 5.3) yields the recovery time given in Theorem 5.1.

5.3. Analysis of $\mathcal{P}'$ for pure $\mathcal{P}$. The analysis for the successful packets is the same as above. However, since we do not have reserved time slots for the failed packets, we need a different analysis for them if $\mathcal{P}$ is pure.

Bounding the routing time of failed packets. Suppose that the injection rate is sufficiently small, that is, each individual packet is successful in every T-interval, w.h.p. For this, we will show how to apply Lemma 5.2 to bound the routing time of a failed packet by N^c for some constant c. Combined with the bound of $N^{-\phi}$ on the probability that a packet fails, this will result in an expected routing time not much larger than the time that the packet requires if it does not fail in any of its T-intervals, provided that $\phi \geq c$. This would yield the bounds for the expected routing time in Theorem 5.1 and would imply that $\mathcal{P}'$ is stable.

Now we show how to apply Lemma 5.2. Let us consider some fixed edge e. For any generator g and time step t, let the binary random variable Y_t^g be 1 if and only if g generates a packet at step t that intends to cross e and that fails in some T-interval. From Lemma 5.3 we know that $\Pr[Y_t^g = 1]$ can be made as small as $N^{-\phi}$ for any constant ϕ. Furthermore, similar to the case that $\mathcal{P}$ is nonpure, the event $Y_t^g = 1$ causes at most $(4T \cdot (D(\mathcal{S})+1)+1) \cdot N$ other random variables $Y_{t'}^{g'}$ to have a probability of $\Pr[Y_{t'}^{g'} = 1] > N^{-\phi}$, namely, those with $t - 2T \cdot (D(\mathcal{S})+1) \leq t' \leq t + 2T \cdot (D(\mathcal{S})+1)$. In order to incorporate these dependencies in the model of Lemma 5.2, we again construct a dependency graph $G = (V, E)$ that has a node (g, t) for each random variable Y_t^g, and in which two nodes (g, t) and (g', t') are connected in G if and only if $t - 2T \cdot (D(\mathcal{S}) + 1) \leq t' \leq t + 2T \cdot (D(\mathcal{S}) + 1)$. Thus the maximum degree of G is $(4T \cdot (D(\mathcal{S}) + 1) + 1) \cdot N$. Furthermore, in the model of Lemma 5.2 we set λ equal to the given injection rate and $K = D(\mathcal{S}) \cdot T$. (Successful packets live for at most $D(\mathcal{S}) \cdot T$ time steps.) According to Lemma 5.2, in this case the expected routing time of any failed packet is at most N^{ϕ} if ϕ is large enough. This completes the proof that, also for a pure $\mathcal{P}$, for a small enough λ the expected routing time of any fixed packet is within the time bounds given in Theorem 5.1.

Stability for any $\lambda < 1$. Next we consider the case that λ is arbitrarily close to 1. In this situation $\mathcal{P}$ might not be good enough to ensure that a packet manages its T-interval with high probability. However, from Lemma 5.2 (set $K = D(\mathcal{S}) \cdot T$) it directly follows that nevertheless $\mathcal{P}'$ remains stable for any $\lambda < 1$, although the runtime of a packet might get exponential in the length of its path.

6. Adapting our results to the adversarial model. Consider any bounded adversary of rate (w, λ). Set $\epsilon = 1 - \lambda$. For each injected packet p we choose uniformly

and independently at random an initial delay of δ_p from the set $\{0, \ldots, K-1\}$, where $K = (k \cdot D + w)/\epsilon^2$ and D is the length of a longest possible path. k will be specified later, as it depends on the routing protocol for which we want to adapt the results. After waiting δ_p time steps in its injection buffer, p chooses the actual time step as its *new injection time* and participates in whatever routing protocol chosen. We say that every packet with new injection time t *touches* an edge e at step t' if e is the ith edge on the routing path of p and $t' = t + k \cdot i$. Let λ' denote the maximum, over all edges and time steps, of the expected number of packets that touch an edge in a time step. Then the following lemma holds.

LEMMA 6.1. $\lambda' \leq 1 - \epsilon/2$.

Proof. Consider some fixed edge e and time step t. The maximum number of packets that touch edge e at step t is at most

$$(1-\epsilon)w \cdot \left\lceil \frac{k \cdot D + K}{w} \right\rceil \leq (1-\epsilon) \cdot (k \cdot D + w) \cdot (1 + 1/\epsilon^2).$$

Since each of these packets chooses a random initial delay out of a range of $[(k \cdot D + w)/\epsilon^2]$, it holds that

$$\begin{aligned} \lambda' &\leq \frac{(1-\epsilon) \cdot (k \cdot D + w) \cdot (1 + 1/\epsilon^2)}{(k \cdot D + w)/\epsilon^2} \\ &\leq (1-\epsilon) \cdot (1 + \epsilon^2) \leq 1 - \epsilon/2 \end{aligned}$$

for any $\epsilon \in [0, 1]$. ∎

As is not difficult to verify, this lemma can be used to transfer all results presented in this paper to the adversarial model. For the ghost packet protocol on general networks, we choose $k = 0$. Then it follows from Lemma 6.1 that the expected number of packets with a fixed rank r that traverse an edge e is at most $1 - \epsilon/2$. Hence, the analysis for the ghost packet protocol on general networks holds without any further change also for the adversarial model. We have to add only w/ϵ^2 to the delay bound of the packets to include the random initial delay.

For the growing rank protocol we choose $k = m$. Then it follows from Lemma 6.1 that, for any edge e and rank r, the expected number of packets with rank r that traverse e is at most $1 - \epsilon/2$. This allows us to use the same analysis as that for the proof of Theorem 4.1 to show that the delay of any packet is bounded by $O(\frac{1}{\epsilon^2}(m \cdot D + w) + m \cdot \log N) = O(m^2 \cdot D + m \cdot (w + \log N))$, w.h.p.

For the black box transformation it suffices to choose $k = T(1/d_T + 1/D)$. Then we get that, for any edge e and T-interval I, the expected number of packets that intend to cross e in I is at most $T(1 - \epsilon/2)$. In this case, the same analysis as in section 5 can be used to show that, for instance, for $\gamma = \Theta(1)$ and $\delta = \Theta(\log^\beta N)$, the delay of any packet is bounded by $O(\frac{1}{\epsilon^2}(\frac{1}{\epsilon^2}\delta D + w + \log^\alpha N))$, w.h.p., if $\lambda \leq (1-\epsilon)/\gamma$.

7. Open problems. In this paper, we presented transformations of well-known static routing algorithms (such as the ghost packet protocol and the growing rank protocol) into efficient dynamic routing algorithms. We obtained, for instance, a dynamic routing algorithm for arbitrary leveled networks that requires only buffers of size depending on the injection rate and the maximum degree of the network to be stable up to a maximum possible injection rate. This algorithm, however, uses ghost or control packets. Although our analysis implicitly shows that these packets are sent very rarely, the question arises whether or not control packets can be avoided completely.

Furthermore, we presented a black box transformation scheme applicable to every static, oblivious routing algorithm. Our results show that it might be important for static routing protocols to know the exact constant in front of the C in the runtime bound, since this determines up to which injection rate the resulting dynamic protocol ensures a small delay for the packets. It is an interesting question how large this constant is for the known static routing protocols, and whether there exist static routing protocols with runtime $C + O(D + \log N)$.

REFERENCES

[1] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu, *Universal stability results for greedy contention-resolution protocols*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 380–389.

[2] M. Andrews, A. Fernández, M. Harchol-Balter, T. Leighton, and L. Zhang, *General dynamic routing with per-packet delay guarantees of $O(\mathit{distance} + 1/\mathit{session\ rate})$*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 294–302.

[3] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. P. Williamson, *Adversarial queueing theory*, in Proceedings of the 28th ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 376–385.

[4] A. Z. Broder, A. M. Frieze, and E. Upfal, *A general approach to dynamic packet routing with bounded buffers*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 390–399.

[5] E. G. Coffman, Jr., N. Kahale, and F. T. Leighton, *Processor-ring communication: A tight asymptotic bound on packet waiting times*, SIAM J. Comput., 27 (1998), pp. 1221–1236.

[6] R. Cypher, F. Meyer auf der Heide, C. Scheideler, and B. Vöcking, *Universal algorithms for store-and-forward and wormhole routing*, in Proceedings of the 28th ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 356–365.

[7] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*: *Arrays, Trees, Hypercubes*, Morgan-Kaufmann, San Mateo, CA, 1992.

[8] F. T. Leighton, B. M. Maggs, A. G. Ranade, and S. B. Rao, *Randomized routing and sorting on fixed-connection networks*, J. Algorithms, 17 (1994), pp. 157–205.

[9] F. T. Leighton, B. M. Maggs, and S. B. Rao, *Packet routing and job-shop scheduling in $O(\mathit{congestion} + \mathit{dilation})$ steps*, Combinatorica, 14 (1994), pp. 167–186.

[10] F. Meyer auf der Heide and B. Vöcking, *A packet routing protocol for arbitrary networks*, in Proceedings of the 12th Symposium on Theoretical Aspects of Computer Science, Munich, Germany, 1995, pp. 291–302.

[11] F. Meyer auf der Heide and B. Vöcking, *Shortest paths routing in arbitrary networks*, J. Algorithms, 31 (1999), pp. 105–131.

[12] R. Ostrovsky and Y. Rabani, *Universal $O(\mathit{congestion} + \mathit{dilation} + \log^{1+\epsilon} n)$ local control packet switching algorithms*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 644–653.

[13] Y. Rabani and E. Tardos, *Distributed packet switching in arbitrary networks*, in Proceedings of the 28th ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 366–375.

[14] A. G. Ranade, *How to emulate shared memory*, J. Comput. System Sci., 42 (1991), pp. 307–326.

[15] G. D. Stamoulis and J. N. Tsitsiklis, *The efficiency of greedy routing in hypercubes and butterflies*, in Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, Hilton Head, SC, 1991, pp. 248–259.

[16] C. Scheideler and B. Vöcking, *Universal continuous routing strategies*, in Proceedings of the 8th ACM Symposium on Parallel Algorithms and Architectures, Padua, Italy, 1996, pp. 142–151.

[17] C. Scheideler and B. Vöcking, *Universal continuous routing strategies*, Theory Comput. Systems, 31 (1998), pp. 425–449.

[18] C. Scheideler and B. Vöcking, *From static to dynamic routing: Efficient transformations of store-and-forward protocols*, in Proceedings of the 31th ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 215–224.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1156–1168

DETERMINING CONSENSUS NUMBERS*

ERIC RUPPERT†

Abstract. Conditions on a shared object type T are given that are both necessary and sufficient for wait-free n-process consensus to be solvable using objects of type T and registers. The conditions apply to two large classes of deterministic shared objects: read-modify-write objects [C. P. Kruskal, L. Rudolph, and M. Snir, *ACM Trans. Prog. Lang. Syst.*, 10 (1988), pp. 579–601] and readable objects, which have operations that allow processes to read the state of the object. These classes include most objects that are used as the primitives of distributed systems. When the sequential specification of T is finite, the conditions may be checked in a finite amount of time to decide the question "Is the consensus number of T at least n?" The conditions are also used to provide a clear proof of the robustness of the consensus hierarchy for read-modify-write and readable objects.

Key words. consensus, distributed computing, asynchronous, wait-free, shared memory, read-modify-write, readable, consensus hierarchy, robust

AMS subject classifications. 68Q22, 68Q10

PII. S0097539797329439

1. Introduction.

A shared-memory distributed system consists of a number of processes that communicate with one another by performing operations on shared data objects. This paper studies asynchronous systems where processes take steps at arbitrarily varying speeds. The wait-free model of fault tolerance is used: each nonfaulty process must complete its task correctly even if any number of processes experiences halting failures. The power of a system to solve problems depends on the types of shared data objects that are available. Determining whether a given collection of object types can be used to solve a given problem is therefore of great importance in the design of distributed systems. A problem may, itself, be modelled as a shared object type: processes pass their input to the object, and the object returns the desired output. Thus, a question about computability may be formulated as a question about implementations: Can one object type be implemented using a given set of primitive object types? Herlihy [10] showed that the consensus problem, in which each process begins with an input and all nonfaulty processes must agree on one of the input values, plays a central role in the study of the power of object types. He proved that objects that solve the consensus problem for n processes can be used, along with read/write registers, to obtain a wait-free implementation of any object for n or fewer processes. This leads to the idea of classifying object types according to their consensus number [10, 14]. An object type T *has consensus number* n if n-process consensus can be solved by using objects of type T and read/write registers, but $(n+1)$-process consensus cannot. If there is no such n, the consensus number of T is ∞. Similarly, the consensus number of a set $\mathcal{S}$ of types is the largest n for which n processes can solve consensus using objects whose types are in $\mathcal{S} \cup \{\text{register}\}$, or ∞ if no such n exists.

*Received by the editors November 3, 1997; accepted for publication (in revised form) April 14, 2000; published electronically September 20, 2000. A preliminary version of this paper appeared at the 16th ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, pp. 93–99. This research was supported by the Natural Sciences and Engineering Research Council of Canada.

http://www.siam.org/journals/sicomp/30-4/32943.html

†Department of Computer Science, Brown University, Providence, RI 02912 (ruppert@cs.brown.edu).

This paper addresses the problem of determining whether a given deterministic object type has consensus number n. Two important classes of objects that include most of the objects considered as primitives for distributed systems are studied: read-modify-write (RMW) objects [16] and readable objects. It is reasonable to assume that processes will be able to access the information stored in the data object in some simple way. If processes may read the information directly, without altering the object's state, then the object is called readable. If the operation that reads the information in the data object is combined with an update operation, so that the read and update occur as a single atomic action, the object may be modelled as an RMW object.

A *readable* object O is modelled by an I/O automaton whose state set is the Cartesian product $Q = \bigtimes_{k\in\Gamma} Q_k$, where Γ is an index set and Q_k is a set for each $k \in \Gamma$. Neither Γ nor Q_k must be finite. For each $k \in \Gamma$, processes may execute the operation $read_k$, which returns component k of the current state of O. In addition to the *read* operations, the object may have an arbitrary set of other (deterministic) operations defined on it. Each operation can update any set of components of the state, and the set of components that are updated by an operation may depend on the current state of the object. Any object type with a *read* operation that returns the entire state of the object is readable; in this case, $|\Gamma| = 1$. An array of m 1-bit registers is a readable object, with $\Gamma = \{1, \ldots, m\}$ and $Q_k = \{0, 1\}$ for all $k \in \Gamma$. Readable objects include arrays of registers whose elements can be read, copied, or swapped atomically. An array of unbounded registers that can be read or used for indirect addressing is another example of a readable object; in this case the state set would be $Q = \bigtimes_{k\in\mathbb{N}} \mathbb{N}$ and an indirect write to component i updates the component indexed by the number stored in component i.

An RMW object type allows various kinds of *RMW* operations to be performed on it. Let V be the state set of the RMW object. Any function $f : V \to V$ defines an *RMW* operation, denoted RMW_f, that reads the current value, x, of the RMW object, updates the value to $f(x)$, and returns x. For example, a *fetch&increment* operation applies the function $f(x) = x + 1$, and a *read* operation applies the identity function. An RMW object type is defined by the set F of functions mapping V to itself that may be applied by the *RMW* operations. This class of objects includes compare&swap, test&set, and fetch&add variables.

Previously, ad hoc arguments have been used to determine whether particular object types can be used along with registers to solve n-process consensus. An exception was Herlihy's proof [10] that 2-process consensus can be solved using registers and RMW objects, where the *RMW* operations apply functions from the set F if and only if F contains a function different from the identity. Herlihy also gave a necessary condition on the set F to describe when RMW objects can be used with registers to solve 3-process consensus. In this paper, a decision procedure is developed to determine whether given readable and RMW types with finite specifications can be used to solve n-process consensus for any n. Jayanti and Toueg [15] showed that this question is undecidable for arbitrary object types that are allowed to have infinite state sets. It will be shown here that an RMW or readable type T can be used together with registers to solve the consensus problem among n processes if and only if T satisfies certain conditions which are decidable for types that have finite specifications. An object type that satisfies the conditions is called *n-discerning*. This characterization has been useful for studying the consensus numbers of multi-objects and transactional objects, where processes can access more than one object in an atomic action [21, 22].

Some work has been done on deciding whether a given task can be solved using some particular types of shared objects. Biran, Moran, and Zaks [4] showed that one can decide whether a given task can be solved in a message-passing system if at most one process may fail. Chor and Moscovici [7] gave a decidable characterization of tasks that can be solved by randomized algorithms that use registers only. Gafni and Koutsoupias [9] showed that there is no algorithm which determines whether a given task for three processes can be solved by read/write registers in a wait-free manner. Herlihy and Rajsbaum [11] gave decidability results for the question of whether a given task can be solved using various types of objects (registers, consensus objects, and set consensus objects) in the presence of t process failures.

The proof that the property of being n-discerning is sufficient for the solvability of n-process consensus will also provide upper bounds on the resources required to solve the consensus problem. If an RMW or readable object type T can be used together with registers to solve consensus among n processes, then there is a consensus protocol that uses at most $n-1$ objects of type T and $2(n-1)$ registers. Furthermore, if T is an RMW object type, each process takes $O(n)$ steps in this protocol. It also follows from the construction of the protocol that n-process consensus can be solved using $2(n-1)$ registers and $n-1$ objects, $X_2, \ldots, X_n$, where X_i's type is i-discerning, for each i.

The characterization of n-discerning types will be used in section 5 to obtain a clear proof that the consensus hierarchy is robust [14] for RMW and readable objects. This means that if n-process consensus can be solved using RMW and readable objects of types $T_1, \ldots, T_r$ and registers, then n-process consensus can also be solved using only registers and objects of type T_i for some i. This robustness property allows the consensus number of a set of RMW and readable types to be determined by finding the consensus number of each type separately. Borowsky, Gafni, and Afek [5] claimed that the consensus hierarchy is robust for all deterministic objects. A full version of their paper is not yet available.

The n-discerning conditions for readable objects will also be used to generalize a result about atomic snapshot objects. A snapshot object can be thought of as a finite array of registers with an additional *scan* operation that reads the entire array at once. It is known that the addition of the *scan* operation does not increase the power of an array of registers to solve consensus, since the snapshot object can be implemented from ordinary registers [1, 2, 3]. Here, it will be shown that the addition of an atomic *scan* operation does not increase the power of *any* readable object type to solve consensus.

2. Preliminaries. An object type is defined using a sequential specification, which describes the operations that may be performed on the object and the responses the object should return if the operations are performed sequentially. Formally, an object can be specified as an I/O automaton (see [18]). Each operation causes a state transition and returns a response. If the state transition and response are uniquely determined by the current state of the object and the operation applied, then the object is *deterministic*. When an object is accessed by more than one process concurrently, its behavior can be specified by insisting that operations appear to occur instantaneously at some time between their invocations and their responses. Such an object is called *linearizable* [12]. This paper deals only with deterministic, linearizable objects. For simplicity, it is assumed that all objects are *oblivious*: every process can invoke every operation, and, for any operation, the state transition and response do not depend on the process that invokes the operation.

It is assumed that the designer of a consensus protocol may choose the initial states of the shared objects. There is no real loss of generality in this assumption; Borowsky, Gafni, and Afek [5] showed that if consensus can be solved when the initial states are chosen by the programmer, then consensus can be solved using objects initialized to a particular state, assuming there is some sequence of operations that will move the object from the given initial state into any other state.

The systems studied here are completely asynchronous, so that algorithms must exhibit correct behavior regardless of the way in which the steps of different processes are interleaved by a scheduler. Algorithms are required to be wait-free [10]. This means that the algorithm executed by each nonfaulty process must work correctly even if other processes experience halting failures.

An implementation of an object of type T from objects $O_1, \ldots, O_m$ is a protocol that uses only the shared objects $O_1, \ldots, O_m$. This protocol consists of an algorithm *apply*(*op*) for each process that can simulate every operation *op* on an object of type T. The protocol should also specify the initial states for $O_1, \ldots, O_m$ to be used for any possible initial state of the simulated object. If each process performs the *apply* routine repeatedly, the responses returned should appear as if the operations were carried out atomically on an object of type T. All implementations are assumed to be wait-free.

The consensus number of a set of object types is defined in terms of its ability to solve the consensus problem. For the n-consensus problem, n processes each begin with a private input value, and each nonfaulty process outputs a value in a wait-free manner. The output values must all be the same, and every output must be the input value of some process. These two conditions are called *consistency* and *validity*, respectively.

The proofs that the property of being n-discerning is necessary for the solvability of n-consensus are bivalency arguments. This type of proof was introduced by Fischer, Lynch, and Paterson [8]. The following terminology will be used in the bivalency arguments. The *configuration* of a protocol at any moment in its execution consists of the state of every shared object, together with the internal state of every process. Two configurations are *indistinguishable to process* P if the state of each shared object and the internal state of P is the same in the two configurations. A configuration of a consensus protocol is x-*valent* if all nonfaulty processes decide on the value x in all executions continuing from that configuration. A configuration is called *univalent* if it is x-valent for some x, and *multivalent* otherwise. A configuration is *critical* if it is multivalent and the next step by any process will move the system into a univalent configuration. The value that would be decided if a particular process takes the next step after a critical configuration is called the *critical value* of that process.

3. Solving consensus with RMW objects. Consider an RMW type T with state set V whose *RMW* operations can apply functions from the set F. This section describes the power of a distributed system with $n \geq 2$ processes, $P_1, \ldots, P_n$, to solve consensus using objects of type T and registers. The sets V and F need not be finite.

Let v_0 be a value in V. Partition the set of processes $\{P_1, \ldots, P_n\}$ into two nonempty teams, A and B. Associate a function $f_i \in F$ with each process P_i. The functions f_i need not be distinct. The type T is called n-*discerning* if there exist choices for $v_0, A, B, f_1, \ldots, f_n$ so that, in any schedule in which each process P_i applies the single operation RMW_{f_i} to an object X which initially has value v_0, every process can determine (from the value returned by its operation) whether a process from team A or a process from team B was the first process to take a step in the schedule. Such a type T is called n-discerning since processes can easily use an object of type T to

discern the difference between schedules that start with a process on team A from those that begin with a process on team B. This description is formalized in the following definition. The notation $f \circ g$ is used to denote functional composition: $(f \circ g)(x) = f(g(x))$.

DEFINITION 1. *Let $n \geq 2$. The RMW type defined by the state set V and the set F of functions is n-discerning if there exist*

- *$v_0 \in V$,*
- *a partition of the set of processes $\{P_1, \ldots, P_n\}$ into two nonempty teams A and B, and*
- *a function $f_i \in F$ for each process P_i*

such that

I. *for all $j \in \{1, \ldots, n\}, V_{A,j} \cap V_{B,j} = \emptyset$,*

II. *for all $P_j \in B, v_0 \notin V_{A,j}$, and*

III. *for all $P_j \in A, v_0 \notin V_{B,j}$, where*

$V_{A,j} = \{(f_{i_\alpha} \circ \cdots \circ f_{i_1})(v_0) \mid P_{i_1} \in A,\ \alpha \geq 1$, and $i_1, \ldots, i_\alpha$ are distinct process indices, not including $j\}$, and $V_{B,j} = \{(f_{i_\alpha} \circ \cdots \circ f_{i_1})(v_0) \mid P_{i_1} \in B,\ \alpha \geq 1$, and $i_1, \ldots, i_\alpha$ are distinct process indices, not including $j\}$.

Suppose each process performs its assigned operation to the object X, which is initialized with the value v_0. Consider a process P_j on team A. The set $V_{A,j}$ contains the values that P_j can see when it accesses object X if some other process on team A performs the first step. (The set $V_{A,j}$ will be empty if P_j is the only process on team A. It may be the case that v_0 is in $V_{A,j}$.) The set $V_{B,j}$ is the set of responses that P_j can receive if a process from team B takes the first step. Thus, condition I ensures that P_j can distinguish, using the response it receives, executions starting with another process on team A from those in which a process on team B takes the first step. Condition III ensures that P_j can distinguish executions in which P_j itself takes the first step from those executions in which a process from team B takes the first step. (Similarly, if the process P_j is on team B, conditions I and II guarantee that it can tell which team took the first step.)

It will be shown in Theorems 3 and 9 that RMW objects of type T can be used with registers to solve n-consensus if and only if T is n-discerning.

LEMMA 2. *If S_0 is a critical configuration of an n-process consensus protocol where the next step of every process accesses the same RMW object X of type T, then T is n-discerning.*

Proof. Suppose each process P_i applies the operation RMW_{f_i} to X during its first step after S_0. Let v_0 be the value of X in the configuration S_0. Let a be the critical value of one of the processes. Let A be the set of processes with critical value a, and let B contain the rest of the processes. Team A is nonempty by construction. Since S_0 is multivalent, team B must also be nonempty.

Condition I of Definition 1 must hold for these values of $v_0, A, B, f_1, \ldots, f_n$. Otherwise, let j be a process index such that $V_{A,j} \cap V_{B,j}$ is nonempty. Then, X has the same value in two configurations that can be reached from S_0 by sequences of steps in which each process takes at most one step, process P_j takes no steps, and the first steps in the two sequences are taken by processes on opposite teams. These two configurations are indistinguishable to P_j. A solo execution by P_j from either of these configurations would therefore lead to the same decision value, contradicting the definitions of teams A and B.

Condition II must also hold. Otherwise, let j be the index of a process on team B such that $v_0 \in V_{A,j}$. Then, some sequence of processes, not including process $P_j \in$

B and starting with a process on team A, could each take a step to arrive at a configuration that P_j cannot distinguish from S_0. A solo execution of P_j from these two configurations would then lead to the same decision, contradicting the fact that $P_j \in B$.

The argument that condition III holds is symmetric. □

This lemma can be combined with a bivalency argument to prove that the conditions for being n-discerning are necessary for solving n-process consensus using RMW objects.

THEOREM 3. *If the n-process consensus problem can be solved using registers and RMW objects of type T, then T must be n-discerning.*

Proof. Suppose there is some protocol for n-process consensus using registers and RMW objects of type T. If only one process is scheduled to take steps in an execution, the process must output its own input value. Thus, any initial configuration where processes begin with different input values is multivalent. It follows that there is a critical configuration S_0 of the protocol. Otherwise, one could produce an execution of infinite length by always scheduling a process whose next step produces a multivalent configuration. No process would ever output a decision in this execution, since any configuration in which some process has produced an output is univalent. This is impossible in a wait-free protocol.

A bivalency argument (see [10]) may be used to show that the next operation performed by any process when the system is in the configuration S_0 must be an operation on the same object, say X, and that X cannot be a register. It follows from Lemma 2 that T is n-discerning. □

The following lemma and Propositions 5 and 6 will be used first in the proof of Theorem 9, which provides a converse to Theorem 3, and again in section 4.

A set $\mathcal{O}$ of objects is said to be capable of solving *team-restricted n-consensus* if there is a partition of the n processes $P_1, \ldots, P_n$ into two nonempty teams A and B such that the consensus problem can be solved using only the objects in the set $\mathcal{O}$, provided all processes on the same team have the same input value.

LEMMA 4. *If a set $\mathcal{O}$ of objects can be used to solve team-restricted $(n+1)$-consensus, then $\mathcal{O}$ can be used to solve team-restricted n-consensus.*

Proof. Any consensus protocol for team-restricted $(n+1)$-consensus may be viewed as a protocol for team-restricted n-consensus by thinking of one of the processes on a team with at least two processes as having failed before executing any of its steps. □

PROPOSITION 5. *Let $n \geq 2$. Suppose O_i is a shared object that can be used along with k registers to solve team-restricted i-consensus for all i, $2 \leq i \leq n$. Then n-consensus can be solved using objects $O_2, \ldots, O_n$ and $k(n-1)$ registers.*

Proof. The proposition will be proved by induction on n. For $n = 2$, the team-restricted form of the consensus problem is identical to the general problem of consensus for two processes.

Let $n > 2$. Suppose the claim holds when the number of processes is less than n. This means that, for all $m < n$, $O_2, \ldots, O_m$ can be used, along with $k(m-1)$ registers, to solve m-consensus. Divide the n processes into two nonempty teams A and B as described in the definition of team-restricted n-consensus. The processes of team A first execute a consensus protocol to agree on one of their input values. If $|A| = 1$, no shared objects are used. Otherwise, team A can agree on an input value using $k(|A|-1)$ registers and the shared objects $O_2, \ldots, O_{|A|}$, by the induction hypothesis. Similarly, the processes of team B agree on one of their input values. If $|B| = 1$, no

shared objects are used. Otherwise, it can be done using $k(|B|-1)$ registers and the shared objects $O_{|A|+1}, \ldots, O_{n-1}$. This is possible, since $O_{i+|A|-1}$ can be used with k registers to solve team-restricted i-consensus for $2 \leq i \leq |B|$, by Lemma 4.

Next, the processes agree on which team's value should be used as the common decision value. The processes execute the team-restricted n-consensus protocol, with each process using the decision value of its team as its input. This can be done using O_n and an additional k registers for a total of $k(n-1)$ registers.

By the inductive hypothesis, the agreement within each team is wait-free. The team-restricted n-consensus protocol used to decide between the two teams' values is also wait-free. So, the entire consensus protocol is wait-free. The protocol satisfies the validity condition, since the value agreed upon by the winning team must be one of the input values of a process on that team, by the inductive hypothesis. The protocol satisfies the consistency condition, since the team-restricted n-consensus protocol must satisfy the consistency condition. □

PROPOSITION 6. *Suppose that one object of type T and k registers can be used to solve team-restricted n-consensus. Then, n-consensus can be solved using $n-1$ objects of type T and $k(n-1)$ registers.*

Proof. This follows immediately from Lemma 4 and Proposition 5. □

LEMMA 7. *An RMW object of type T and two registers can be used to solve team-restricted n-consensus if T is n-discerning.*

Proof. Divide the processes into two nonempty teams A and B, assign a function f_i to each process P_i, and choose v_0 to satisfy the conditions of Definition 1. An algorithm will be constructed for the team-restricted n-consensus problem. The algorithm uses an object X of type T that initially has state v_0 and two registers called R_A and R_B.

Each process P_j first writes its team's common input value into the register R_A, if it belongs to team A, or into the register R_B, if it belongs to team B. The process P_j then performs its assigned operation RMW_{f_j} on X and uses the result of this operation to determine whether a process from team A or from team B was the first to access X.

Without loss of generality, suppose that process P_j belongs to team A. If P_j's *RMW* operation returns the value v_0, then a process from team A was the first to access X. To see why this is true, suppose some processes, starting with a process from team B, did access X before P_j. Let $i_1, \ldots, i_\alpha$ be the indices of these processes. Then $(f_\alpha \circ \cdots \circ f_1)(v_0) = v_0$ and $P_{i_1} \in B$, violating condition III. If P_j's *RMW* operation returns a value different from v_0, the process must be able to deduce which team accessed X first by checking whether the value belongs to $V_{A,j}$ or $V_{B,j}$. These two finite sets are disjoint (by condition I), and contain all possible values of the object X that can be observed by process P_j in this protocol. Once P_j decides whether team A or team B accessed X first, it returns the value in R_A or R_B, respectively.

Each process performs only $O(1)$ steps, so wait-free termination is guaranteed. The protocol satisfies the validity condition, since the winning team's value is written into the team's register before any process from that team can access the object X. The protocol also satisfies the consistency condition: all processes agree on the winning team and return the value of that team's register (which never changes after it is first written, since all processes on the same team have the same input value). □

The following theorems follow immediately from Lemma 7 and Propositions 5 and 6.

THEOREM 8. *The RMW objects $X_2, \ldots, X_n$ can be used, with $2(n-1)$ registers,*

to solve n-consensus if the type of X_i is i-discerning for each i.

THEOREM 9. *If T is an n-discerning RMW type, then there is a protocol for n-consensus that uses $n-1$ objects of type T and $2(n-1)$ registers.*

Theorems 3 and 9 show that an RMW type can be used with registers to solve n-consensus if and only if it is n-discerning. The remainder of this section discusses some consequences of this characterization.

The constructive proof of Theorem 9 provides upper bounds on the complexity of solving the consensus problem using RMW objects and registers. If objects of an RMW type T and registers can be used to solve n-consensus at all, then T is n-discerning by Theorem 3, and the tournament-style algorithm in the proof of Proposition 5 can solve n-process consensus in $O(n)$ steps per process, using $n-1$ objects of type T and $2(n-1)$ registers.

COROLLARY 10. *RMW objects of type T can be used, with registers, to implement any type of object in a system of n processes if and only if T is n-discerning.*

Proof. This follows from Theorems 3 and 9 and the fact that n-consensus objects can be used to obtain an implementation of any shared object in a system of n processes [10]. □

COROLLARY 11. *For RMW types with finite state sets, the following question is decidable: "Given an integer n and an RMW type T, can n-consensus be solved using registers and RMW objects of type T?"*

Proof. The conditions of Definition 1 can be checked for each of the finite number of possible choices of $v_0, A, B, f_1, \ldots, f_n$ in a finite amount of time. □

For every value of n, there is an RMW object type with consensus number exactly n. This can be shown by considering an RMW object that behaves like a sticky bit [20] that gets reset after n accesses.

PROPOSITION 12. *Let $n \geq 2$. Let $V = \{\perp\} \cup (\{A, B\} \times \{1, \ldots, n-1\})$. Let T be the RMW type with state set V whose operations can apply the functions f_A and f_B, where*

$$f_{team}(x) = \begin{cases} (team, 1) & \text{if } x = \perp, \\ \perp & \text{if } x = (team', n-1), \\ (team', z+1) & \text{if } x = (team', z) \text{ and } z < n-1. \end{cases}$$

Then T is n-discerning but not $(n+1)$-discerning.

Proof. First, it is shown that T is n-discerning. Let $v_0 = \perp$, $A = \{P_1\}$, $B = \{P_2, \ldots, P_n\}$, $f_1 = f_A$, and $f_2 = \cdots = f_n = f_B$. Then, each value in $V_{\text{team},j}$ is an ordered pair whose first component is team. The conditions of Definition 1 are therefore clearly satisfied.

Next, it is shown that T is not $(n+1)$-discerning. Consider any choice of $v_0, A, B, f_1, \ldots, f_{n+1}$.

If v_0 is an ordered pair, let z be the second component of v_0. Let P_j be any process. Both $V_{A,j}$ and $V_{B,j}$ contain the element $\perp$, since any sequence of $n-z$ functions applied to v_0 will result in the value $\perp$. This violates condition I in the definition of $(n+1)$-discerning.

If $v_0 = \perp$, let P_j be a process in team B. The set $V_{A,j}$ contains v_0, since any sequence of n functions when applied to $\perp$ results in the value $\perp$. This violates condition II of the definition of $(n+1)$-discerning. □

4. Solving consensus with readable objects. Let T be a readable object type with state set $Q = \bigtimes_{k \in \Gamma} Q_k$. Consider a distributed system with $n \geq 2$ processes, called $P_1, \ldots, P_n$, which communicate via shared objects of type T and

registers. The ability of such a system to solve the consensus problem will be studied in this section.

A readable type T is defined to be *n-discerning* if the set $\{P_1, \ldots, P_n\}$ can be partitioned into two nonempty teams and a single operation can be assigned to each process so that if processes from some subset of $\{P_1, \ldots, P_n\}$ each perform their own operation on an appropriately initialized object X of type T, then each one could determine which team accessed X first, provided that it could see the final state of X. The operation assigned to each process cannot be a *read* operation: it must be possible for the operation to update the state of the readable object. This is formalized as follows.

DEFINITION 13. *The readable type T is called n-discerning if there exist*

- *a state $q_0 \in Q$,*
- *a partition of the set of processes $\{P_1, \ldots, P_n\}$ into two nonempty teams A and B, and*
- *an update operation op_i for $1 \leq i \leq n$*

such that $R_{A,j} \cap R_{B,j} = \emptyset$, for all $j \in \{1, \ldots, n\}$, where $R_{A,j}$ is the set of pairs (r, q) for which there exist distinct process indices $i_1, \ldots, i_\alpha$ including j with $P_{i_1} \in A$ such that if $P_{i_1}, \ldots, P_{i_\alpha}$ each perform their operations (in that order) on an object of type T that is initially in state q_0, P_j gets the result r, and the object ends in state q. The set $R_{B,j}$ is defined similarly as the set of pairs (r, q) for which there exist distinct process indices $i_1, \ldots, i_\alpha$ including j with $P_{i_1} \in B$ such that if $P_{i_1}, \ldots, P_{i_\alpha}$ each perform their operations (in that order) on an object of type T that is initially in state q_0, P_j gets the result r, and the object ends in state q.

It will be shown in Theorems 15 and 18 that readable objects of type T can be used, along with registers, to solve the consensus problem for n processes if and only if T is n-discerning. First, the conditions are shown to be necessary.

LEMMA 14. *If S_0 is a critical configuration of an n-process consensus protocol and the next step by every process is an operation on a readable object X of type T, then T is n-discerning.*

Proof. This proof is similar to the proof of Lemma 2. Let q_0 be the state of X in S_0, and let op_i be the operation performed by P_i in its first step after S_0. Partition the processes into two teams A and B according to their critical values.

To derive a contradiction, suppose these choices for $q_0, A, B, op_1, \ldots, op_n$ do not satisfy Definition 13. Then, there is a pair $(r, q) \in R_{A,j} \cap R_{B,j}$ for some j. There is some sequence $i_1, \ldots, i_\alpha$ of distinct process indices, including j, such that $P_{i_1} \in A$ and if processes $P_{i_1}, \ldots, P_{i_\alpha}$ each perform their next operation, in that order, starting from S_0, process P_j will receive the result r, and the system will end in a configuration S_A where X is in state q. There is some other sequence $k_1, \ldots, k_\beta$ of distinct process indices, including j, such that $P_{k_1} \in B$ and if processes $P_{k_1}, \ldots, P_{k_\beta}$ each perform their next operation, in that order, starting from S_0, process P_j will again receive the result r, and the system will end in a configuration S_B where X is in state q. The configurations S_A and S_B are indistinguishable to P_j, so a solo execution by P_j from either of these two configurations would lead to the same decision value. This contradicts the fact that S_A and S_B are univalent configurations that lead to different decision values.

The operation performed by each process must be an update operation; otherwise the configuration obtained from S_0 by allowing the process to perform its operation could not be distinguished from S_0 by any process on the opposite team. □

Combining this lemma with a bivalency argument yields the following theorem.

THEOREM 15. *If n-process consensus can be solved using registers and objects of a readable type T, then T is n-discerning.*

Proof. A bivalency argument, as in the proof of Theorem 3, shows that any n-consensus protocol that uses registers and objects of type T must have a critical configuration S_0, and that the next operation by each process will be applied to the same object of type T. The theorem then follows from Lemma 14. □

Theorem 15 may be used to establish an upper bound on the consensus number of any (deterministic) type T, whether it is readable or not. If, for some n, the update operations that are permitted for type T do not satisfy Definition 13, then type T cannot be used with registers to solve n-consensus. This is because the addition of a *read* operation to the specification of type T would create a readable type T' that is at least as powerful as type T but has consensus number less than n, by Theorem 15.

The team-restricted n-consensus problem will now be used to provide a converse to Theorem 13.

LEMMA 16. *Let T be an n-discerning readable object type. An object of type T and two registers can be used to solve team-restricted n-consensus.*

Proof. Choose $q_0, A, B, op_1, \ldots, op_n$ to satisfy Definition 13. A protocol will be developed for team-restricted n-consensus that uses one register for each team and one shared object X of type T, initialized to the state q_0. Each process P_j writes its team's common input value into its team's register. It then applies the operation op_j to X and attempts to read the state of X to determine which team accessed X first.

The state set of T has the form $Q = \bigtimes_{k\in\Gamma} Q_k$. Since Γ may be an infinite set, it will first be shown that process P_j can determine the winning team from the values of a finite number of the components. Let $R_{A,j}$ and $R_{B,j}$ be the disjoint sets defined in Definition 13. These sets are finite, since the number of ways to choose $\alpha, i_1, \ldots, i_\alpha$ in the definitions of $R_{A,j}$ and $R_{B,j}$ is bounded by $n \cdot n!$. For $(r, q) \in R_{A,j}$ and $(r', q') \in R_{B,j}$, let $k(q, q')$ be an element of Γ that indexes some state component where q and q' differ, if such a component exists. Let Δ_j be the set of such indices $k(q, q')$ for all possible choices of (r, q) and (r', q'). The number of such choices is finite, so Δ_j is a finite set. Let π_j be the projection function from Q to the set $\bigtimes_{k\in\Delta_j} Q_k$. This projection function discards all components of the state, except for the finite number of components indexed by the elements of Δ_j.

Suppose the sets $\{(r, \pi_j(q)) : (r, q) \in R_{A,j}\}$ and $\{(r', \pi_j(q')) : (r', q') \in R_{B,j}\}$ have an element in common. Then there are two distinct pairs $(r, q) \in R_{A,j}$ and $(r, q') \in R_{B,j}$ such that $q \neq q'$ and $\pi_j(q) = \pi_j(q')$. This is impossible, since $k(q, q') \in \Delta_j$. Thus, process P_j can discern executions in which team A performed the first update from executions in which team B performed the first update, using only the response to its own update operation and the projection $\pi_j(q)$ of the state q of X at any time after P_j's update has been performed.

After performing its update operation, the process P_j reads, one by one, the components of the state that are indexed by Δ_j. The state of X may be updated by other processes while process P_j is performing this scan of the components. Each scan produces a view of the image of the state of X under the projection π_j. Such a view is called *accurate* if it correctly reflects the state of X at some moment during the execution of the scan. If another process performs an update operation during a scan, the resulting view may not be accurate, but any scan that is not interrupted by an update will produce an accurate view.

To ensure that P_j can correctly determine which team accessed X first, the scan of the components of X is repeated $2n - 1$ times. An update of X can occur during

at most $n-1$ of these scans, so at least n of the scans will return an accurate view of the state of X. By Definition 13, P_j can correctly determine which team accessed X first using the information from any accurate scan and the result of its operation op_j. Since a majority of the scans are accurate, P_j can correctly determine which team accessed X first. Process P_j then decides on the value stored in the register belonging to the team that accessed X first.

The validity condition for the consensus problem is satisfied, since every process must agree on the team that accessed X first. The consistency condition is also satisfied, since a process from the winning team must have written its value to its team's register before accessing X. The protocol is wait-free, since each of the $2n-1$ scans reads only a finite number of components of X. □

The following theorems follow immediately from Lemma 16 and Propositions 5 and 6.

THEOREM 17. *Let T_i be an i-discerning readable object type for $2 \leq i \leq n$. Then the n-consensus problem can be solved using one object O_i of each type T_i, together with $2(n-1)$ registers.*

THEOREM 18. *If T is an n-discerning readable object type, then there is a protocol for n-consensus that uses $n-1$ objects of type T and $2(n-1)$ registers.*

This completes the proof that a readable type T can be used with registers to solve n-consensus if and only if T is n-discerning. This characterization has the following consequences.

COROLLARY 19. *Readable objects of type T can be used, along with registers, to implement every other type of object in a system with n processes if and only if T is n-discerning.*

Proof. This follows from Theorems 15 and 18 and the fact that n-consensus objects can be used to obtain an implementation of any shared object in a system of n processes [10]. □

COROLLARY 20. *If the state set of object type T and the set of possible operations on object type T are both finite, then the following question is decidable: "Given a positive integer n and a readable type T, can n-consensus be solved using only objects of type T and registers?"*

Proof. The conditions of Definition 13 can be checked for each of the finite number of choices of $q_0, A, B, op_1, \ldots, op_n$ in a finite amount of time. □

It will now be shown that the addition of a *scan* operation, which reads the entire state atomically, to any readable type T does not increase its power to solve consensus.

COROLLARY 21. *Let T be a readable type. Let T' be a type that is the same as T, except that it allows an additional scan operation that reads the entire state of T. Then T and T' have the same consensus number.*

Proof. Let n be the consensus number of T'. Clearly, the consensus number of T is at most n. By Theorem 15, T' is an n-discerning readable type. Let $q_0, op_1, \ldots, op_n, A$ and B be chosen to satisfy Definition 13 for T'. None of the operations can be a *scan*, since *scan* operations never update the state of an object. Therefore, the choice of $q_0, op_1, \ldots, op_n, A$ and B will also satisfy Definition 13 for type T. By Theorem 18, it is possible to solve n-consensus using objects of type T and registers. □

It can be shown that, for each $n > 1$, there is a readable object type, analogous to the RMW object defined in Proposition 12, that has consensus number n. (See [23] for a detailed description of this object.)

5. Robustness for RMW and readable objects. Jayanti [14] formalized Herlihy's notion of a hierarchy [10] of shared object types and defined a number

of desirable properties for hierarchies, including robustness. A *wait-free hierarchy* classifies object types according to their power to implement one another. Formally, it is a mapping h of object types to the set of levels $\mathbb{N} \cup \{\infty\}$, where a type T is in level n only if objects of type T, together with registers, can be used to implement any other type of object in a system of n processes. If $h(T) = \infty$, then objects of type T and registers can be used to implement any other type of object in a system of n processes for all n. A wait-free hierarchy is *tight* if every object type is mapped to the highest level possible. Thus, if type T is mapped to level n of a tight wait-free hierarchy, there is some type that cannot be implemented using objects of type T and registers in a system of $n + 1$ processes. A wait-free hierarchy is *robust* if no object in any level of the hierarchy can be implemented using a finite number of types of objects from lower levels. In the consensus hierarchy, h^r_m, the level of a type T is the consensus number of T. Jayanti showed that h^r_m is the (unique) tight wait-free hierarchy [14].

Chandra et al. [6] showed that the consensus hierarchy is not robust, if nondeterministic, nonoblivious objects are allowed. Schenk [24] proved that the consensus hierarchy is not robust, even for oblivious objects, if objects with unbounded nondeterminism are allowed. Lo and Hadzilacos [17] improved this, showing that the hierarchy h^r_m restricted to oblivious objects is not robust even when nondeterminism is bounded. Moran and Rappoport [19] showed that the consensus hierarchy is not robust for deterministic nonoblivious objects using the restricted hard-wired binding model. (See Jayanti's survey [13] for a description of binding models, which restrict the ways that processes can access nonoblivious objects.)

Borowsky, Gafni, and Afek [5] claimed that the consensus hierarchy is robust for deterministic objects using a less restrictive binding model. Their paper is quite complex. Here, the characterizations of RMW and readable objects that can solve n-process consensus will be used to provide a concise proof of the robustness of the hierarchy when restricted to deterministic RMW and readable objects.

THEOREM 22. *Let T be a readable or RMW object type. Let $\mathcal{S}$ be a finite set of readable and RMW object types such that $h^r_m(T') < h^r_m(T)$ for each $T' \in \mathcal{S}$. Then an object of type T cannot be implemented using objects whose types are from the set $\mathcal{S}$.*

Proof. Let $n = \max\{h^r_m(T') \mid T' \in \mathcal{S}\} + 1$. This quantity is finite, since $h^r_m(T')$ is less than $h^r_m(T)$ and therefore finite for each $T' \in \mathcal{S}$, and $\mathcal{S}$ is a finite set.

To derive a contradiction, suppose the claim is false. Then, since $h^r_m(T) \geq n$, there is a protocol using objects whose types are from the set $\mathcal{S}$ that solves consensus among n processes. A bivalency argument [10] shows that this protocol has a critical configuration, S_0, and that the next operation taken by any process when the system is in this configuration must be an operation on the same object, X. Let T_X be the type of object X.

First, suppose that T_X is an RMW type. Then T_X is n-discerning, by Lemma 2. It follows from Theorem 9 that $h^r_m(T_X) \geq n$, contradicting the definition of n.

Now suppose that T_X is a readable type. The type T_X is n-discerning, by Lemma 14. By Theorem 18, $h^r_m(T_X) \geq n$, which again contradicts the definition of n. □

This theorem allows the decidability results of Corollaries 11 and 20 to be extended to finite sets of object types. If $\mathcal{S}$ is any finite set of finitely-specified RMW and readable object types, then one can decide whether objects whose types are in $\mathcal{S} \cup \{\text{register}\}$ can be used to solve n-process consensus, by checking whether any of the types in $\mathcal{S}$ are n-discerning.

Acknowledgments. This research forms part of my Ph.D. thesis [23] which was done at the University of Toronto. I am grateful for the guidance of my adviser, Faith Fich. I thank the anonymous PODC referees for their suggestions, and Vassos Hadzilacos, Maurice Herlihy, Wai-Kau Lo, Michael Merritt, and Eric Schenk for helpful discussions.

REFERENCES

[1] Y. Afek, H. Attiya, D. Dolev, E. Gafni, M. Merritt, and N. Shavit, *Atomic snapshots of shared memory*, J. ACM, 40 (1993), pp. 873–890.
[2] J. H. Anderson, *Composite registers*, Distributed Computing, 6 (1993), pp. 141–154.
[3] J. Aspnes and M. Herlihy, *Wait-free data structures in the asynchronous PRAM model*, in Proceedings of the 2nd ACM Symposium on Parallel Algorithms and Architectures, Crete, Greece, 1990, pp. 340–349.
[4] O. Biran, S. Moran, and S. Zaks, *A combinatorial characterization of the distributed* 1*-solvable tasks*, J. Algorithms, 11 (1990), pp. 420–440.
[5] E. Borowsky, E. Gafni, and Y. Afek, *Consensus power makes (some) sense!*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 363–372.
[6] T. Chandra, V. Hadzilacos, P. Jayanti, and S. Toueg, *Wait-freedom vs. t-resiliency and the robustness of wait-free hierarchies*, in Proceedings of the 13th ACM Symposium on Principles of Distributed Computing, Los Angeles, CA, 1994, pp. 334–343.
[7] B. Chor and L. Moscovici, *Solvability in asynchronous environments*, in Proceedings of the 30th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 422–433.
[8] M. J. Fischer, N. A. Lynch, and M. S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. ACM, 32 (1985), pp. 374–382.
[9] E. Gafni and E. Koutsoupias, *Three-processor tasks are undecidable*, SIAM J. Comput., 28 (1999), pp. 970–983.
[10] M. Herlihy, *Wait-free synchronization*, ACM Trans. Prog. Lang. Syst., 11 (1991), pp. 124–149.
[11] M. Herlihy and S. Rajsbaum, *The decidability of distributed decision tasks*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 589–598.
[12] M. P. Herlihy and J. M. Wing, *Linearizability: A correctness condition for concurrent objects*, ACM Trans. Prog. Lang. Syst., 12 (1990), pp. 463–492.
[13] P. Jayanti, *Wait-free computing*, in Distributed Algorithms, Lecture Notes in Comput. Sci. 972, Springer-Verlag, Berlin 1995, pp. 19–50.
[14] P. Jayanti, *Robust wait-free hierarchies*, J. ACM, 44 (1997), pp. 592–614.
[15] P. Jayanti and S. Toueg, *Some results on the impossibility, universality and decidability of consensus*, in Distributed Algorithms, Lecture Notes in Comput. Sci. 647, Springer-Verlag, Berlin, 1992, pp. 69–84.
[16] C. P. Kruskal, L. Rudolph, and M. Snir, *Efficient synchronization on multiprocessors with shared memory*, ACM Trans. Prog. Lang. Syst., 10 (1988), pp. 579–601.
[17] W.-K. Lo and V. Hadzilacos, *All of us are smarter than any of us: Wait-free hierarchies are not robust*, in Proceedings of the 29th ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 579–588.
[18] N. A. Lynch, *Distributed Algorithms*, Morgan Kaufmann, San Francisco, CA, 1996, chapter 8.
[19] S. Moran and L. Rappoport, *On the robustness of* h_m^r, in Distributed Algorithms, Lecture Notes in Comput. Sci. 1151, Springer-Verlag, Berlin, 1996, pp. 344–361.
[20] S. Plotkin, *Sticky bits and universality of consensus*, in Proceedings of the 8th ACM Symposium on Principles of Distributed Computing, Edmonton, AB, Canada, 1989, pp. 159–175.
[21] E. Ruppert, *Consensus numbers of multi-objects*, in Proceedings of the 17th ACM Symposium on Principles of Distributed Computing, Puerto Vallarta, Mexico, 1998, pp. 211–217.
[22] E. Ruppert, *Consensus numbers of transactional objects*, in Distributed Computing, Lecture Notes in Comput. Sci. 1693, Springer-Verlag, Berlin, 1999, pp. 312–326.
[23] E. Ruppert, *The Consensus Power of Shared-Memory Distributed Systems*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 2000. Available from www.cs.yorku.ca/∼ruppert.
[24] E. Schenk, *The consensus hierarchy is not robust*, in Proceedings of the 16th ACM Symposium on Principles of Distributed Computing, Santa Barbara, CA, 1997, p. 279.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1169–1196

EARLY DETECTION OF MESSAGE FORWARDING FAULTS*

AMIR HERZBERG† AND SHAY KUTTEN‡

Abstract. In most communication networks, pairs of processors communicate by sending messages over a path connecting them. We present communication-efficient protocols that quickly detect and locate any failure along the path. Whenever there is excessive delay in forwarding messages along the path, the protocols detect a failure (even when the delay is caused by maliciously programmed processors). The protocols ensure optimal time for either message delivery or failure detection.

We observe that the actual delivery time δ of a message over a link is usually much smaller than the a priori known upper bound D on that delivery time. The main contribution of this paper is the way to model and take advantage of this observation. We introduce the notion of asynchronously early-terminating protocols, as well as protocols that are asynchronously early-terminating, i.e., time optimal in both worst case and typical cases. More precisely, we present a time complexity measure according to which one evaluates protocols both in terms of D and δ. We observe that asynchronously early termination is a form of competitiveness.

The protocols presented here are asynchronously early terminating since they are time optimal both in terms of D and of δ. Previous communication-efficient solutions were slow in the case where $\delta \ll D$. We observe that this is the most typical case.

It is suggested that the time complexity measure introduced, as well as the notion of asynchronously early-terminating, can be useful when evaluating protocols for other tasks in communication networks. The model introduced can be a useful step towards a formal analysis of real-time systems.

Our protocols have $O(n \log n)$ worst-case communication complexity. We show that this is the best possible for protocols that send immediately any acknowledgment they ever send. Then we show an early-terminating protocol which uses timing and delay to reduce the communication complexity in the typical executions where the number of failures is small and $\delta \ll D$. In such executions, its message complexity is linear, as is the complexity of nonfault tolerant protocols.

Key words. real time, distributed algorithms, fault tolerance, competitive algorithms, network protocols, time adaptive

AMS subject classifications. 68W15, 68W10, 68W20, 94C15, 68R25

PII. S0097539796312745

1. Introduction.

In this paper, we introduce a complexity measure of time complexity for asynchronous networks for which there exists an upper bound on the delivery time of a message over a link. It is suggested that this can be a useful step toward improving the analysis of actual communication networks, as well as a step toward the formal analysis of real-time systems. Using this complexity measure, we develop optimal protocols for dealing with the task of managing end-to-end communication sessions.

The end-to-end delivery of information from *source to destination* is a basic communication task. The most communication-complexity-efficient method for delivering

*Received by the editors December 2, 1996; accepted for publication (in revised form) October 13, 1999; published electronically September 20, 2000. Preliminary reports of parts of this work appeared in *Proceedings of the Ninth International Conference on Computers and Communication*, Tel Aviv, Israel, 1988 and *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, Edmonton, Canada, 1989, pp. 339–353.

http://www.siam.org/journals/sicomp/30-4/31274.html

†IBM Haifa Reserach Lab, IBM, 5th floor, 2 Weizmann Street, Tel Aviv, Israel, 52960 (amir@il.ibm.com).The research of this author was partially supported by grant 86-00301 from the United States–Israel Binational Science Foundation (BSF). Part of the research was done while this author was with the Technion–Israel Institute of Technology.

‡Faculty of Industrial Engineering and Management, Technion, Haifa 32000, Israel (kutten@ie.technion.ac.il).

information is to send it along a fixed (short) path between two processors. There are also some other reasons that make this method the most common (e.g., as used in [Tan81, BGJ+85, MRR80]). For example, "first-in first-out" (FIFO) service for messages is then guaranteed, without the need for expensive hardware or for software intervention to restore the order of the messages.

Of course, this requires that all the links and the processors along the path are operational. When a link or a processor fails (which does not happen very often, compared to the rate of message traffic), a different fixed path is established instead of the disconnected one. For that, additional mechanisms are used to detect and locate failures. These mechanisms are based on acknowledgments and use a time-out value D, which is a bound on the transmission delay over one link. Often fail-stop failures are detected by *hop-by-hop acknowledgments*, which are control messages sent by a processor to its neighbor upon receiving a data message from that neighbor. If processor u does not receive an acknowledgment from neighbor v within D since u sent a data message to v, then either v or link (u, v) failed.

There are some failures which are not detected by the hop-by-hop acknowledgments mechanism. A simple example is a malicious failure, where v "intentionally" sends an acknowledgment without forwarding the data message toward the destination. A similar outcome may result without malicious intent, i.e., when a processor breaks down after sending the acknowledgment but before succeeding in forwarding the data message. (A more likely case is that a processor did forward the message before it failed, but the message got lost over the link; the failed processor cannot now retransmit the message.) This kind of failure is usually dealt with by an *end-to-end acknowledgment* mechanism. This is an acknowledgment message which is sent from the destination to the source when the destination receives a data message. If the source does not receive an acknowledgment within $2(n-1)D$ since it sent the data message, where n is the number of processors along the path, then a failure has occurred.

For a given execution of the protocol, and for any f, let δ_f denote the maximal transmission delay over a link in this execution, when not counting the worst f links. Moreover, $\delta_f \leq D$ is not any bound (even unknown) but rather the actual maximal delay as could be observed had there been some outside observer. Intuitively, when f is the number of faults in an execution, δ_f is the maximum delay over nonfaulty links (and between two nonfaulty neighboring processors). However, this definition makes sense also when there are no "real faults" (or at least no faults can be detected, since the delay on all links is still smaller than D); some links just happened to be slower than others. For simplicity, we prefer to use δ_f rather than the actual delay (in the execution) over each link separately. In addition, since our protocols can deal even with malicious faults of processors, most of the paper uses this meaning of f. Thus, f is clear from the context, and we shall use the notation δ instead of δ_f. In section 7, we analyze briefly the meaning of the results in the case where no faults occur.

Network designers usually choose a bound D which is much larger than the typical transmission delay. This is due to the unpredictability of the actual delay, and to the huge overhead of disconnecting a link (see, e.g., [GSK87]). Therefore, in typical executions, $\delta \ll D$ holds. This motivates an analysis of the time complexity which considers both D and δ; intuitively, one would like to derive a bound on the time complexity that will depend, as much as possible, on the (usually small) value of δ, rather than on the value of D.

We call a protocol *asynchronously early-terminating* if its time complexity is op-

TABLE 1
Parameters.

Parameter	Meaning	Typical value	Initially known?
n	Number of processors	< 20	Yes
f	Number of faulty processors and links	0 or 1	No
D	Bound on the delay on a nonfaulty link	1 sec.	Yes
δ	Actual delay on a nonfaulty link	10 msec.	No

timal for any selection of the number of faults f, of D, and, of δ. (We shorten it to *early-terminating*, when there is no ambiguity with the "early-stopping" synchronous Byzantine agreement protocols discussed in many papers [DRS86].) We present early-terminating detection protocols, with time complexity $O(fD + n\delta)$, where f is the number of faults. This improves substantially over the common end-to-end mechanism, which requires $2(n-1)D$, for typical executions where $f \ll n$ and $\delta \ll D$. Typical values of the parameters n, f, D, and δ are shown in Table 1.

Another way to see the improvement is when considering the competitiveness of the protocols. Consider the time complexity $T_C(P)$ of a protocol P for a given configuration C, that is, given the set F of faulty processors (where $|F| = f$) and the values of D and δ. Define the competitive ratio of a protocol P_1 with respect to protocol P_2 and the given configuration as $\frac{T_C(P_1)}{T_C(P_2)}$. The competitive ratio of P_1 with respect to P_2 is $\max_C \frac{T_C(P_1)}{T_C(P_2)}$. Clearly, the competitive ratio of the common end-to-end mechanism with respect to our protocols is $\Omega(n)$, where the maximum ratio is achieved when δ approaches zero.

To compute the distributed competitiveness of a protocol P_1, one needs to compare it to the "best" distributed algorithm, rather than to any other. To rephrase the recent definition of [AADW94], for a given configuration C, one divides $T_C(P_1)$ by the complexity (for configuration C) of a distributed protocol OPT_C that achieves the best time complexity for configuration C. It is required that OPT_C will perform correctly in any case. The distributed competitiveness of P_1 is the maximum (over all the configurations) of such ratio. It can be shown that early-terminating protocols for our problem are distributively competitive (i.e., have constant competitive ratios). Thus, our protocols are also distributively competitive.

The protocols presented in this paper allow early-terminating and communication-efficient detection of arbitrary faults, while forwarding information from the source to the destination. Our protocols also provide fault *location*, i.e., when a failure occurs, the source learns of a specific link such that either the link or one of its endpoints is faulty. This is useful for most recovery actions. Both detection and location are done in optimal time for any value of D and δ. (Recall that we analyze the time complexity as a function of D and δ.)

Table 2 summarizes our results and previous results. Note that all of our protocols provide fault location, which was achieved previously only with $O(n^2)$ communication complexity.

Our main contribution is the measure of time complexity as depending on D and δ, with the concept that one should strive to obtain time complexity bounds that depend as much as possible on δ rather than on D. Other contributions are communication-efficient early-terminating protocols. The *immediate-Acks protocol*,

TABLE 2
Protocols.

Protocol	Time	Communication	Remarks
ISO 8072/3, CCITT x.224	$O(nD)$	$O(n)$	Not locating
[Per88, section 3.5]	$O(nD)$	$O(n^2)$	Locating
End-to-end (section 4.1)	$O(nD)$	$O(n)$	Locating
Hop-by-hop (section 4.2)	$O(fD + n\delta)$	$O(n^2)$	Locating
Immediate-Acks (section 6.1)	$O(fD + n\delta)$	$O(n \log(n))$	Locating
Adaptive (section 6)	$O(fD + n\delta)$	$O(n \log(f + \frac{n\delta}{D}))$	Locating

presented in section 6.1, has $O(n \log n)$ communication complexity. We present in sections 6.3 and 6.4 an early terminating protocol which is adaptive, in the sense that the number of acknowledgment messages sent depends on the delays and the behavior of the adversary in the particular execution. Its communication complexity is $O(n \log(2 + f + \frac{n\delta}{D}))$. We argue that in typical applications of this type of protocol, the number of failures f is zero or very small; otherwise the network designer will avoid transmission of messages over a fixed path. The term $\frac{n\delta}{D}$ is also usually small. Hence, in practice, this communication complexity is close to the optimal communication complexity ($O(n)$).

We show that for other kinds of protocols (i.e., those that are message-driven, except for the decision to disconnect a link that may be time-driven), a lower bound of $\Omega(n \log n)$ exists. The proof of the lower bound shows collections of paths of total length $\Omega(n \log n)$ that any protocol must use for sending message (acknowledgments). Our adaptive protocol mentioned economizes on message-sending by avoiding using some of these paths when the delays and number of faults are small. However, there are some executions in which this protocol must utilize all the paths in its collections. It is an open problem whether there exists an optimal time, early terminating protocol whose worst-case message complexity is better (in order of magnitude). An interesting corollary from the proof of the lower bound is that such a protocol (if it exists) will not have an execution that sends messages over every path that is used in some execution.

Related works. The asynchronous model with bounded delay was previously studied in [AE86, CCGZ88, DHSS84] without considering early termination.

Communication via a fixed path was studied in [SJ86]. Detection of failures was addressed in accepted and in proposed standards [Sta87]. Recovery from (detected) errors during such communication was studied in [Gro82]. Detection, location, and recovery from arbitrary transmission failures were studied by Perlman [Per88], who introduced the notions of communication failure detection in the environment of malicious processors, and that of failure location in this context.

Many works presented end-to-end communication protocols which do not depend on a single path [CR87, Fin79, Per88, AG88, AAGMRS97, AGH90]. The goal of these works is to increase reliability. In the extreme, these works achieve communication even if there is no moment when there is a nonfaulty path from the source to the destination [AE86, AG88, AAGMRS97]. These methods are useful for source-to-destination communication only in applications where the increased reliability compensates for the much higher communication complexity, storage requirements, and local processing.

We permitted the processors to fail in an arbitrary manner. However, we assumed that the links are "well-behaved"; namely, the links either work correctly or their failure is detected. The justification for this assumption is the known link protocols [BS88, GHM89, Zim80].

Our adaptive protocol (section 6) is based on the idea that a processor i can "piggyback" its acknowledgment on an acknowledgment of another processor j. This raises the question: How long should processor i wait for the acknowledgment of processor j before it gives up the idea of piggybacking (and sends its acknowledgment)? A similar problem was studied in [AAPS87, AS87, BYKWZ87, K88].

Our protocols can be viewed as being competitive (especially in the time complexity measure). Competitive distributed algorithms were studied further in later papers, e.g., [AKP92, AADW94].

Finally, this work should be viewed in the context of the important work published later dealing with formal approach and modeling of distributed real-time systems. A paper with a strong impact is [ADLS91], which suggests a more detailed model, and gives an algorithm for agreement whose time depends mostly on δ, and only minimally on D. This work was extended by [Pon91] to handle omission and Byzantine faults. In [AL89], formal analysis of timing uncertainties and time bounds is done with respect to another task.

Organization. In the next two sections, we define the communication model and the problem. In section 4, we present two natural, simple protocols that solve the problem. The two protocols are presented mainly in order to demonstrate the problem and the model. One (which is similar to the x.244 protocol [Sta87]) is communication-optimal, but is not early-terminating. (The difference between this protocol and that of x.244 is that our protocol also locates the faults.) The other is early-terminating, but has high communication complexity. This protocol is similar to that of [Per88]. In section 5, we present a high level design of a fault isolator. This design is implemented by all the protocols in this paper and the protocols previously published. In section 6.1, we give an implementation which is early-terminating and with message complexity which is $O(n \log n)$. This message complexity is achieved by optimizing a certain combinatorial cover problem introduced in that section. We also show that every early-terminating protocol sends messages over paths that are included in such a cover. In sections 6.3–6.5, we present an early-stopping protocol that economizes on messages by avoiding sending messages over some of these paths in favorable executions. We conclude and discuss open problems in section 7.

2. The model. Our model is a modification of the standard model of dynamic networks [AE86, AAG87]. Since we are interested in detecting failures, we do not include recoveries. We assume some synchronization, namely, a known bound D on the transmission time over a nonfaulty link. We also introduce some new notations and assumptions, since we discuss communication only along a fixed path.

Denote the path as processors $1, \ldots, n$. Even though the task of our protocols is to deal with a message from processor 1 to processor n, our protocols send additional messages from intermediate nodes and to intermediate nodes. Consider a message ϕ (e.g., an acknowledgment) that was sent by processor $1 \leq j \leq n$ (the *sender* of ϕ) to another processor $1 \leq k \leq j$ (the *recipient* of ϕ). If k is not a neighbor of j, then message Φ needs to be received and resent by processors on the path between j and k. We use the following (somewhat "visual") notation to emphasize that in this case $k \leq j$: The protocol in processor i for $(k \leq i \leq j)$ interacts with the links by the

events $\texttt{Send}_i^{k\leftarrow j}[\phi]$, and $\texttt{Receive}_i^{k\leftarrow j}[\phi]$. Similarly, in the case that k is larger than j: $\texttt{Send}_i^{j\rightarrow k}[\phi]$, $\texttt{Receive}_i^{j\rightarrow k}[\phi]$. The events have their natural meanings.

In actual networks, following a message sent by i to $i+1$, the lower layer link level protocol will deliver a $\texttt{fail}_{i+1}$ event to i (i.e., the failure of the link from i to $i+1$) when more than $2D$ time passed without an acknowledgment whose sender is processor $i+1$, being received at i. For simplicity, we do not use such an event. Our protocol will detect a fault in such a case anyhow. We do not distinguish here between faults of links and those of processors except that when i detects a fault in the communication with $i+1$, we call it a failure of link $(i, i+1)$ (although it may be just the fault of processor $i+1$), and similarly in the case when $i+2$ detects a fault in its communication with processor $i+1$, we speak of a failure of link $(i+1, i+2)$. Since we do not allow recoveries, we assume that each link fails at most once.

We assume that whenever processor i receives a message, the message was indeed issued by the sender and later forwarded by every processor between the sender and i, and moreover that processor i is between the sender and the recipient. This assumption holds if the failures are nonmalicious, and otherwise can be enforced by cryptographic techniques.

AXIOM 1. *If* $\texttt{Receive}_i^{j\rightarrow k}[\phi]$ *occurs, then* $j < i \leq k$ *and for every* p *between* j *and* i, *previously* $\texttt{Receive}_p^{j\rightarrow k}[\phi]$ *and* $\texttt{Send}_p^{j\rightarrow k}$ *(as well as* $\texttt{Send}_j^{j\rightarrow k}$*) occurred. Similarly, if* $\texttt{Receive}_i^{j\leftarrow k}[\phi]$ *occurs, previously* $\texttt{Receive}_q^{j\leftarrow k}[\phi]$ *and* $\texttt{Send}_p^{j\leftarrow k}$ *(and* $\texttt{Send}_k^{j\leftarrow k}$*) occurred.*

The major deviation of our model from the standard dynamic network model is the addition of *synchronization* assumptions. Intuitively, these assumptions imply that the lower layer guarantees that it takes at most D time units from a $\texttt{Send}_i^{j\rightarrow k}[\phi]$ (respectively, $\texttt{Send}_i^{j\leftarrow k}[\phi]$) event till the corresponding $\texttt{Receive}_{i+1}^{j\rightarrow k}[\phi]$ (respectively, $\texttt{Receive}_{i-1}^{j\leftarrow k}[\phi]$) event occurs, unless the link (or one of the processors) failed. For simplicity of exposition, we use global time terminology and assume that all of the clocks have the same rate. We model the clocks by an "alarm clock" that generates an event every D time units. Namely, a "ticker" that sends interrupts every D time actually suffices for our protocols. We express this a little more formally in the following axiom.

AXIOM 2. *For every processor* i, *a TICK event occurs exactly once every* D *time units.*

Faulty processors whose number, f, is unknown are chosen by the adversary. In section 7, we discuss other meanings of f (and of the time complexity) in the cases in which there are no faults. A faulty link is one that is adjacent to a faulty processor. (This is a simplification. As mentioned above, in reality, it is possible that a processor will continue to function, and its other links will thus not be faulty.) The time for message delivery over faulty links is bounded by D, after which we say that a fault has occurred. Note that the adversary can choose to deliver messages over faulty links very quickly (e.g., in less than δ) or very slowly (e.g., even more than D). However, if a message is sent over a link at a time t and no acknowledgment arrives at time $t+2D$, a fault has occurred, and an algorithm is permitted to announce a detected fault. Axiom 3 bounds the message delivery time to δ over nonfaulty links.

AXIOM 3. *Assume that at time* t, *a* $\texttt{Send}_i^{j\rightarrow k}[\phi]$ *(*$\texttt{Send}_i^{j\leftarrow k}[\phi]$*) event occurs. Then, before* $t+\delta \leq t+D$, *if the link* $(i, i+1)$ *(respectively,* $(i-1, i)$ *is nonfaulty, then a* $\texttt{Receive}_{i+1}^{j\rightarrow k}[\phi]$ *(respectively,* $\texttt{Receive}_{i-1}^{j\leftarrow k}[\phi]$*) event occurs.*

3. The task. Intuitively, we want to deliver a message from a source processor to a destination processor. The protocol should detect any failure of links or processors which may delay (or disable) the transmission along the path. Both the transmission and the detection should be done in minimal time.

In our model, processor 1 is the *source* and processor n is the *destination*. The path consists of processors $1, \ldots, n$ and the links connecting them. We discuss only the transmission of a single message. There are standard methods to extend the results when many messages are transmitted, e.g., appending counters.

The operation of the protocol is based on transmitting the message and additional (control) information between the processors. Therefore, the protocol *accepts* a message from the higher layer in the source, and *delivers* a message to the higher layer in the destination; and for this purpose, it sends and receives (other) messages between processors along the path. Whenever confusion arises, we use the term *data message* (recall that in this paper we discuss only the case of a single data message, though, of course, multiple data messages can be handled by the same protocols) to refer to the message accepted from and delivered to the higher layer, and the term *control messages* to refer to the messages transmitted over the links (by the protocol). Note that some of the "control messages" that we use contain the "data message." (Some papers instead make the distinction between "messages" that arrive at the sender from a higher layer and delivered to a higher layer at the receiver, and "packets" that are sent in the network.)

In the protocols, we use two kinds of control messages (in addition to those that carry the data to be delivered): acknowledgments and disconnection notifications $Disc_i$. A disconnection notification $Disc_i$ means that processor i detected a failure in processor $i+1$ or in the link $(i, i+1)$. Such messages are normally flooded in the network, and therefore we assume that the protocol terminates when a nonfaulty processor sends $Disc_i$.

Loosely speaking, the protocols are resilient to a strong "adversary," which "knows" the state of every processor and every link, and "controls" the transmission delays of every link (up to D), the actual failures of the faulty links (delay larger than D), and the entire behavior of the faulty processors. However, the nonfaulty links never fail (always deliver messages in less than D) and the nonfaulty processors always operate according to the protocol. This resilience is formally stated in the following definition.

DEFINITION 1. *A protocol* $(P_1, \ldots, P_n)$ *is a* resilient forwarding faults detector *if for every selection of faulty processors and links, in every execution where every nonfaulty processor* i *executes* P_i*, the following conditions are kept.*

Detection: If the source and the destination are nonfaulty, then within a bounded time from the time the source accepts the message, either the message is delivered or a $Disc_i$ message is sent by some nonfaulty processor.

Location: If a $Disc_i$ message is sent by a nonfaulty processor, the link $(i, i+1)$ is faulty (that is, either processor i or processor $i+1$ is faulty).

Note that in the detection condition we do not require that the $Disc_i$ message be issued by a nonfaulty processor i.

In fact, a correct protocol should also guarantee the following.

Safety: If the source and the destination are nonfaulty, then the destination delivers a message only if this message was the one accepted at the source.

However, from Axiom 1 we assume that when a message is delivered to a processor it indeed knows who was its initiator.

Complexity measures. The complexity measure given here is the main difference between the model in the paper and the ones in previous works.

We consider time and communication complexities. Both measures are stated as functions of n, f, D, and δ, where the parameters are defined in Table 1. The complexities are the worst-case values for any execution over paths of length n with the actual (unknown) number f of faulty processors, bound D on the delay, and actual delay δ over nonfaulty links whose endpoints are also nonfaulty.

The time complexity is the maximum over all executions of the time since the source accepts the (data) message and until either the destination delivers that message or a failure is detected. To compute the time complexity, we consider only executions where the source and the destination processors are nonfaulty, since otherwise it is impossible to guarantee termination.

The communication complexity is the maximum number of transmissions of messages by nonfaulty processors. Messages transmitted by faulty processors are not counted.

4. Simple solutions. To demonstrate the problem, this section contains two simple protocols. The first, presented in section 4.1, is communication-optimal but has high time complexity. We point out the cause of the high time complexity. This weakness is removed in the protocol presented in section 4.2. The protocol of section 4.2 is early-terminating but has high communication complexity.

4.1. End-to-end fault detector. This protocol resembles the time-out mechanism of the data link. The data message ϕ is forwarded towards the destination (by $\texttt{Send}_i^{1\to n}[\phi]$ events). When the destination accepts the data message ($\texttt{Receive}_n^{1\to n}[\phi]$), it sends an acknowledgment backward ($\texttt{Send}_n^{1\leftarrow n}[Ack]$). Every processor $i < n$ checks whether $i+1$ is faulty. Namely, processor i expects to receive the acknowledgment ($\texttt{Receive}_i^{1\leftarrow n}[Ack]$) after $3(n-i)$ or less $TICK_i$ events since $\texttt{Send}_i^{1\to n}[\phi]$. If neither the acknowledgment nor the disconnection message is accepted, then i disconnects link $(i, i+1)$. Processor i forwards ϕ at most one $TICK_{i-1}$ event after $i-1$ forwarded it (assuming i, $i-1$, and $(i-1, i)$ are nonfaulty). Hence, and from the synchronization axioms, processor $i-1$ will accept either the acknowledgment or the disconnection from i at most $3(n-(i-1))$ $TICK_{i-1}$ events since forwarding ϕ. This means that nonfaulty processors will not be accidentally disconnected.

The communication complexity of the end-to-end detector is optimal ($3(n-1)$). The time complexity is $3(n-1)\cdot D$. (This is the complexity in the case that processor 2 is faulty.) When $\delta \ll D$, this time complexity is much inferior to the early-terminating time complexity $O(fD + n\delta)$, achieved by the protocols presented later.

4.2. Hop-by-hop detector. The end-to-end detector suffers from $O(nD)$ time complexity. We now describe a detector with time complexity $O(n\delta + fD)$, which is later shown to be optimal. We do this by extending the use of the acknowledgments. In the end-to-end detector, we use only one acknowledgment message, which signals the completion of the transmission. The idea is to use additional acknowledgments, which signal that the transmission is progressing properly.

In the hop-by-hop fault detector, we carry this idea to the extreme, thereby obtaining optimal time complexity. Namely, each processor sends an acknowledgment towards the source immediately upon receiving the message ϕ en route to the destination.

The improvement in time complexity is obtained by a tighter time-out check in the processors. Consider an arbitrary processor i. In the end-to-end detector, processor i

Constants of processor i:
  A_i (*integer*)): array of integer; (different for each protocol)
  (If $A_i(t) \neq \perp$, then after tD since processor i accepts ϕ, it sends Ack to processor $A_i(t)$.)

Variables of processor i:
{
  $done_i$: logical (init: *FALSE*); (*True* after i terminated (ϕ delivered or failure detected))
  $time_i$: integer; (The current time, i.e., number of $TICK_i$ events since start)
  $AckSet_i$: set of intervals (init: $\emptyset$); (Intervals of Ack which i received or sent)
}

FIG. 1 *Design of forwarding faults detector: declarations.*

disconnects from $i+1$ if it does not receive the acknowledgment from the destination n after more than $3(n-i)$ $TICK_i$ events since i forwarded ϕ toward the destination. (In fact, for this specific protocol we could have written $2(n-i)$, but $3(n-i)$ is used for compatibility with protocols presented later.) In essence, i waits the time needed in the worst case for the data message to reach its final destination n and for an acknowledgment to arrive from n to i. Consider the case that $i+1$ never forwards the message to $i+2$. Intuitively, this can be detected by i (using another protocol in which $i+2$ sends an acknowledgment to i) with $O(D)$ time. However, the end-to-end protocol detects the disconnection only in $\Omega((n-i)D)$ time.

In the hop-by-hop detector, processor i disconnects from $i+1$ if it does not receive any of the acknowledgments from $k = i+2, i+3, \ldots, n$ after more than $3(k-i)$ $TICK_i$ events. Intuitively, this mechanism guarantees that every $3D$ time unit the message processes toward the destination over an additional link (if this link is faulty; otherwise traversing the link costs δ time).

5. A design of resilient detectors. Both simple detectors presented in section 4 are extremely inefficient in one measure (either time or communication) and optimal in the other measure. In the rest of the paper, we present detectors which are efficient in both measures by providing reasonable trade-offs between them. In particular, the detectors are time-optimal up to a constant factor. It is also easy to present implementations of the design which are communication-optimal but with suboptimal time complexity.

Instead of presenting each detector "from scratch," we regard them all as implementations of a common "design." The end-to-end and the hop-by-hop detectors may also be regarded as implementations of this design. We prove that every implementation of this design, which satisfies a simple condition, is a resilient forwarding faults detector. Furthermore, we give a simple yet useful bound on the time complexity of implementations. In particular, these general results are used to prove that the detectors of section 4 are resilient and that the hop-by-hop detector is time-optimal.

The design is presented in Figures 1 and 2, and the explanations are given below.

Different implementations are defined by different selections of values for the array $A_{node}(time)$. Basically, if $A_{node}(time) \neq \perp$, then processor *node* will send an acknowledgment to processor $A_{node}(time)$ after *time* events of type D_{node} (i.e., additional D time units elapsed) occurred since *node* entered the protocol. The detectors of section 4 are implemented by using $A_i(t) = \perp$, except for the following.

For the end-to-end detector: use $A_n(0) = 1$.

For the hop-by-hop detector: for every $1 < i \leq n$, use $A_i(0) = 1$.

That is, in the end-to-end detector, only processor n initiates an acknowledgment

Algorithm for the source $i = 1$:

⟨B1⟩ On accepting ϕ:
⟨B2⟩ { $\texttt{Send}_1^{1\rightarrow n}[\phi]$;
⟨B3⟩ repeat WORKLOOP() until $done_i$;}

Algorithm for the destination $i = n$;

⟨C1⟩ On $\texttt{Receive}_n^{1\rightarrow n}[\phi]$: { deliver ϕ; $\texttt{Send}_n^{1\leftarrow n}[Ack]$; $done_n \leftarrow TRUE$;}

Algorithm for intermediate processors $1 < i < n$:

⟨D1⟩ On $\texttt{Receive}_i^{1\rightarrow n}[\phi]$:
⟨D2⟩ { $\texttt{Send}_i^{1\rightarrow n}[\phi]$; (forward message)
⟨D3⟩ if $A_i(0) \neq \perp$ then
⟨D4⟩ { $\texttt{Send}_i^{A_i(0)\leftarrow i}[Ack]$;
⟨D5⟩ $AckSet_i \leftarrow \{[A_i(0), i]\}$; }
⟨D6⟩ repeat WORKLOOP() until $done_i$; }

Procedure WORKLOOP():{
⟨E1⟩ On $\texttt{Receive}_i^{j\leftarrow l}[Ack]$ such that
$((\exists t)A_l(t) = j) \wedge ((\forall [j', l'] \in AckSet_i)(j < j') \vee (l' < l))$:
⟨E2⟩ { if $j = 1$ and $l = n$ then $done_i \leftarrow TRUE$;
⟨E3⟩ if $j < i$ then $\texttt{Send}_i^{j\leftarrow l}[Ack]$; (Forward Ack if i is not its destination)
⟨E4⟩ $AckSet_i \leftarrow AckSet_i \cup \{[j, l]\}$; }

⟨F1⟩ On $TICK_i$:
⟨F2⟩ { increment $time_i$;
⟨F3⟩ if $(A_i\ (time_i) \neq \perp) \wedge ((\forall [j', l'] \in AckSet_i)A_i\ (time_i) < j')$ then
⟨F4⟩ { $\texttt{Send}_i^{A_i(time_i)\leftarrow i}[Ack]$; $AckSet_i \leftarrow AckSet_i \cup \{[A_i(time_i), i]\}$ }
⟨F5⟩ if $\exists l > i$ and $(\exists t) t < time_i - 3(l - i)$ such that $i \geq A_l(t) \neq \perp$ and
$(\forall [j', l'] \in AckSet_i)(A_l(t) < j' \vee l' < l)$ then DISCONNECT() ;}

⟨G1⟩ On $\texttt{fail}_i$: DISCONNECT() ;

⟨H1⟩ On $\texttt{Receive}_i^{1\leftarrow j}[Disc_j]$:
⟨H2⟩ { $done_i \leftarrow TRUE$; $\texttt{Send}_i^{1\leftarrow j}[Disc_j]$; }
}

Procedure DISCONNECT() ; (Disconnect processor i from $i + 1$)
⟨I1⟩ { $\texttt{Send}_i^{1\leftarrow i}[Disc_i]$;
⟨I2⟩ $done_i \leftarrow TRUE$; }

FIG. 2 *Design of a forwarding faults detector for processor i.*

(thus $A_i(0) = \perp$ for every $i \neq n$). Moreover, processor n sends the acknowledgment to processor 1, and after 0 time, i.e., immediately on receiving the message. In the hop-by-hop detector, every processor sends an acknowledgment to processor 1 immediately upon receiving the message. For now, it will be easier to think of the more intuitive case that $A_i(t) = \perp$ for every $t > 0$. The usefulness of the case that $A_i(t) \neq \perp$ for $t > 0$ is demonstrated in subsection 6.3.

The operation begins when the source (processor 1) accepts the message ϕ from the higher layer. Each processor $i > 1$ begins operating when receiving ϕ from $i - 1$. When the message is received, every processor i (except for the destination) forwards it to the next processor $i + 1$. If $i = n$, the message is delivered to the higher layer, an Ack is sent to the source, and the protocol terminates. In the other processors ($i < n$), an Ack is sent to $A_i(0)$ (provided that $A_i(0) \neq \perp$) and i starts executing its

WORK_LOOP procedure. The WORK_LOOP is executed repeatedly, terminating only if a failure is detected or the *Ack* from n is accepted.

Processor i may also issue *Ack* later, sometime after it started operating. This is done according to the protocol-dependent $A_i(t)$ array. The value of $A_i(t)$ is the identity of the processor to which i should send an *Ack* at $t \cdot D$ after i began executing. We say that processor $A_i(t)$ is the *recipient* of this *Ack* message.

Some economizing is done at that point. (This economizing does not occur in the hop-by-hop and the end-to-end implementation; however, it proves very useful in the implementations of section 6.) Assume that $t_1 D$ time units after i started operating, it forwarded some *Ack* whose destination is some node k, and later, after $t_2 D$ time units, it is supposed to send an *Ack* to some $j > k$. (That is, $A_i(t_2) = j$.) Intuitively, this later *Ack* is no longer necessary, since the earlier *Ack* (that of time $t_1 D$) was already supposed to tell j (as well as k) that processor i received the message.

Thus, the *Ack* is sent only if its recipient $A_i(t)$ is farther from i than the most distant recipient of some previous *Ack* which i already forwarded. The set $AckSet_i$ holds all the intervals of *Ack* which i already forwarded. Processor i checks every $TICK_i$ event while in WORK_LOOP, if it should issue an *Ack*. The time since i began executing is approximated (in the variable $time_i$) by counting the number of $TICK_i$ events since i began executing.

The *Ack* messages are forwarded to their recipients by the nodes along the path. Namely, when processor i receives an *Ack* (from $i+1$) then i sends this *Ack* to processor $i-1$. There are three exceptions. First, if i is the recipient, then, of course, it does not forward the *Ack* any further. Second, processor i checks that this *Ack* is not "bogus," namely, that for some t and some $l > i$ the value of $A_l(t)$ is j. This prevents *Ack* messages from "maliciously" increasing message complexity.

The third exception is that the *Ack* is forwarded only if it may give some processor new information about the progress of the protocol. When a processor i receives an *Ack* which cannot give new information about the progress of the protocol, we say that the *Ack* is *redundant* and i does not forward it toward the recipient. Formally, an *Ack* from l to j is *redundant* when received by i if i already sent an *Ack* from some $l' \geq l$ to some $j' \leq j$.

Let us comment about nonredundant *Acks*. Note that for i to forward l's *Ack* it is unnecessary for i to learn anything new from that *Ack*. For example, it may be the case that i already received an *Ack* from $l+1$, and thus i already "knows" that l received the messages. Still, it may be the case that this *Ack* of $l+1$ was not forwarded to j, and that j does not "know" that the message arrived at $l+1$, or even at l. Thus, this *Ack* may be nonredundant.

In every $TICK_i$ event, processor i checks for time-out of any expected *Ack*. A time-out is a failure of $i+1$ to deliver the acknowledgment in time or to disconnect from $i+2$. Note that if i does not receive an *Ack* on time from any processor $l > i+1$ that is supposed to send an *Ack* to i, then $i+1$ has the opportunity to discover that before i does. In this case, if $i+1$ is not faulty, it must detect a disconnection of its link to $i+2$ and tell it to i. If this did not happen, then i concludes that $i+1$ is faulty.

Let us now elaborate on instruction ⟨F5⟩, where the mentioned check is done. Processor i checks if there exists some processor $l > i$ that was supposed to send an *Ack* to be received by i. That is $A_l()$ equals some j that is either i or smaller than i. As mentioned above, such an *Ack* message should pass i. If such an *Ack* was supposed to be sent, it may have not been sent yet, since the original message did not have time

to reach l yet. Alternatively, the *Ack* may have already been sent by l, but did not have enough time to reach i. That is why in instruction $\langle$F5$\rangle$ i also checks that there was enough time for the *Ack* to reach it if there were no disconnections. This is the meaning of the check on t in instruction $\langle$F5$\rangle$. Finally, it could be the case that the *Ack* was redundant, and thus was never sent or was omitted. Processor i verifies that this did not happen, by checking that other *Ack* messages it received or forwarded were not sent over intervals that contained l and j.

The check is done using the values of $A_i()$ of the different processor $j > i$, and the fact that $time_{i+1} \geq time_i - 2$. This fact follows from the synchronization axioms and the fact that ϕ is forwarded immediately. If the check fails, procedure DISCONNECT() is used to disconnect processor i from $i+1$ (since processor $i+1$ or the link to it, or both, are faulty).

5.1. Resilience of the design. In this subsection, we prove that every implementation of the design, which satisfies a simple condition, is a resilient forwarding faults detector. Most of the effort is required to prove the location property, which shows that a nonfaulty link $(i, i+1)$ between nonfaulty processors i, $i+1$ will not be disconnected. We begin with several simpler observations regarding such a link. First, we show that if $i+1$ finishes operating, then i will also finish operating after at most δ.

LEMMA 1. *Consider an execution in which link* $(i, i+1)$ *and processors* i *and* $i+1$ *are nonfaulty. Processor* i *sets* $done_i \leftarrow TRUE$ *at most* δ *after processor* $i+1$ *sets* $done_{i+1} \leftarrow TRUE$.

Proof. Processor $i+1$ sets $done_{i+1} \leftarrow TRUE$ only after $\texttt{Send}_{i+1}^{1 \leftarrow n}[Ack]$ or $\texttt{Send}_{i+1}^{1 \leftarrow j}[Disc_j]$. In any case, the corresponding `Receive` will occur at most after δ, by Axiom 3. Either message will cause $done_i \leftarrow TRUE$ unless it is already $TRUE$. □

We now prove that until processors i and $i+1$ finish, they are "nearly synchronized" in the values of the variable *time*.

LEMMA 2. *Consider an execution in which link* $(i, i+1)$ *and processors* i *and* $i+1$ *are nonfaulty. Whenever* $done_{i+1} = FALSE$, *then* $time_i - 2 \leq time_{i+1}$.

Proof. Processor i performs $\texttt{Send}_i^{1 \rightarrow n}[\phi]$ immediately upon starting operation. From Axiom 3, processor $i+1$ accepts ϕ at most $\delta < D$ later. As long as $done_{i+1} = FALSE$, the value of $time_{i+1}$ is the number of $TICK_{i+1}$ events since $i+1$ began executing. Similarly, the value of $time_i$ is at most the number of $TICK_i$ events since i began executing. The claim follows from Axiom 2. □

We now prove that if $i+1$ sends an *Ack* to i, then this *Ack* will not be ignored by (statement $\langle$E1$\rangle$ of the design) processor i.

LEMMA 3. *Consider an execution in which link* $(i, i+1)$ *and processors* i *and* $i+1$ *are nonfaulty. Assume that a* $\texttt{Send}_{i+1}^{j \leftarrow l}[Ack]$ *occurs at time* τ *for* $j \leq i$, $l \geq i+1$, *while* $done_i = done_{i+1} = FALSE$. *Then, at time* $\tau + \delta$, *either* $done_i = TRUE$ *or* $(\exists [j', l'] \in AckSet_i)(j' \leq j < l \leq l')$.

Proof. From Axiom 3, event $\texttt{Receive}_i^{j \leftarrow l}[Ack]$ will occur before time $\tau + \delta$. Assume that at time $\tau + \delta$, the following holds: $done_i = FALSE$. Therefore, statement $\langle$E1$\rangle$ is executed in i upon $\texttt{Receive}_i^{j \leftarrow l}[Ack]$. However, since $i+1$ is nonfaulty, it also used statement $\langle$E1$\rangle$, $\langle$D3$\rangle$, or $\langle$F3$\rangle$ before $\texttt{Send}_{i+1}^{j \leftarrow l}[Ack]$. Hence $(\exists t) A_l(t) = j$. This proves the claim, since if the other check of statement $\langle$E1$\rangle$ fails, then the claim holds trivially; and if both checks succeed, then statement $\langle$E4$\rangle$ is executed and the claim follows. □

We now prove the core of the location property. This claim is still slightly weaker than the location property, since it deals only with the case that i sends the $Disc_i$ message.

LEMMA 4. *Consider an execution in which link* $(i, i+1)$ *and processors* i *and* $i+1$ *are nonfaulty. Then the execution does not contain a* $\texttt{Send}_i^{1 \leftarrow i}[Disc_i]$.

Proof. The proof is by contradiction. Assume that $\texttt{Send}_i^{1 \leftarrow i}[Disc_i]$ occurs at time τ. By definition, a $\texttt{fail}_i$ event does not occur. Therefore, statement $\langle$F5$\rangle$ in the design caused the $\texttt{Send}_i^{1 \leftarrow i}[Disc_i]$ event at τ. Namely, while $done_i = FALSE$, a $TICK_i$ event occurs where for some $l > i$ and $t < time_i - 3(l-i)$ holds $i \geq A_l(t) \neq \perp$ and $(\forall [j', l'] \in AckSet_i)(A_l(t) < j' \vee l' < l)$.

The proof is based on considering the state of $i+1$ at $\tau - D$. From Lemma 1, if $done_{i+1}$ is $TRUE$ at $\tau - D$, then $done_i$ is $TRUE$ at $\tau - D + \delta < \tau$. Assume, therefore, that $done_{i+1} = FALSE$ at $\tau - D$. From Axiom 2, a $TICK_i$ event occurred at $\tau - D$ with $time_i$ less by one than at τ; namely, at time $\tau - D$ the following holds: $time_i = t + 3(l-i)$. (Recall that t, l, i are integers.) From Lemma 2, the maximum value of $time_i - time_{i+1}$ until $\tau - D$ is 2 (since $done_{i+1}$ is $FALSE$). Hence, the $TICK_{i+1}$ event in which $time_{i+1} \leftarrow t + 1 + 3(l - (i+1))$ is before $\tau - D$. Denote the time of this $TICK_{i+1}$ event by τ'. We derive a contradiction from considering the state of $i+1$ at τ'. Consider the two cases: $l > i+1$ and $l = i+1$.

We first deal with the case $l > i+1$, i.e., processor $i+1$ failed to forward to i the Ack from l (to some $A_l(t) \leq i$) or to disconnect from $i+2$. At τ', since $done_{i+1}$ is $FALSE$, then $(\exists [j', l'] \in AckSet_{i+1})(j' \leq A_l(t) \leq i < l \leq l')$. (Otherwise, processor $i+1$ would have invoked statement $\langle$F5$\rangle$.) An interval $[j', l']$ is added to $AckSet_{i+1}$ only by one of statements $\langle$F4$\rangle$, $\langle$D5$\rangle$, or $\langle$E4$\rangle$. Since $j' < i+1$, exactly before any of these statements is executed, a $\texttt{Send}_{i+1}^{j' \leftarrow l'}[Ack]$ occurs (see Figure 2). Hence, at (or before) $\tau - D$ a $\texttt{Send}_{i+1}^{j' \leftarrow l'}[Ack]$ occurs with $j' \leq A_l(t) < l \leq l'$. From this follows, using Lemma 3, that at τ there must be some $[x', y'] \in AckSet_i$ such that $x' \leq j' < l' \leq y'$. This contradicts the assumption that statement $\langle$F5$\rangle$ was invoked at τ.

We now deal with the case $l = i+1$. First, assume $t = 0$. Namely, $\texttt{Send}_{i+1}^{j \leftarrow i+1}[Ack]$ occurs when $i+1$ starts operating, i.e., at most δ after i starts operating. Hence, from Lemma 3, there will be some $[j', l'] \in AckSet_i$ such that $j' \leq j$ and $i+1 \leq l'$, after at most an additional δ (i.e., 2δ since i started). From Axiom 2 and the design, the value of $time_i$ at 2δ since i started is at most 2. Hence, when $time_i > 3$ statement $\langle$F5$\rangle$ is not invoked by i with $l = i+1$ and $t = 0$.

Assume, therefore, that $t > 0$ and $l = i+1$. Recall that $done_{i+1} = FALSE$ at τ'. Processor $i+1$ checks, at τ', the condition of statement $\langle$F3$\rangle$. If the condition holds, then statement $\langle$F4$\rangle$ is executed, i.e., a $\texttt{Send}_{i+1}^{j \leftarrow i+1}[Ack]$ occurs at τ', and the contradiction again follows from Lemma 3.

Assume that the condition of $\langle$F3$\rangle$ does not hold, namely, $(\exists [j', l'] \in AckSet_{i+1})$ $j' \leq j$. Then, previously one of statements $\langle$F4$\rangle$, $\langle$D5$\rangle$, or $\langle$E4$\rangle$ was executed, adding $[j', l']$ to $AckSet_{i+1}$. Since $j' < i+1$, exactly before any of these statements is executed, a $\texttt{Send}_{i+1}^{j' \leftarrow l'}[Ack]$ occurred (see Figure 2). Hence, at or before τ', a $\texttt{Send}_{i+1}^{j' \leftarrow l'}[Ack]$ occurs with $j' \leq j$. The contradiction follows from Lemma 3. □

We now complete the proof that every implementation of the design, which satisfies a simple condition, is a resilient forwarding faults detector.

THEOREM 5. *Every implementation* $A()$ *of the design such that* $A_n(0) = 1$ *is a resilient forwarding faults detector.*

Proof. The safety property follows immediately from the design and Axiom 1. To prove the detection property, consider an execution where no message is delivered,

and the source and destination are nonfaulty. Since the destination is nonfaulty and no message is delivered, there will be no $\texttt{Send}_n^{1\leftarrow n}[Ack]$ event. From Axiom 1, there will be no $\texttt{Receive}_1^{1\leftarrow n}[Ack]$ event. We assumed that $A_n(0) = 1$. Hence, after $time_1 = 3(n-1)$, statement $\langle$F5$\rangle$ sets $done_1 \leftarrow TRUE$ (unless $done_1 = TRUE$ already). But since processor 1 did not accept Ack from n, when it sets $done_1 \leftarrow TRUE$ it also did $\texttt{Send}_1^{1\leftarrow 1}[Disc_j]$.

We now prove the location property. Consider an execution where $Disc_i$ is sent by a nonfaulty processor j, and i is nonfaulty. The only event in which j sends, according to the design, a $Disc_i$ message, is by a $\texttt{Send}_j^{1\leftarrow i}[Disc_i]$ event. From Figure 2, a nonfaulty processor $j \neq i$ sends $Disc_i$ only if a $\texttt{Receive}_j^{1\leftarrow i}[Disc_i]$ occurred. Since $j \neq i$, it follows from Axiom 1 that $j < i$ and that before the $\texttt{Send}_j^{1\leftarrow i}[Disc_i]$ event, a $\texttt{Send}_i^{1\leftarrow i}[Disc_i]$ event occurred. From Lemma 4, either $i+1$ or the link $(i, i+1)$ or both are faulty. □

Since both detectors of section 4 are implementations of the design with $A_n(0) = 1$, we conclude with the following corollary.

COROLLARY 6. *The end-to-end and the hop-by-hop detectors are both resilient forwarding faults detectors.*

5.2. A bound on time complexity. In this subsection, we show a bound on the time complexity of implementations of the design. This bound suffices to show that the implementations we present later, as well as the hop-by-hop detector, are early-terminating.

Let $T_A(n, f, \delta, D)$ (respectively, $T_{\text{opt}}(n, f, \delta, D)$) be the maximal time since an execution of implementation A starts (respectively, since an execution of a time-optimal implementation starts) and until the destination receives the message ϕ or an error is detected. We want a bound for the worst ratio $\frac{T_A(n,f,\delta,D)}{T_{\text{opt}}(n,f,\delta,D)}$ over every selection of n, f, δ, and D.

Obviously, we can bound T_A by the time required to reach from 1 to any processor i, plus the time required to reach from i to n. Likewise, we bound T_A by the sum of times required to reach from 1 to 2, from 2 to 3, and so on. Also, if some processor k between j and l is nonfaulty, then the time to reach from j to l is bounded by the time to reach from j to k plus the time to reach from k to l. Note that if every processor and link from j to l is nonfaulty, then it takes exactly $(l-j)\delta$ to reach from j to l. Therefore, we can bound the time complexity by regarding the worst ratio of the times required to reach from processor j to l when all of the processors between j and l are faulty.

The best time to reach from j to l is achieved if j expects l to acknowledge immediately; then the delay is $3D(l-j)$. The time of a specific implementation A is the minimal value of $3D(l'-j)+t$, where l' is a processor after l which sends at time t, according to A, an acknowledgment whose recipient is $j' \leq j$. We call this ratio the *covering factor* of the interval $[j, l]$.

DEFINITION 2. *Let j, l be processors such that $1 \leq j < l \leq n$. The covering factor of $[j, l]$ with respect to $A_i(t)$ is denoted $F_{[j,l]}(A())$ and defined as follows:*

$$F_{[j,l]}(A()) \stackrel{\text{def}}{=} \min_{t',j',l'} \left\{ \frac{(3(l'-j)+t') | (A_{l'}(t') = j') \wedge (j' \leq j < l \leq l')}{3(l-j)} \right\}. \tag{1}$$

Note that $A_i(t)$ is a set of intervals. The covering factor $F_{[j,l]}(C)$ for any set of intervals C (called a *cover*) is defined similarly.

We now define the covering factor of an implementation, which is the worst covering factor of any interval.

DEFINITION 3. *The covering factor with respect to $A_i(t)$ is denoted $F(A())$ and is defined as follows:*

$$F(A()) \stackrel{\text{def}}{=} \max_{1 \le j < l \le n} F_{[j,l]}(A()). \tag{2}$$

The covering factor $F(C)$ for a cover C is defined in a similar way.

The covering factor of an implementation gives an upper bound on the time complexity of this implementation, as follows.

THEOREM 7. *The time complexity of every implementation $A()$ of the design is $O(n\delta + f \cdot F(A()) \cdot D)$.*

Proof. For simplicity, we ignore link failures, which may be emulated by corresponding processor failures. Also, we assume that the source accepted the message from the higher layer at time 0. Finally, we assume that the source and the destination are nonfaulty, since the time complexity is defined under this assumption. We use the following notations.

Notations. Let f_i be the number of faulty processors before processor i. Also, let τ_i denote the time when processor i received the message and entered the protocol, i.e., the time of $\texttt{Receive}_i^{1 \to n}[\phi]$.

We prove the following claim for every processor i: if i is nonfaulty, then before time $i\delta + 8D \cdot F(A()) \cdot f_i$ one of the following happens: either some nonfaulty processor sent $Disc_j$ for some j, or processor i received the message and entered the protocol. The theorem follows by considering $i = n$.

The claim is trivial for $i = 1$. We now prove the claim for processor i assuming that it holds for every processor before i. If processor i is faulty, then the claim holds trivially. If both processors i and $i-1$ are nonfaulty, then the claim holds since processor $i-1$ forwards the message to processor i immediately.

Assume, therefore, that processor i is nonfaulty, but processor $i-1$ is faulty. Let i' be the last nonfaulty processor before i, i.e., $i' < i$ and every processor in $[i'+1, i-1]$ is faulty. By the induction hypothesis, before time $i' \cdot \delta + 8D \cdot F(A()) \cdot f_{i'}$, either some nonfaulty processor sent $Disc_j$, for some j, or processor i' received the message. In the first case, where some nonfaulty processor sent $Disc_j$, the claim for i holds trivially.

Assume, hence, that processor i' received the message before $i' \cdot \delta + 8D \cdot F(A()) \cdot f_{i'}$. Denote by τ the time when the $1 + 3 \cdot (i - i') \cdot F(A())^{th}$ event of the kind $TICK_{i'}$ occurs since i' received the message. From Axiom 2, time τ is not more than $3 \cdot (i - i') \cdot F(A()) \cdot D + 2D$ time units since processor i' received the message, namely,

$$\tau \le i' \cdot \delta + 8D \cdot F(A()) \cdot f_{i'} + 3(i - i') \cdot F(A()) \cdot D + 2D.$$

Since $f_i = f_{i'} + (i - i' - 1)$ and $i' + 1 < i$, then

$$\tau < i \cdot \delta + 8D \cdot F(A()) \cdot f_i.$$

Hence, it suffices to show that at time τ, either processor i received the message or processor i' sent $Disc_j$ for some j. We now consider two cases, depending on the state of processor i' at τ. The first case we consider is that at time τ holds $done_{i'} = TRUE$; later we deal with the other case.

Since at τ holds $done_{i'} = TRUE$, then previously either statement $\langle$H2$\rangle$ of procedure DISCONNECT() or statement $\langle$E2$\rangle$ was executed at processor i'. If procedure

DISCONNECT() was executed, then $\texttt{Send}_{i'}^{1 \leftarrow i'}[Disc_{i'}]$ occurred already, and the claim for i follows. Similarly, if statement $\langle$H2$\rangle$ was executed, then $\texttt{Send}_{i'}^{1 \leftarrow j}[Disc_j]$ occurred already for some j, and the claim for i follows.

If statement $\langle$E2$\rangle$ was executed, then statement $\langle$E3$\rangle$ was also executed, since $j = 1 < i'$. Hence, $\texttt{Send}_{i'}^{1 \leftarrow n}[Ack]$ occurred. This happens only if i previously received the message, and then the claim for i follows.

Consider now the second case, where at time τ it holds that $done_{i'} = FALSE$. Hence, at τ processor i' executes $\langle$F2$\rangle$, after which $time_{i'} = 1 + 3 \cdot (i - i') \cdot F(A())$.

By Definition 3, $F(A()) \geq F_{[i',i]}(A())$. By Definition 2, there are l, t such that $A_l(t) \leq i'$ and $i \leq l$ and

$$F_{[i',i]}(A()) = \frac{3(l - i') + t}{3(i - i')}.$$

Hence, when processor i' executes $\langle$F5$\rangle$ at time τ, then

$$\begin{aligned} time_{i'} &= 1 + 3 \cdot (i - i') \cdot F(A()) \\ &> 3 \cdot (i - i') \cdot F_{[i',i]}(A()) \\ &= 3 \cdot (i - i') \cdot \frac{3 \cdot (l - i') + t}{3(i - i')} \\ &= 3 \cdot (l - i') + t. \end{aligned}$$

Hence, $t < time_{i'} - 3 \cdot (l - i')$ at time τ. Namely, at time τ, either $\texttt{Send}_{i'}^{j' \leftarrow l'}[Ack]$, $\texttt{Receive}_{i'}^{j' \leftarrow l'}[Ack]$ occurred with $j' \leq A_l(t) \leq i' < l \leq l'$, or processor i' executes procedure DISCONNECT() due to $\langle$F5$\rangle$. If processor i' executes procedure DISCONNECT(), then $\texttt{Send}_{i'}^{1 \leftarrow i'}[Disc_{i'}]$ occurs and the claim follows. On the other hand, from Axiom 1, if $\texttt{Receive}_{i'}^{j' \leftarrow l'}[Ack]$ occurred, then $\texttt{Send}_{i}^{j' \leftarrow l'}[Ack]$ occurred before, and this happens only after i entered the algorithm. □

We now observe that the optimal time complexity is bounded by $(n - f)\delta + fD$.

LEMMA 8. *Every forwarding faults detector has a run with time complexity at least* $(n - f)\delta + fD$.

Proof. Consider the execution where processors $2, \ldots, 2 + f - 1$ are faulty. The fault merely causes the delay upon forwarding the message through these processors to be D instead of δ. □

We deduce the following.

COROLLARY 9. *A detector that implements the design such that* $F(A())$ *is bounded by a constant is early-terminating. In particular, the hop-by-hop detector is early-terminating.*

Proof. The general claim follows from Theorem 7 and Lemma 8. The hop-by-hop detector has $A_l(t) = 1$. By definition, for every j, l, $F_{[j,l]}(A()) = 1$; hence, $F(A()) = 1$. The claim follows. □

6. Optimal time and communication-efficient implementations. In this section we present three implementations of the design, which ensure early termination (time complexity $O(n\delta + fD)$) and efficient ($O(n \log n)$ in the worst case) communication. Each implementation is a refinement of the previous one.

Throughout this section, we make the simplifying assumption that $n - 1$ is an even power of 2. This at most quadruples the complexities of the solutions, when applied to paths were $n - 1$ is not an even power of 2. In these cases, the source

processor may "play the rule" of a sufficient number of processors to extend n so that $n-1$ will become an even power of 2.

6.1. The immediate acknowledgments implementation. We begin by considering implementations where every acknowledgment is sent immediately upon receiving the message. For such an implementation $A()$, the covering factor of an interval $[j,l]$, as defined in (1), has the following simplified form:

$$F_{[j,l]}(A()) = \min_{j',l'} \left\{ \frac{l'-j \mid (A_{l'}(0) = j') \wedge (j' \le j < l \le l')}{l-j} \right\}. \tag{3}$$

We are interested in implementations which are early-terminating. From Corollary 9, such are the implementations where $F(A())$ is bounded by a constant. Namely, for every $[j,l]$ where $j < l$ there is some interval $[j',l']$ such that $A_{l'}(0) = j'$ and $j' \le j < l \le l'$ and $\frac{l'-j}{l-j}$ is bounded by a constant.

A natural selection of $A()$ is to send acknowledgments over intervals of lengths which are powers of two, i.e., $1, 2, 4, \ldots, (n-1)$. Let us describe the set of intervals used (see also a definition below). For every length $2^k \le (n-1)$ there are two types of intervals. Intervals of the first type start at every processor in a position of the form $r \cdot 2^k + 1$ for every r for which such a processor exists. For example, if $2^k = \frac{n-1}{2}$ one such interval starts at processor $2^k + 1$ (for $r = 1$) and the other starts at n. All acknowledgment intervals of length 2^k that start at a processor, i, end at processor $i - 2^k$. For example, the interval that starts at processor $2^k + 1$ ends at processor 1. Note, for example, that a subpath (of the message path) of length L such that $\frac{n}{4} < L < \frac{n}{2}$ is covered with a covering factor of less than 2 if and only if it is contained in one of the two acknowledgment intervals (described above) of length $\frac{n}{2}$. However, if it partially intersects with both, then only the end-to-end acknowledgment interval covers it. This effect becomes more damaging to the covering factor when we consider a shorter subpath.

To alleviate this effect, we introduce the second type of acknowledgment interval used. A subpath not covered (with a covering factor of 2 or less) by an interval of the first type will be covered (with a covering factor of 4 or less) by an interval of the second type. The intervals of the second type (still of length 2^k) start at $(r+\frac{1}{2}) \cdot 2^k + 1$ for every $r > 0$ for which there is such a processor. For example, for $2^k = \frac{n-1}{2}$ there is only one interval of the second type, and it starts at $\frac{3}{4}(n-1)+1$. The acknowledgment intervals which are "shifted," and start at $(r+\frac{1}{2}) \cdot 2^k + 1$, are needed to cover intervals which span over the connection between the acknowledgment intervals of the first kind, e.g., intervals which include processor $\frac{n-1}{2}+1$. This selection of acknowledgment intervals ensures that every interval $[j,l]$ is covered by an acknowledgment interval which is not "much larger," as we now formalize.

We specify this implementation in (4). We now prove that $F(A())$ is bounded by a constant, and hence that it is early-terminating.

$$A_i^{(1)}(0) \stackrel{\text{def}}{=} i - \max_{k \ge 0} \left\{ 2^k \,\middle|\, (\exists r \in \mathbb{N}) \left((i = 2^k \cdot r + 1) \text{ or } \left(i = 2^k \cdot \left(r + \frac{1}{2} \right) + 1 \right) \right) \right\}. \tag{4}$$

(No acknowledgment interval is defined for the case that $t > 0$.)

LEMMA 10. *The implementation $A^{(1)}()$ is early-terminating.*

Proof. From Corollary 9, it suffices to show that $F(A^{(1)}())$ is bounded by a constant. The proof is by showing that every interval $[j,l]$ is "covered" by an interval in $A^{(1)}()$ whose length is at most four times $l - j + 1$.

Consider an interval $[j,l]$ such that $1 \leq j < l \leq n$. Without loss of generality, assume that $l - j < \frac{n-1}{4}$. Let k be the minimal such that $l - j < \frac{2^k}{4}$, i.e., $k \stackrel{\text{def}}{=} \lceil \log_2(4 \cdot (l - j + 1)) \rceil$. Let r be the minimal such that $l < 2^k \cdot r + 1$. Namely, $2^k \cdot (r-1) + 1 \leq l$.

If $2^k \cdot (r-1) + 1 < j$, then $[j,l]$ is covered by the acknowledgment interval $[A^{(1)}_{2^k \cdot r+1}(0),\ 2^k \cdot r + 1]$. From (3), $F_{[j,l]}(A^{(1)}()) \leq 4$.

Assume, therefore, that $j \leq 2^k \cdot (r-1) + 1$. In this case, $2^k \cdot (r - \frac{3}{2}) + 1 < j \leq l < 2^k \cdot (r - \frac{1}{2}) + 1$, since $l - j < 2^{k-2}$ and $2^k \cdot (r-1) + 1 \leq l$. Again, from (3), $[j,l]$ is covered by the acknowledgment interval whose last processor is $2^k \cdot (r - \frac{1}{2}) + 1$. Hence, again $F_{[j,l]}(A^{(1)}()) \leq 4$. □

To complete the analysis of this implementation, we note that the communication complexity is obviously the total length of the intervals, which is $O(n \log(n))$.

6.2. A tight lower bound for oblivious protocols. We consider protocols whose operations include forwarding the message, computing, using time-outs, and sending acknowledgments. It is natural to classify such protocols by the way they handle the acknowledgments. An important subset, termed *oblivious protocols*, sends any acknowledgments they wish to send immediately, without delaying it. Similarly, if they receive an acknowledgment to be forwarded, they forward it immediately. All the previously published protocols, as well as all the protocols up to this point in this paper, are oblivious. In the design, this family of protocols is captured by having $A_i(t) = 0$ for every $t \neq 0$.

THEOREM 11. *An oblivious protocol is time-optimal if and only if $F(A())$ is a constant in it.*

Proof. The "if" part follows from Corollary 9. For the "only if" part, consider an oblivious protocol for which $F(A())$ is some $f(n)$. (Notice that for an oblivious protocol $F(A())$ does not depend on t.) Let j, l be two processors such that $F_{[j,l]}(A()) = f(n)$ (see Definitions 2 and 3) and let l' be the one mentioned in Definition 2. Consider the case that all the processors in the closed interval $[j,l]$ are faulty, but no other processor is faulty. Consider the state of knowledge (see, e.g., [HM90]) of processor j' at any time before $(l' - j)D$. Clearly there is a run where it did not receive any message from a nonfaulty processor $p > l$. Thus, the state of knowledge of processor j at such a time is the same as in the case that all the processors in the interval $[j,l']$ are faulty. The theorem now follows from Lemma 8. □

We now prove that the set of intervals used by the previous implementation is optimal in the sum of the lengths of the intervals. Note that this sum determines the message complexity of the protocol. Let an *interval* $c = [i,j]$ for $1 \leq i < j \leq n$ be the set $\{i, i+1, \ldots, j\}$. The *length* of interval c, denoted $L(c)$, is $j - i$. An *interval cover* C of an interval $[1,n]$ is a set of intervals that includes interval $[1,n]$.

The claim is that for an interval cover C, if $F(C)$ is a constant, then $L(C) = \Theta(n \log n)$, where $L(C) = \sum_{c \in C} L(c)$.

Intuitively, to cover long intervals, the cover must contain some long intervals. In fact, a few long intervals in the cover suffices to cover every long interval. The main observation in the proof is that a long interval cannot cover too many short intervals. Thus, additional intervals must be introduced into the cover. These intervals may be short and thus, it may seem that the contribution of each of them to $L(C)$ is small. However, many short intervals are needed in the cover to cover all the short intervals. Thus, the total contribution of the short intervals to $L(C)$ is large.

THEOREM 12. *For every interval cover C such that $F(C)$ is a constant, the total*

length $L(C)$ is $\Omega(n \log n)$.

Proof. Without loss of generality, assume $(\exists k)n - 1 = 2(F(C))^k$. (Otherwise, prove for $n' > n$ for which there exists such k; this adds only a constant factor.)

Let I_x be the set of intervals of path $[n, 1]$ that contain (each) exactly x links. The key observation is that any single interval in C (even a very long one) can cover at most $(F(C)-1)x+1 \leq (F(C)-1)x+x \leq F(C)x$ intervals in I_x with covering factor of $F(C)$. Consider the sets $I_{x(i)}$, where $x(i) = \frac{n-1}{2F(C)^{i+1}}$ for $1 \leq i \leq -1 + \log_{F(C)}(\frac{n-1}{2})$. Clearly $|I_{x(i)}| \geq \frac{n-1}{2}$. Thus, the observation implies that C must contain at least $\frac{|I_{x(i)}|}{F(C)x(i)} \geq \frac{(n-1)2F(C)^{i+1}}{2(n-1)F(C)} \geq F(C)^i$ intervals, the length of each at least $\frac{n-1}{2F(C)^{i+1}}$.

We now partition C into sets of intervals and give a lower bound for the total sum of each set. The sum of these bounds will later give us a lower bound for the total sum of C. Let $C_0 \subset C$ include the interval $[n, 1]$. For $i = 1$ we have that $x(i) = \frac{n-1}{2F(C)^2}$. To cover $I_{x(1)}$ cover C must include $F(C)$ intervals, the length of each is at least $x(1) = \frac{n-1}{2F(C)^2}$. One of them can be the $[n, 1]$ interval, but additional $F(C) - 1$ intervals are needed. Let $C_1 \subset C$ be a set of such additional intervals. Note that $C_1 \cap C_0 = \emptyset$. The total sum of C_1 is at least $\frac{(F(C)-1)(n-1)}{2F(C)^2}$. We continue to construct the C_i's inductively. For intervals in $I_{x(i)}$, C must include at least $F(C)^i$, where the length of each is at least $\frac{n-1}{2F(C)^{i+1}}$. As before, $F(C)^{i-1}$ such intervals are already included in the sets $C_0, C_1, C_2, \ldots, C_{i-1}$. Thus, we can construct a set $C_i \subset C$ such that $\forall_{1 \leq j < i} C_i \cap C_j = \emptyset$ and the cardinality of C_i is $F(C)^i - F(C)^{i-1}$. Thus, the total sum of C_i is at least $\frac{F(C)^{i-1}(F(C)-1)(n-1)}{2F(C)^{i+1}} = \frac{(n-1)(F(C)-1)}{2F(C)^2}$. Since we can construct $r(\log_{F(C)} n)$ such disjointed sets, the total length of C is $\Omega(\frac{n}{F(C)} \log_{F(C)} n)$. Since $F(C)$ is a constant, the theorem follows. □

6.3. The ideas behind the adaptive implementations. The immediate-Ack implementation sends $r(n \log n)$ messages even in executions where there are no faults ($f = 0$) and δ is small. We next present implementations of the design which are early-terminating but use less messages when f and δ are small. The worst-case communication complexity remains $O(n \log n)$.

Recall that in the design (Figure 2), processor i does not forward an *Ack* from processor l to processor j if this *Ack* is redundant, namely, if i already forwarded an *Ack* from some $l' \geq l$ to some $j' \leq j$. However, in the immediate-Ack implementation, each processor sends its acknowledgments upon beginning execution. Therefore, the immediate-Ack implementation of the design sends all of its acknowledgments in every execution (without failures).

However, if some of the processors *delay* issuing their acknowledgments, then it is possible that a delayed acknowledgment will become "redundant." For example, suppose that each of the processors delay all their acknowledgments by D except for the destination. If *Ack* from the destination n reaches the source 1 before D since the source started the protocol, then all of the other acknowledgments become redundant.

Obviously, the delay until the acknowledgments are sent increases the maximal time until the protocol terminates. For example, if the processors i s.t. $1 < i < n$ delay their acknowledgments by $2Dn$, then the resulting implementation has the same time complexity, up to a constant, as the end-to-end fault detector.

The decrease in message complexity comes, therefore, at the cost of an increase in the time complexity. The implementations presented in this section are early-terminating, since the delay is bounded by twice the time required for the immediate-Ack implementation. In fact, the adaptive implementations, presented in the rest of

this section, are modifications of the immediate-Ack implementation.

The first adaptive implementation saves messages mainly if there are no faulty processors. The number of acknowledgments sent is a function of the time complexity, which is lowest when there are no faults. The second adaptive implementation saves messages when there are faulty processors. In the following subsections we explain each of these implementations and analyze their properties.

6.4. Send *Ack* only when really needed. The first idea is to wait, as long as possible, before sending intermediate acknowledgments (i.e., *Ack* from $l < n$ to $j \geq 1$). The longer the delay in sending an intermediate acknowledgment, the larger the hope that the acknowledgment from n to 1 will make the intermediate acknowledgment redundant. In this subsection, we study how much any specific processor can wait without increasing the time complexity "too much." By implementing the idea of this subsection, the communication complexity is reduced to $O(n \log(1 + \frac{n\delta}{D}))$ if $f = 0$. In the next subsection, we show how to keep the communication complexity low, even when $f > 0$.

To demonstrate the idea, let us first investigate the case that there is exactly one faulty processor i (although the algorithm must still be early-terminating for any number of faults). Early termination is assured in the immediate-Ack implementation since i is covered by the interval $[i-1,\ i+1]$ of length two in $A^{(1)}()$. Note that an *Ack* from $i+1$ to $i-1$ is allowed to take at least D times if i is faulty. (Recall also that time complexity in the presence of one fault is $\Omega(D)$.) Hence, if $i+1$ waits D times before sending the *Ack*, it at most doubles the protocol's time complexity in executions with one fault. In executions with no faults, no acknowledgment is needed (although acknowledgments must be sent, since the number of faults is not known). Thus, a delay in any acknowledgment does not increase the time complexity in such executions. Finally, in executions with more than one fault, the acknowledgments over intervals of length 2 do not help, so any delay in them cannot increase the time complexity.

The adaptive detector sends the acknowledgments of the intervals of length 2 of $A^{(1)}()$, but only after waiting $O(D)$ for *Ack* from n. Early termination is always obtained, as explained above. However, in executions with exactly one fault (and a small δ), this achieves both early termination and optimal message complexities. The optimal message complexity is achieved, in this case, since the acknowledgment from processor n arrives at every other processor before it sends any acknowledgment of its own. Thus, all the other acknowledgments become redundant and are not sent.

In general, if there are at most f faults, then the time complexity with f faulty processors is at least $O(f \cdot D)$. Hence, the adaptive implementation delays sending the acknowledgments of intervals of length f by $f \cdot D$.

We now formally present the implementation of this subsection. It is easy to see that the covering factor is bounded by a constant. Hence, by Corollary 9, the implementation is early-terminating; we later show this formally. For simplicity, we assume that $n-1$ is an even power of 2. For $1 < i \leq n$ and $0 \leq t < n-1$, define $A_i^{(2)}(t)$ as follows:

$$A_i^{(2)}(t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } i = n, \\ i - t & \text{if } (\exists k) t = 2^k \wedge (\exists r \in \mathbb{N})\ i = rt + 1, \\ i - t & \text{if } (\exists k) t = 2^k \wedge (\exists r \in \mathbb{N})\ i = (r + \frac{1}{2})t + 1, \\ \perp & \text{otherwise.} \end{cases} \tag{5}$$

The implementation of this subsection achieves communication complexity $O(n \log(\frac{n\delta}{D}))$ if $f = 0$. We do not prove this here, since it is a corollary of a theorem presented in what follows.

6.5. Saving acknowledgments. In this section, we improve the previous implementation to save messages in a particular bad scenario. This improves the worst-case message complexity for $f > 0$.

Recall that acknowledgments are sent (in the previous implementation) at times $2^k \cdot D$ for $k = 0, 1, 2, \ldots$. At first glance, this seems to imply that the communication complexity is $O(n)$ times the logarithm of the time complexity divided by D, i.e., $O(n \log(f + \frac{n\delta}{D}))$. However, there is a bad scenario in which the complexity is $\Omega(nf)$. Indeed, in this scenario, the number of acknowledgments sending (and forwarding) events until the message is delivered at processor n (at time that is $O(fD)$) is just $O(n \log f)$. However, additional acknowledgments are sent *after* the message is delivered.

Put differently, what we did prove (regarding time complexity) is that if there are f faults, then the message must be delivered at node n at time that is $O(fD)$ (if δ is small). However, we did not prove that the acknowledgment from n to 1 is delivered in such a time. In fact, for the implementation of the previous subsection, one can show a case where for $f = O(\log n)$ faults the time for delivering n's acknowledgment in that implementation is $\Omega(2^f D)$. The way the time complexity is defined (only until the delivery of the message, or the disconnection of a link), we do not care about the time it takes the acknowledgment to arrive after the message is delivered. Still, this increases the message complexity of the previous implementation, since its message complexity is, actually, in the order of n multiplied by the logarithm of the time until all protocol-related communication ceases. The improvement of the message complexity in this section is obtained by shortening that ceasing time.

Let us demonstrate a bad scenario. Assume that processor $n-1$ is faulty; more specifically, assume that a message over any link of processor $n-1$ is delivered exactly after D time (rather than after δ time). Processor $n-2$ expects (and receives) an Ack from n within $4D$ time after $n-2$ forwarded the message to $n-1$. By that time, processor $n-2$ already sent a length 2 interval acknowledgment to $n-4$. Thus, the next acknowledgment that $n-4$ is waiting for is a length 4 interval acknowledgment, expected to arrive after a time that is double that of the length 2 acknowledgment.

Carrying this argument further, when a length 2^i interval acknowledgment Ack_i arrives at its destination, j, the next acknowledgment expected by processor $j-1$ is a length 2^{i+1} interval acknowledgment Ack_{i+1}, that is expected in double the time. Even if both Ack_i and Ack_{i+1} arrive at j at the same time, processor j can delay Ack_{i+1} without $j-1$ noticing a fault and disconnecting the link. Let f_j be the number of faults in the interval $[j, n]$. The state of knowledge of $j-1$ at this point is the same as in the case that the number of faults is 2^{f_j}. Thus, the length $n-1$ interval acknowledgment from n to 1 (i.e., the end-to-end acknowledgment) can also be delayed 2^{f_j} without causing $j-1$ to detect the fault.

Let us now describe the improvement to the previous implementation. The idea is to send (a few) other acknowledgments for "long" intervals. Acknowledgments sent quickly over "long" intervals which do not contain faults will reach every processor in the interval quickly. Therefore, many "short" acknowledgments whose intervals are contained in the "long" interval will become redundant, and therefore, these "short" acknowledgments will not be sent.

A natural selection is the set of intervals whose length is $\sqrt{n-1}$. Let us comment that using *only* this set, without using the shorter acknowledgments, we could obtain $O(n\delta + f\sqrt{n}D)$ time with $O(n)$ communication, which is a communication optimal but time suboptimal solution. However, to achieve time-optimality, we do combine this set with the "short" intervals.

Furthermore, as we now explain, we need also to send "quickly" acknowledgments over "long" intervals, i.e., intervals whose length is *more* than $\sqrt{n-1}$. By sending acknowledgments over the intervals of length $\sqrt{n-1}$ immediately upon forwarding the message, we prevent scenarios as described above for intervals whose length is less than $\sqrt{n-1}$, except for the few intervals of length $\sqrt{n-1}$ which contain faulty processors. However, we still have to deal with the intervals whose length is more than $\sqrt{n-1}$.

Recall that the "short acks," as defined in (5), are sent in order of increasing length. Namely, the *Ack* of interval of length l is sent after lD. The "long acks" are acknowledgments sent in the *reverse order*, from the longest to the shortest. Namely, the "long ack" of interval of length $\frac{n-1}{l}$ is sent after lD. (The length of the intervals divide $n-1$.) The reason the "long" intervals are sent in this order is similar to the idea behind the intervals of length $\sqrt{n-1}$: a single successful longer interval may make many *relatively* shorter intervals redundant. Note that the intervals which we want to become redundant as a result of the "long" intervals are not really short; their lengths are *more* than $\sqrt{n-1}$.

Formally, the implementation of this subsection is presented in (6) below. For simplicity, we assume that $n-1$ is an even power of 2. For $1 < i \leq n$ and $0 \leq t < n-1$ define $A_i^{(3)}(t)$ as follows:

$$
(6) \qquad A_i^{(3)}(t) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } i = n, \\ i - \sqrt{n-1} & \text{if } (\exists r \in \mathbb{N}) i = r\sqrt{n-1} + 1, \\ i - \sqrt{n-1} & \text{if } (\exists r \in \mathbb{N}) i = (r + \frac{1}{2})\sqrt{n-1} + 1, \\ i - \frac{n-1}{t} & \text{if } (\exists k) t = 2^k \wedge (\exists r \in \mathbb{N}) i = r\frac{n-1}{t} + 1, \\ i - \frac{n-1}{t} & \text{if } (\exists k) t = 2^k \wedge (\exists r \in \mathbb{N}) i = (r + \frac{1}{2})\frac{n-1}{t} + 1, \\ i - t & \text{if } (\exists k) t = 2^k \wedge (\exists r \in \mathbb{N}) i = rt + 1, \\ i - t & \text{if } (\exists k) t = 2^k \wedge (\exists r \in \mathbb{N}) i = (r + \frac{1}{2})t + 1, \\ \perp & \text{otherwise.} \end{cases}
$$

Note that the first, sixth, and seventh lines correspond to acknowledgments that are sent also by the previous implementation. We term the acknowledgments defined in the sixth and seventh lines "short" acknowledgments. The new acknowledgments defined in the fourth and fifth lines are termed "long" acknowledgments. The new acknowledgments defined in the second and third lines are termed "medium" acknowledgments.

Intuitively, this helps since "long" intervals which contain no faulty processors are "almost unaffected" by the faults "outside." For example, if all the processors from 1 till $(\sqrt{n-1}+1)$ are nonfaulty, they send only $O(n \log(\frac{n\delta}{D}))$ messages.

6.6. Complexities of the adaptive implementations. We begin by showing that the time complexity has not deteriorated significantly.

LEMMA 13. *The implementations $A^{(2)}()$ and $A^{(3)}()$ are early-terminating.*

Proof. We state the proof for $A^{(3)}()$, but all of our arguments hold for $A^{(2)}()$ as well. From Corollary 9, it is sufficient to bound $F(A^{(3)}())$ by a constant. We use the similarity between $A^{(3)}()$ and $A^{(1)}()$.

From Lemma 10 we know that $F(A^{(1)}())$ is bounded by a constant. For every j, l such that $1 \leq j < l \leq n$, we show that $F_{[j,l]}(A^{(3)}()) \leq \frac{1}{3} + F_{[j,l]}(A^{(1)}())$. Let j', l' be such that $j' \leq j < l \leq l'$ and $A^{(1)}_{l'}(0) = j'$ and $[j', l']$ is the "best cover" of $[j, l]$ in $A^{(3)}()$, namely,

$$F_{[j,l]}(A^{(1)}()) = \frac{l' - j}{l - j}.$$

By the definition of $A^{(3)}()$ there is some $t' \leq (l - j)$, such that $A^{(3)}_{l'}(t') = j'$. Hence,

$$\begin{aligned} F_{[j,l]}(A^{(3)}()) &\leq \frac{3(l' - j) + t'}{3(l - j)} \\ &\leq \frac{3(l' - j) + (l - j)}{3(l - j)} \\ &\leq \frac{1}{3} + \frac{3(l' - j)}{3(l - j)} \\ &\leq \frac{1}{3} + F_{[j,l]}(A^{(1)}()), \end{aligned}$$

which completes the proof. □

We now turn to the proof of the communication complexity. We begin by showing a simple necessary condition for sending a particular "short acknowledgment."

LEMMA 14. *Let j, i be processors such that $0 < i - j < \frac{\sqrt{n-1}}{2}$. If $\texttt{Send}_i^{j \leftarrow i}[Ack]$ occurs, then either there is a faulty processor in $[i, i + \sqrt{n-1}\,]$ or in $i - j \leq \sqrt{n-1} \cdot \frac{2\delta}{D}$.*

Proof. Since $i - j < \frac{\sqrt{n-1}}{2}$ and from (6), there is some processor l such that $A^{(3)}_l(0) \neq \perp$ and $l - A^{(3)}_l(0) = \sqrt{n-1}$ and $A^{(3)}_l(0) \leq j < i \leq l$. If there is a faulty processor in $[i, l]$, then the claim holds. Assume, therefore, that there is no faulty processor in $[i, l]$.

Since every nonfaulty processor forwards the message and the acknowledgments immediately, then $\texttt{Receive}_i^{A^{(3)}_l(0) \leftarrow l}[Ack]$ occurs not later than $2\sqrt{n-1} \cdot \delta$ after processor i entered the protocol. From the second condition of ⟨F3⟩, processor i does not issue the acknowledgment to j after $\texttt{Receive}_i^{A^{(3)}_l(0) \leftarrow l}[Ack]$. Hence, $\texttt{Send}_i^{j \leftarrow i}[Ack]$ may occur only before $\texttt{Receive}_i^{A^{(3)}_l(0) \leftarrow l}[Ack]$ occurs, i.e., before $2\sqrt{n-1} \cdot \delta$ since processor i started.

On the other hand, from (6) and since $i - j < \frac{\sqrt{n-1}}{2}$, then $\texttt{Send}_i^{j \leftarrow i}[Ack]$ occurs only when $time_i > i - j$. From ⟨F2⟩ and Axiom 2 holds $i - j < time_i$ only after at least $(i - j) \cdot D$ since processor i started. Hence, $\texttt{Send}_i^{j \leftarrow i}[Ack]$ occurs only if $(i - j) \cdot D \leq 2 \cdot \sqrt{n-1} \cdot \delta$. □

Lemma 14 shows that a "short acknowledgment" is issued only if it is one of the very short ones which are required since δ is not negligible, or if it is "close" to a faulty processor. We now bound the maximal number of "short" acknowledgments issued due to "close" faulty processors.

LEMMA 15. *For every k such that $\sqrt{n-1} \cdot \frac{2\delta}{D} < 2^k < \frac{\sqrt{n-1}}{2}$, there are at most $f \cdot 2 \cdot \frac{\sqrt{n-1}+1}{2^k}$ events of type $\texttt{Send}_i^{(i-2^k) \leftarrow i}[Ack]$.*

Proof. From Lemma 14, if $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ occurs as specified, then there is a faulty processor in $[i, i+\sqrt{n-1}\,]$. From (6), for some integer r either $i = r2^k + 1$ or $i = (r+\frac{1}{2})2^k + 1$ holds. Hence, for every faulty processor l, there are at most $2 \cdot \frac{\sqrt{n-1}+1}{2^k}$ intervals $[i-2^k, i]$ such that $l \in [i, i+\sqrt{n-1}\,]$. ∎

Lemma 15 bounds the communication due to acknowledgment intervals shorter than $\frac{\sqrt{n-1}}{2}$. In order to bound the entire communication complexity, we have to consider also longer acknowledgment intervals. We begin by bounding the maximal number of intervals either containing, or "near to," faulty processors.

LEMMA 16. *For every $k \in \mathbb{N}$, there are at most $5 \cdot f$ processors i such that for some t holds $A_i^{(3)}(t) = i - 2^k$ and the interval $[i-2^k, i+2^k]$ contains a faulty processor.*

Proof. From (6), and since we assumed that $\sqrt{n-1}$ is a power of 2, it follows that if $A_i^{(3)}(t) = i - 2^k$, then $(\exists r \in \mathbb{N})(i = r \cdot 2^k + 1) \vee (i = (r+\frac{1}{2}) \cdot 2^k + 1)$. Hence, for any faulty processor l, there are at most five processors i such that $A_i^{(3)}(t) = i - 2^k$ and $l \in [i-2^k,\, i+2^k]$. ∎

All that remains is to bound the communication in "long" intervals that do *not* contain, and are not near, a faulty processor.

LEMMA 17. *Consider k such that $\sqrt{n-1} < 2^k$. If $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ occurs, then either there is a faulty processor in $[i, i+2^k]$ or in $2^k > \sqrt{\frac{D\cdot(n-1)}{8\cdot\delta}}$.*

Proof. Assume that $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ occurs. Since $\sqrt{n-1} < 2^k$, from subsection 6.4 we know that processor $i+2^k$ is to send an acknowledgment over an interval of length 2^{k+1} (if $2^{k+1} \le n-1$). That is,

$$A_{i+2^k}^{(3)}\left(\frac{n-1}{2^{k+1}}\right) = (i+2^k) - 2^{k+1} = i - 2^k.$$

Assume that there is no faulty processor in $[i, i+2^k]$. Every nonfaulty processor forwards the message immediately. Hence, processor $i+2^k$ receives the message and starts executing at most $2^k \cdot \delta$ after processor i started executing. After at most $\frac{n-1}{2^{k+1}}\, TICK_{i+2^k}$ events, processor $i+2^k$ either sends its own length 2^{k+1} interval acknowledgment, or this acknowledgment is already redundant since it already performed some other $\texttt{Send}_{i+2^k}^{j\leftarrow l}[Ack]$ such that $j \le i-2^k < i < i+2^k \le l$. From Axiom 2, this occurs after at most $(\frac{n-1}{2^{k+1}}+1) \cdot D$ since processor $i+2^k$ started. Since every processor in $[i, i+2^k]$ is nonfaulty, it follows that $\texttt{Receive}_i^{j'\leftarrow l'}[Ack]$ occurs at most $2^k \cdot \delta$ after $\texttt{Send}_{i+2^k}^{j\leftarrow l}[Ack]$ with $j' \le j < l \le l'$. Namely, $\texttt{Receive}_i^{j'\leftarrow l'}[Ack]$ occurs after at most $2 \cdot 2^k \cdot \delta + (\frac{n-1}{2^{k+1}}+1) \cdot D$ since processor i started.

From (6) and since $\sqrt{n-1} < 2^k$, it follows that $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ occurs only after $time_i \ge \frac{n-1}{2^k}$. From Axiom 2, this occurs at least $\frac{n-1}{2^k} \cdot D$ since processor i starts executing. However, $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ does not happen after $\texttt{Receive}_i^{j'\leftarrow l'}[Ack]$ where $j' \le i-2^k < i < l'$. Since we assumed that $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ does occur, it follows that

$$\frac{n-1}{2^k} \cdot D < 2 \cdot 2^k \cdot \delta + \left(\frac{n-1}{2^{k+1}} + 1\right) \cdot D$$

from which the claim follows. ∎

We now use Lemmas 14–17 to compute the communication complexity.

THEOREM 18. *The communication complexity of implementation* $A^{(3)}()$ *is* $O(n\log(f+\frac{n\delta}{D}))$.

Proof. It is immediate that the communication complexity due to the data message and to the disconnection messages is $O(n)$. Therefore, we consider only acknowledgments.

Let n_k be the number of $\texttt{Send}_i^{(i-2^k)\leftarrow i}[Ack]$ events during the execution. The communication complexity due to acknowledgments is at most

$$C_{Ack} \leq \sum_{k=0}^{\log(n-1)} n_k \cdot 2^k \tag{7}$$

directly from (6) and since $\sqrt{n-1}$ is a power of 2, $n_k \leq \frac{n-1}{2^k}\cdot 2$.

By substituting this bound for n_k in (7), we obtain

$$C_{Ack} = O(n\log(n)). \tag{8}$$

The rest of the proof is needed to refine this bound. Let us first outline the proof. Lemma 15 gives tighter bounds of n_k for $\frac{\sqrt{n-1}\cdot 2\delta}{D} < 2^k < \frac{\sqrt{n-1}}{2}$. Lemmas 16 and 17 give tighter bounds of n_k for $\sqrt{n-1} < 2^k \leq \sqrt{\frac{D\cdot(n-1)}{8\cdot\delta}}$. We combine these tighter bounds with the simple bound of $\frac{(n-1)\cdot 2}{2^k}$ for other values of k and obtain the desired bound on the communication complexity.

Directly, since $n_k \leq \frac{n-1}{2^k}\cdot 2$, i.e., $n_k \cdot 2^k \leq (n-1)\cdot 2$, it follows that

$$\sum_{k=0}^{\lceil\log\frac{\sqrt{n-1}\cdot 2\delta}{D}\rceil} n_k\cdot 2^k \leq \sum_{k=0}^{\lceil\log\frac{\sqrt{n-1}\cdot 2\delta}{D}\rceil} 2(n-1) = O\left(n\log\frac{n\delta}{D}\right), \tag{9}$$

$$\sum_{k=(\log\sqrt{n-1})-1}^{\log\sqrt{n-1}} n_k\cdot 2^k \leq \sum_{k=(\log\sqrt{n-1})-1}^{\log\sqrt{n-1}} 2(n-1) = O(n), \tag{10}$$

$$\begin{aligned}\sum_{k=\lceil\log\sqrt{\frac{D(n-1)}{8\delta}}\rceil}^{\log(n-1)} n_k\cdot 2^k &\leq \sum_{k=\lceil\log\sqrt{\frac{D(n-1)}{8\delta}}\rceil}^{\log(n-1)} 2(n-1)\\ &\leq 2(n-1)\cdot\log\sqrt{\frac{8\delta(n-1)}{D}} = O\left(n\log\frac{n\delta}{D}\right),\end{aligned} \tag{11}$$

$$\begin{aligned}\sum_{k=\lceil\log\frac{2(n-1)}{5f}\rceil}^{\log(n-1)} n_k\cdot 2^k &\leq \sum_{k=\lceil\log\frac{2(n-1)}{5f}\rceil}^{\log(n-1)} 2(n-1)\\ &\leq 2(n-1)\cdot\log\frac{5f}{2} = O(n\log(f)).\end{aligned} \tag{12}$$

From Lemmas 16 and 17, if $\sqrt{n-1} < 2^k \leq \sqrt{\frac{D\cdot(n-1)}{8\cdot\delta}}$, then $n_k \leq 5\cdot f$. Therefore, the following holds:

$$\sum_{k=1+\log\sqrt{n-1}}^{\lceil\log\sqrt{\frac{D\cdot(n-1)}{8\delta}}\rceil-1} n_k\cdot 2^k \leq \sum_{k=1+\log\sqrt{n-1}}^{\lceil\log\frac{2(n-1)}{5f}\rceil} 5f\cdot 2^k + \sum_{k\lceil\log\frac{2(n-1)}{5f}\rceil}^{\log(n-1)} n_k\cdot 2^k. \tag{13}$$

Obviously,

$$\sum_{k=1+\log\sqrt{n-1}}^{\lceil \log \frac{2(n-1)}{5f} \rceil} 5f \cdot 2^k \leq 5f \cdot \frac{2(n-1)}{5f} \cdot 2 = O(n). \tag{14}$$

From inequalities (12), (13), and (14), we obtain

$$\sum_{k=1+\log\sqrt{n-1}}^{\lceil \log \sqrt{\frac{D\cdot(n-1)}{8\delta}} \rceil - 1} n_k \cdot 2^k \leq O(n) + O(n\log(f)) = O(n\log(f)). \tag{15}$$

From Lemma 15, it follows that if $\frac{\sqrt{n-1}\cdot 2\delta}{D} < 2^k < \frac{\sqrt{n-1}}{2}$, then $n_k \leq 2f \cdot \frac{\sqrt{n-1}+1}{2^k}$. Hence,

$$\begin{aligned} \sum_{k=1+\lceil \log \frac{\sqrt{n-1}\cdot 2\delta}{D} \rceil}^{(\log\sqrt{n-1})-2} n_k \cdot 2^k &\leq \sum_{k=1+\lceil \log \frac{\sqrt{n-1}\cdot 2\delta}{D} \rceil}^{(\log\sqrt{n-1})-2} 2f \cdot \frac{\sqrt{n-1}+1}{2^k} \cdot 2^k \\ &= O(\sqrt{n} \cdot f \cdot \log(n)). \end{aligned} \tag{16}$$

From inequalities (7), (9), (10), (11), (15), and (16), we obtain

$$C_{Ack} \leq \sum_{k=0}^{\log(n-1)} n_k \cdot 2^k = O\left(n \log \frac{n\delta}{D}\right) + O(n\log(f)) + O(\sqrt{n} \cdot f \cdot \log(n)). \tag{17}$$

For $f \geq n^{\frac{1}{4}}$ the claim follows from (8), since $C_{Ack} = O(n \log n)$, and in this case $O(n\log(f)) = O(n\log(n))$. For the case that $f < n^{\frac{1}{4}}$ the claim follows from (17) since then $O(\sqrt{n} \cdot f \cdot \log(n)) \leq O(n)$. □

7. Conclusions. We have observed that the actual delivery time in asynchronous bounded networks is much shorter than the known bound on the delivery time. We introduced a way to model and take advantage of that fact and the notion of early termination for protocols in the asynchronous bounded network. Following [AADW94], we observed that early termination is a form of distributed competitiveness.

The protocols presented ensure early-terminating detection of arbitrary failures in forwarding a message along a fixed route. The protocols are quite simple and need only finite memory. The penalty in message complexity is acceptable for most applications. The message complexity is at most $O(n \log n)$, where n is the length of the path, compared to $O(n)$ for the trivial protocol which cannot overcome faults other than those detectable by the link protocol. The message complexity of our adaptive detector is $O(n \log(f + \frac{n\delta}{D}))$, where $\frac{\delta}{D}$ is the ratio between the actual delay and the a priori bound on the delay, and f is the number of faults. Since usually $\frac{\delta}{D} \ll 1$ and $f = 0$ (or $f = O(1)$), the communication complexity is nearly optimal. Theorems 5 and 7 may be used to achieve other trade-offs by different implementations of the design, some of which may be better in practical applications. For example, it is easy to keep the communication complexity optimal (i.e., $O(n)$) while still improving the time complexity from the $O(nD)$ of the trivial protocol to $(O(n\delta + \sqrt{n} \cdot f \cdot D))$.

Further work is needed to find the best communication complexity for early-terminating protocols, possibly generalizing our lower bound to hold for a general

protocol. Let us list some additional open problems: generalizing our results to networks and rings (rather than a path), considering probabilistic protocols, dealing with clock drifts, and efficient handling of many messages. Additional further work is needed in order to understand the implications of the model for other tasks and, possibly, to generalize the model further. As mentioned in the introduction, some of this further research has meanwhile already taken place.

Acknowledgments. Special thanks to Oded Goldreich and Adrian Segall, who supervised the work of the first author, for their encouragement and help in obtaining the results and improving the exposition.

It is a pleasure to thank Hagit Attiya, Baruch Awerbuch, Gil Barzilai, Tsipi Barzilai, Inder Gopal, Madan Gopal, George Grover, Radia Perlman, Ken Perry, and Moti Yung for helpful discussions and motivations for this work.

REFERENCES

[AAG87] Y. Afek, B. Awerbuch, and E. Gafni, *Applying static network protocols to dynamic networks*, in Proceedings of the 28th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 358–370.

[AAPS87] Y. Afek, B. Awerbuch, S. A. Plotkin, and M. Saks, *Local management of a global resource in a communication network*, J. ACM, 43 (1996), pp. 1–19.

[AKP92] B. Awerbuch, S. Kutten, and D. Peleg, *Competitive distributed job scheduling*, in Proceedings of the 24th ACM Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 571–580.

[AL89] H. Attiya and N. Lynch, *Time bounds for real-time process control in presence of timing uncertainty*, Inform. and Comput., 110 (1994), pp. 183–232.

[ADLS91] H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer, *Bounds on the time to reach agreement in the presence of timing uncertainty*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, New Orleans, LA, 1991, pp. 359–369.

[AE86] B. Awerbuch and S. Even, *Reliable broadcast protocols in unreliable networks*, Networks, 16 (1986), pp. 381–396.

[AG88] Y. Afek and E. Gafni, *End-to-end communication in unreliable networks*, in Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing, Toronto, Ontario, Canada, 1988, pp. 131–148.

[AGH90] B. Awerbuch, O. Goldreich, and A. Herzberg, *A quantitative approach to dynamic networks*, in Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing, Quebec City, Quebec, Canada, 1990, pp. 189–204.

[AAGMRS97] Y. Afek, B. Awerbuch, E. Gafni, E. Rosen, and N. Shavit, *Slide—the key to polynomial end-to-end communication*, J. Algorithms, 22 (1997), pp. 158–186.

[AS87] Y. Afek and M. Saks, *Detecting global termination conditions in the face of uncertainty*, in Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, Canada, 1987, pp. 109–124.

[BGJ+85] A. E. Baratz, J. P. Gray, P. E. Green, Jr., J. M. Jaffe, and D. P. Pozefsky, *SNA networks of small systems*, IEEE J. Selected Areas in Comm., 3 (1985), pp. 416–426.

[BS88] A. E. Baratz and A. Segall, *Reliable link initialization procedures*, IEEE Trans. Comm., 36 (1988), pp. 144–152.

[BYKWZ87] R. Bar-Yehuda, S. Kutten, Y. Wolfstahl, and S. Zaks, *Making distributed spanning tree algorithms fault resilient*, in Proceedings of the Fourth Symposium on Theoretical Aspects of Comptuer Science, Passau, Germany, Lecture Notes in Comput. Sci. 247, Springer-Verlag, Berlin, 1987, pp. 432–444.

[CCGZ88] C. T. Chou, I. Cidon, I. Gopal, and S. Zaks, *Synchronizing asynchronous bounded delay networks*, in Distributed Algorithms: Second International Workshop, J. van Leeuwen, ed., Lecture Notes in Comput. Sci. 312, Springer-Verlag, New York, 1988, pp. 212–218.

[CR87] I. Cidon and R. Rom, *Failsafe end-to-end protocols in computer networks with changing topology*, IEEE Trans. Comm., 35 (1987), pp. 410–413.

[DHSS84] D. Dolev, J. Halpern, B. Simons, and R. Strong, *Dynamic fault-tolerant clock synchronization*, J. ACM, 42 (1995), pp. 143–185.

[DRS86] D. Dolev, R. Reischuk, and H. R. Strong, *Early stopping in Byzantine agreement*, J. ACM, 37 (1990), pp. 720–741.

[AADW94] M. Ajtai, J. Aspnes, C. Dwork, and O. Waarts, *A theory of competitive analysis for distributed algorithms*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 401–411.

[Fin79] S. G. Finn, *Resynch procedures and failsafe network protocol*, IEEE Trans. Comm., 27 (1979), pp. 840–846.

[GHM89] O. Goldreich, A. Herzberg, and Y. Mansour, *Source to destination communication in the presence of faults*, in Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Canada, 1989, pp. 85–101.

[Gro82] G. Grover, *High Availability for Networks (HAPN)—Non-Disruptive VR Switching*, internal memorandum, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1982.

[GSK87] P. M. Gopal, R. A. Sultan, and B. K. Kadaba, *Performance Limits of APPN Architecture*, internal memorandum, IBM T. J. Watson Research Center, Yorktown Heights, NY, 1987.

[Her88] A. Herzberg, *Network management in the presence of faults*, in Proceedings of the Ninth International Conference on Computers and Communication, Tel Aviv, Israel, 1988.

[HK89] A. Herzberg and S. Kutten, *Efficient detection of message forwarding faults*, in Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Canada, 1989, pp. 339–353.

[K88] S. Kutten, *Optimal fault tolerant distributed construction of a spanning forest*, Inform. Process. Lett., 27 (1988), pp. 299–307.

[MRR80] J. M. McQuillan, I. Richer, and E. C. Rosen, *The new routing algorithm for the ARPANET*, IEEE Trans. Comm., 28 (1980), pp. 711–719.

[HM90] J. Y. Halpern and Y. Moses, *Knowledge and common knowledge in a distributed environment*, J. ACM, 37 (1990), pp. 549–587.

[Per88] R. Perlman, *Network Layer Protocols with Byzantine Robustness*, Ph.D. thesis, MIT Laboratory for Computer Science, Cambridge, MA, 1988.

[Pon91] S. Ponzio, *Network consensus in the presence of timing uncertainty: Omission and Byzantine failures*, in Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, 1991, pp. 125–138.

[SJ86] A. Segall and J. M. Jaffe, *Route setup with local identifiers*, IEEE Trans. Comm., 34 (1986), pp. 45–53.

[Sta87] W. Stallings, *Handbook of Computer Communication Standards*, Vol. 1, Macmillan, New York, 1987.

[Tan81] A. Tannenbaum, *Computer Networks*, Prentice–Hall, Englewood Cliffs, NJ, 1981.

[Zim80] H. Zimmerman, *OSI reference model—the ISO model of architecture for open systems interconnection*, IEEE Trans. Comm., 28 (1980), pp. 425–432.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1197–1210

THE DENSITY OF WEAKLY COMPLETE PROBLEMS UNDER ADAPTIVE REDUCTIONS*

JACK H. LUTZ† AND YONG ZHAO‡

Abstract. Given a real number $\alpha < 1$, every language that is weakly $\leq^{\mathrm{P}}_{n^{\alpha/2}-\mathrm{T}}$-hard for E or weakly $\leq^{\mathrm{P}}_{n^{\alpha}-\mathrm{T}}$-hard for E_2 is shown to be exponentially dense. This simultaneously strengthens the results of Lutz and Mayordomo (1994) and Fu (1995).

Key words. complexity classes, computational complexity, resource-bounded measure, polynomial reductions, weakly complete problems

AMS subject classification. 68Q15

PII. S0097539797321547

1. Introduction.

In the mid-1970s, Meyer [16] proved that every $\leq^{\mathrm{P}}_{\mathrm{m}}$-complete language for exponential time—in fact, every $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard language for exponential time—is dense. That is,

$$\mathrm{E} \not\subseteq \mathrm{P}_{\mathrm{m}}(\mathrm{DENSE}^c), \tag{1.1}$$

where $\mathrm{E} = \mathrm{DTIME}(2^{\mathrm{linear}})$, DENSE is the class of all dense languages, DENSE^c is the complement of DENSE, and $\mathrm{P}_{\mathrm{m}}(\mathrm{DENSE}^c)$ is the class of all languages that are $\leq^{\mathrm{P}}_{\mathrm{m}}$-reducible to nondense languages. (A language $A \in \{0,1\}^*$ is *dense* if there is a real number $\epsilon > 0$ such that $|A_{\leq n}| > 2^{n^{\epsilon}}$ for all sufficiently large n, where $A_{\leq n} = A \cap \{0,1\}^{\leq n}$.) Since that time, a major objective of computational complexity theory has been to extend Meyer's result from $\leq^{\mathrm{P}}_{\mathrm{m}}$-reductions to $\leq^{\mathrm{P}}_{\mathrm{T}}$-reductions, i.e., to prove that every $\leq^{\mathrm{P}}_{\mathrm{T}}$-hard language for E is dense. That is, the objective is to prove that

$$\mathrm{E} \not\subseteq \mathrm{P}_{\mathrm{T}}(\mathrm{DENSE}^c), \tag{1.2}$$

where $\mathrm{P}_{\mathrm{T}}(\mathrm{DENSE}^c)$ is the class of all languages that are $\leq^{\mathrm{P}}_{\mathrm{T}}$-reducible to nondense languages. The importance of this objective derives largely from the fact (noted by Meyer [16]) that the class $\mathrm{P}_{\mathrm{T}}(\mathrm{DENSE}^c)$ contains all languages that have subexponential circuit-size complexity. (A language $A \subseteq \{0,1\}^*$ has *subexponential circuit-size complexity* if, for every real number $\epsilon > 0$ and for every sufficiently large n, there is an n-input, 1-output Boolean circuit that decides that the set $A_{=n} = A \cap \{0,1\}^n$ and has fewer than $2^{n^{\epsilon}}$ gates. Otherwise, we say that A has *exponential circuit-size complexity*.) Thus a proof of (1.2) would tell us that E contains languages with exponential circuit-size complexity, thereby answering a major open question concerning the relationship between (uniform) time complexity and (nonuniform) circuit-size complexity. Of course (1.2) also implies the more modest, but more famous, conjecture that

$$\mathrm{E} \not\subseteq \mathrm{P}_{\mathrm{T}}(\mathrm{SPARSE}), \tag{1.3}$$

*Received by the editors February 11, 1997; accepted for publication (in revised form) April 12, 1998; published electronically October 6, 2000. This research was supported in part by National Science Foundation grants CCR-9157382 (with matching funds from Rockwell International, Microware Systems Corporation, and Amoco Foundation) and CCR-9610461.

http://www.siam.org/journals/sicomp/30-4/32154.html

†Department of Computer Science, Iowa State University, Ames, IA 50011 (lutz@cs.iastate.edu).

‡Nationwide Insurance, One Nationwide Plaza, Columbus, OH 43215 (zhaoy@nationwide.com).

where SPARSE is the class of all sparse languages. (A language $A \subseteq \{0,1\}^*$ is *sparse* if there is a polynomial $q(n)$ such that $|A_{\leq n}| \leq q(n)$ for all $n \in \mathbb{N}$.) As noted by Meyer [16], the class $\mathrm{P_T}(\mathrm{SPARSE})$ consists precisely of all languages that have polynomial circuit-size complexity, so (1.3) asserts that E contains languages that do not have polynomial circuit-size complexity.

Knowing (1.1) and wanting to prove (1.2), the natural strategy has been to prove results of the form

$$\mathrm{E} \not\subseteq \mathrm{P}_r(\mathrm{DENSE}^c)$$

for successively larger classes $\mathrm{P}_r(\mathrm{DENSE}^c)$ in the range

$$\mathrm{P_m}(\mathrm{DENSE}^c) \subseteq \mathrm{P}_r(\mathrm{DENSE}^c) \subseteq \mathrm{P_T}(\mathrm{DENSE}^c).$$

The first major step beyond (1.1) in this program was the proof by Watanabe [18] that

$$\mathrm{E} \not\subseteq \mathrm{P}_{O(\log n)-\mathrm{tt}}(\mathrm{DENSE}^c), \tag{1.4}$$

i.e., that every language that is $\leq^{\mathrm{P}}_{O(\log n)-\mathrm{tt}}$-hard for E is dense. The next big step was the proof by Lutz and Mayordomo [11] that, for every real number $\alpha < 1$,

$$\mathrm{E} \not\subseteq \mathrm{P}_{n^\alpha-\mathrm{tt}}(\mathrm{DENSE}^c). \tag{1.5}$$

This improved Watanabe's result from $O(\log n)$ truth-table (i.e., nonadaptive) queries to n^α such queries for α arbitrarily close to 1 (e.g., to $n^{0.99}$ truth-table queries). Moreover, Lutz and Mayordomo [11] proved (1.5) by first proving the stronger result that for all $\alpha < 1$,

$$\mu_{\mathrm{p}}(\mathrm{P}_{n^\alpha-\mathrm{tt}}(\mathrm{DENSE}^c)) = 0, \tag{1.6}$$

which implies that every language that is weakly $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard for E or for $\mathrm{E_2} = \mathrm{DTIME}(2^{\mathrm{poly}})$ is dense. (A language A is *weakly* $\leq^{\mathrm{P}}_r$*-hard* for a complexity class $\mathcal{C}$ if $\mu(\mathrm{P}_r(A) \mid \mathcal{C}) \neq 0$, i.e., if $\mathrm{P}_r(A) \cap \mathcal{C}$ is a nonnegligible subset of $\mathcal{C}$ in the sense of the resource-bounded measure developed by Lutz [10]. A language A is *weakly* $\leq^{\mathrm{P}}_r$*-complete* for $\mathcal{C}$ if $A \in \mathcal{C}$ and A is weakly $\leq^{\mathrm{P}}_r$-hard for $\mathcal{C}$. See [13] or [2] for a survey of resource-bounded measure and weak completeness.) It is now known that the set of $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard languages for $\mathrm{E_2}$ has measure 0 in $\mathrm{E_2}$ [7], while the set of weakly $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard languages for $\mathrm{E_2}$ has measure 1 in $\mathrm{E_2}$ [3]. Thus almost every language in $\mathrm{E_2}$ is weakly $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard, but not $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard, for $\mathrm{E_2}$, so the result of Lutz and Mayordomo [11] for $\mathrm{E_2}$ is provably much more general than the fact that every $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard language for $\mathrm{E_2}$ is dense. We conjecture that this also holds for E.

A word on the relationship between hardness notions for E and $\mathrm{E_2}$ is in order here. It is well known that a language is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for E if and only if it is $\leq^{\mathrm{P}}_{\mathrm{m}}$-hard for $\mathrm{E_2}$; this is because $\mathrm{E_2} = \mathrm{P_m}(\mathrm{E})$. The same equivalence holds for $\leq^{\mathrm{P}}_{\mathrm{T}}$-hardness. It is also clear that every language that is $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard for $\mathrm{E_2}$ is $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard for E. However, it is not generally the case that $\mathrm{P_m}(\mathrm{P}_{n^\alpha-\mathrm{tt}}(A)) = \mathrm{P}_{n^\alpha-\mathrm{tt}}(A)$, so it may well be the case that a language can be $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard for E but not for $\mathrm{E_2}$. These same remarks apply to $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$-hardness.

The relationship between weak hardness notions for E and $\mathrm{E_2}$ is somewhat different. Juedes and Lutz [9] have shown that weak $\leq^{\mathrm{P}}_{\mathrm{m}}$-hardness for E implies weak

$\leq^{\mathrm{P}}_{\mathrm{m}}$-hardness for E_2, and their proof of this fact also works for weak $\leq^{\mathrm{P}}_{\mathrm{T}}$-hardness. However, Juedes and Lutz [9] also showed that weak $\leq^{\mathrm{P}}_{\mathrm{m}}$-hardness for E_2 does not generally imply weak $\leq^{\mathrm{P}}_{\mathrm{m}}$-hardness for E, and it is reasonable to conjecture (but has not been proven) that the same holds for weak $\leq^{\mathrm{P}}_{\mathrm{T}}$-hardness. We further conjecture that the notions of weak $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hardness for E and weak $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hardness E_2 are incomparable, and similarly for weak $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$-hardness. In any case, (1.6) implies that, for every $\alpha < 1$, every language that is weakly $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$-hard for either E or E_2 is dense.

Shortly after, but independently of [11], Fu [8] used very different techniques to prove that, for every $\alpha < 1$,

$$\mathrm{E} \not\subseteq \mathrm{P}_{n^{\alpha/2}-\mathrm{T}}(\mathrm{DENSE}^c) \tag{1.7}$$

and

$$\mathrm{E}_2 \not\subseteq \mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c). \tag{1.8}$$

That is, every language that is $\leq^{\mathrm{P}}_{n^{\alpha/2}-\mathrm{T}}$-hard for E or $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$-hard for E_2 is dense. These results do not have the measure-theoretic strength of (1.6), but they are a major improvement over previous results on the densities of hard languages in that they hold for Turing reductions, which have *adaptive* queries.

In the present paper, we prove results which simultaneously strengthen results of Lutz and Mayordomo [11] and the results of Fu [8]. Specifically, we prove that, for every $\alpha < 1$,

$$\mu_{\mathrm{p}}(\mathrm{P}_{n^{\alpha/2}-\mathrm{T}}(\mathrm{DENSE}^c)) = 0 \tag{1.9}$$

and

$$\mu_{\mathrm{p}_2}(\mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c)) = 0. \tag{1.10}$$

These results imply that every language that is weakly $\leq^{\mathrm{P}}_{n^{\alpha/2}-\mathrm{T}}$-hard for E or weakly $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$-hard for E_2 is dense. The proof of (1.9) and (1.10) is not a simple extension of the proof in [11] or the proof in [8], but rather combines ideas from both [11] and [8] with the martingale dilation technique introduced by Ambos-Spies, Terwijn, and Zheng [3].

Our results also show that the strong hypotheses $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ and $\mu_{\mathrm{p}_2}(\mathrm{NP}) \neq 0$ (surveyed in [13] and [2]) have consequences for the densities of adaptively hard languages for NP. Mahaney [14] proved that

$$\mathrm{P} \neq \mathrm{NP} \Rightarrow \mathrm{NP} \not\subseteq \mathrm{P}_{\mathrm{m}}(\mathrm{SPARSE}), \tag{1.11}$$

and Ogiwara and Watanabe [17] improved this to

$$\mathrm{P} \neq \mathrm{NP} \Rightarrow \mathrm{NP} \not\subseteq \mathrm{P}_{\mathrm{btt}}(\mathrm{SPARSE}). \tag{1.12}$$

That is, if $\mathrm{P} \neq \mathrm{NP}$, then no sparse language can be $\leq^{\mathrm{P}}_{\mathrm{btt}}$-hard for NP. Lutz and Mayordomo [11] used (1.6) to obtain a stronger conclusion from a stronger hypothesis; namely, for all $\alpha < 1$,

$$\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0 \Rightarrow \mathrm{NP} \not\subseteq \mathrm{P}_{n^\alpha-\mathrm{tt}}(\mathrm{DENSE}^c). \tag{1.13}$$

By (1.10) and the known fact that $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ iff $\mu_{\mathrm{p}_2}(\mathrm{NP}) \neq 0$ [3], we now have, for all $\alpha < 1$,

$$\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0 \Rightarrow \mathrm{NP} \not\subseteq \mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c). \tag{1.14}$$

Thus, if $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$, then every language that is $\leq^{\mathrm{P}}_{n^{0.99}-\mathrm{T}}$-hard for NP is dense.

2. Preliminaries. The *Boolean value* of a condition ψ is

$$[\![\psi]\!] = \begin{cases} 1 & \text{if } \psi, \\ 0 & \text{if not } \psi. \end{cases}$$

The *standard enumeration* of $\{0,1\}^*$ is $s_0 = \lambda, s_1 = 0, s_2 = 1, s_3 = 00, \ldots$. This enumeration induces a total ordering of $\{0,1\}^*$ which we denote by $<$.

All *languages* here are subsets of $\{0,1\}^*$. The *Cantor space* is the set $\mathbf{C}$ of all languages. We identify each language $A \in \mathbf{C}$ with its characteristic sequence, which is the infinite binary sequence

$$[\![s_0 \in A]\!][\![s_1 \in A]\!][\![s_2 \in A]\!] \cdots,$$

where $s_0 = \lambda, s_1 = 0, s_2 = 1, s_3 = 00, \ldots$ is the standard enumeration of $\{0,1\}^*$. For $w \in \{0,1\}^*$ and $A \in \mathbf{C}$, we write $w \sqsubseteq A$ to indicate that w is a prefix of (the characteristic sequence of) A. The *symmetric difference* of the two languages A and B is $A \triangle B = (A - B) \cup (B - A)$.

The *cylinder generated* by a string $w \in \{0,1\}^*$ is the set

$$\mathbf{C}_w = \{A \in \mathbf{C} | w \sqsubseteq A\}.$$

Note that $\mathbf{C}_\lambda = \mathbf{C}$.

In this paper, a set $X \subseteq \mathbf{C}$ that appears in a probability $\Pr(X)$ or a conditional probability $\Pr(X|\mathbf{C}_w)$ is regarded as an event in the sample space $\mathbf{C}$ with the uniform probability measure. Thus, for example, $\Pr(X)$ is the probability that $A \in X$ when the language $A \subseteq \{0,1\}^*$ is chosen probabilistically by using an independent toss of a fair coin to decide membership of each string in A. In particular, $\Pr(\mathbf{C}_w) = 2^{-|w|}$. The *complement* of a set $X \subseteq \mathbf{C}$ is the set $X^c = \mathbf{C} - X$.

Let $d \in \mathbb{N}$ and $t : \mathbb{N} \to \mathbb{N}$. A function $f : \mathbb{N}^d \times \{0,1\}^* \to \mathbb{Q}$ is *exactly $t(n)$-time-computable* if there is an algorithm that, on input $(k_1, \ldots, k_d, w) \in \mathbb{N}^d \times \{0,1\}^*$, runs for at most $O(t(k_1, \ldots, k_d, |w|))$ steps and outputs an ordered pair $(a, b) \in \mathbb{Z} \times \mathbb{Z}$ such that $f(k_1, \ldots, k_d, w) = \frac{a}{b}$. A function $f : \mathbb{N}^d \times \{0,1\}^* \to \mathrm{R}$ is *$t(n)$-time-computable* if there is an exactly $t(n)$-time-computable function $\widehat{f} : \mathbb{N}^{d+1} \times \{0,1\}^* \to \mathbb{Q}$ such that, for all $r, k_1, \ldots, k_d \in \mathbb{N}$ and $w \in \{0,1\}^*$,

$$|\widehat{f}(r, k_1, \ldots, k_d, w) - f(k_1, \ldots, k_d, w)| \leq 2^{-r}.$$

We briefly review those aspects of martingales and resource-bounded measure that are needed for our main theorem. The reader is referred to [2], [10], [13], or [15] for more thorough discussion.

A *martingale* is a function $d : \{0,1\}^* \to [0, \infty)$ such that, for all $w \in \{0,1\}^*$,

$$d(w) = \frac{d(w0) + d(w1)}{2}.$$

If $t : \mathbb{N} \to \mathbb{N}$, then a *$t(n)$-martingale* is a martingale that is $t(n)$-time-computable, and an *exact $t(n)$-martingale* is a (rational-valued) martingale that is exactly $t(n)$-time-computable. A martingale d *succeeds* on a language $A \in \mathbf{C}$ if, for every $c \in \mathbb{N}$, there exists $w \sqsubseteq A$ such that $d(w) > c$. The *success set* of a martingale d is the set

$$S^\infty[d] = \{A \in \mathbf{C} | d \text{ succeeds on } A\}.$$

The *unitary success set* of d is

$$S^1[d] = \bigcup_{\substack{w \in \{0,1\}^* \\ d(w) \geq 1}} \mathbf{C}_w.$$

The following result was proved by Juedes and Lutz [9] and independently by Mayordomo [15].

LEMMA 2.1 (exact computation lemma). *Let $t : \mathbb{N} \to \mathbb{N}$ be nondecreasing with $t(n) \geq n^2$. Then, for every $t(n)$-martingale d, there is an exact $n \cdot t(2n+2)$-martingale $\tilde{d}$ such that $S^\infty[d] \subseteq S^\infty[\tilde{d}]$.*

A sequence

$$\sum_{k=0}^{\infty} a_{j,k} \qquad (j = 0, 1, 2, \dots)$$

of series of terms $a_{j,k} \in [0, \infty)$ is *uniformly* p-*convergent* if there is a polynomial $m : \mathbb{N}^2 \to \mathbb{N}$ such that, for all $j, r \in \mathbb{N}$, $\sum_{k=m_j(r)}^{\infty} a_{j,k} \leq 2^{-r}$, where we write $m_j(r) = m(j, r)$. The following sufficient condition for uniform p-convergence is easily verified by routine calculus.

LEMMA 2.2. *Let $a_{j,k} \in [0, \infty)$ for all $j, k \in \mathbb{N}$. If there exist a real number $\epsilon > 0$ and a polynomial $g : \mathbb{N} \to \mathbb{N}$ such that $a_{j,k} \leq e^{-k^\epsilon}$ for all $j, k \in \mathbb{N}$ with $k \geq g(j)$, then the series $\sum_{k=0}^{\infty} a_{j,k}$ $(j = 0, 1, 2, \dots)$ are uniformly* p-*convergent.*

A uniform, resource-bounded generalization of the classical first Borel–Cantelli lemma was proved by Lutz [10]. Here we use the following precise variant of this result.

THEOREM 2.3. *Let $\alpha, \tilde{\alpha} \in \mathrm{R}$ with $1 \leq \alpha < \tilde{\alpha}$, and let*

$$d : \mathbb{N} \times \mathbb{N} \times \{0,1\}^* \to \mathbb{Q} \cap [0, \infty)$$

be an exactly $2^{(\log n)^\alpha}$-time-computable function with the following two properties.

(i) *For each $j, k \in \mathbb{N}$, the function $d_{j,k}$ defined by $d_{j,k}(w) = d(j, k, w)$ is a martingale.*

(ii) *The series $\sum_{k=0}^{\infty} d_{j,k}(\lambda)$ $(j = 0, 1, 2, \dots)$ are uniformly* p-*convergent.*

Then there is an exact $2^{(\log n)^{\tilde{\alpha}}}$-martingale $\tilde{d}$ such that

$$\bigcup_{j=0}^{\infty} \bigcap_{t=0}^{\infty} \bigcup_{k=t}^{\infty} S^1[d_{j,k}] \subseteq S^\infty[\tilde{d}].$$

Proof. Assume the hypothesis, and fix $\alpha' \in \mathbb{Q}$ such that $\alpha < \alpha' < \tilde{\alpha}$. Since $n \cdot 2^{(\log(2n+2))^{\alpha'}} = o(2^{(\log n)^{\tilde{\alpha}}})$, it suffices by Lemma 2.1 to show that there is a $2^{(\log n)^{\alpha'}}$-martingale d' such that

$$\bigcup_{j=0}^{\infty} \bigcap_{t=0}^{\infty} \bigcup_{k=t}^{\infty} S^1[d_{j,k}] \subseteq S^\infty[d']. \tag{2.1}$$

Fix a polynomial $m : \mathbb{N}^2 \to \mathbb{N}$ testifying that the series $\sum_{k=0}^{\infty} d_{j,k}(\lambda)$ $(j = 0, 1, 2, \dots)$ are uniformly p-convergent, and define

$$d'(w) = \sum_{j=0}^{\infty} \sum_{t=0}^{\infty} \sum_{k=m_j(2t)}^{\infty} 2^{t-j} d_{j,k}(w)$$

for all $w \in \{0,1\}^*$. Thus, for each $w \in \{0,1\}^*$,

$$\begin{aligned} d'(w) &\le \sum_{j=0}^{\infty}\sum_{t=0}^{\infty}\sum_{k=m_j(2t)}^{\infty} 2^{t-j+|w|}d_{j,k}(\lambda) \\ &\le 2^{|w|}\sum_{j=0}^{\infty}2^{-j}\sum_{t=0}^{\infty}2^t\cdot 2^{-2t} \\ &= 2^{|w|+2}, \end{aligned}$$

so $d' : \{0,1\}^* \to [0,\infty)$. It is clear by linearity that d' is a martingale. To see that (2.1) holds, assume that $A \in \cup_{j=0}^{\infty}\cap_{t=0}^{\infty}\cup_{k=t}^{\infty}S^1[d_{j,k}]$, and let $c \in \mathbb{N}$ be arbitrary. Then there exist $j \in \mathbb{N}$ and $k \ge m_j(2j+2c)$ such that $A \in S^1[d_{j,k}]$. Fix $w \sqsubseteq A$ such that $d_{j,k}(w) \ge 1$. Then $d'(w) \ge 2^{c+j-j}d_{j,k}(w) \ge 2^c$. Since c is arbitrary here, it follows that $A \in S^\infty[d']$, confirming (2.1).

To see that d' is $2^{(\log n)^{\alpha'}}$-time-computable, define $d_A, d_B, d_C : \mathbb{N}\times\{0,1\}^* \to [0,\infty)$ as follows, using the abbreviation $s = r + |w| + 2$.

$$\begin{aligned} d_A(r,w) &= \sum_{j=0}^{s}\sum_{t=0}^{\infty}\sum_{k=m_j(2t)}^{\infty} 2^{t-j}d_{j,k}(w), \\ d_B(r,w) &= \sum_{j=0}^{s}\sum_{t=0}^{2s}\sum_{k=m_j(2t)}^{\infty} 2^{t-j}d_{j,k}(w), \\ d_C(r,w) &= \sum_{j=0}^{s}\sum_{t=0}^{2s}\sum_{k=m_j(2t)}^{m_j(2s^2+4s+t)} 2^{t-j}d_{j,k}(w). \end{aligned} \tag{2.2}$$

For all $r \in \mathbb{N}$ and $w \in \{0,1\}^*$, it is clear that

$$d_C(r,w) \le d_B(r,w) \le d_A(r,w) \le d'(w),$$

and it is routine to verify the inequalities

$$\begin{aligned} d'(w) - d_A(r,w) &\le 2^{-(r+1)}, \\ d_A(r,w) - d_B(r,w) &\le 2^{-(r+2)}, \\ d_B(r,w) - d_C(r,w) &\le 2^{-(r+2)}, \end{aligned}$$

whence we have

$$d'(w) - 2^{-r} \le d_C(r,w) \le d'(w) \tag{2.3}$$

for all $r \in \mathbb{N}$ and $w \in \{0,1\}^*$. Using formula (2.2), the time required to compute $d_C(r,w)$ exactly is no greater than

$$O((s+1)(2s+1)m(s,2s^2+4s+2s)2^{(\log n)^\alpha}) = O(q(n)\cdot 2^{(\log n)^\alpha}),$$

where $n = r + |w|$ and q is a polynomial. Since $q(n)\cdot 2^{(\log n)^\alpha} = o(2^{(\log n)^{\alpha'}})$, it follows that $d_C(r,w)$ is exactly $2^{(\log n)^{\alpha'}}$-time-computable. By (2.3), then, d' is a $2^{(\log n)^{\alpha'}}$-martingale. □

The proof of our main theorem uses the techniques of weak stochasticity and martingale dilation, which we briefly review here.

As usual, an *advice function* is a function $h : \mathbb{N} \to \{0,1\}^*$. Given a function $q : \mathbb{N} \to \mathbb{N}$, we write $\mathrm{ADV}(q)$ for the set of all advice functions h such that $|h(n)| \leq q(n)$ for all $n \in \mathbb{N}$. Given a language B and an advice function h, we define the language

$$B/h = \{x \in \{0,1\}^* \mid \langle x, h(|x|)\rangle \in B\},$$

where $\langle \cdot\ ,\ \cdot \rangle$ is a standard string-pairing function, e.g., $\langle x, y\rangle = 0^{|x|}1xy$. Given functions $t, q : \mathbb{N} \to N$, we define the advice class

$$\mathrm{DTIME}(t)/\mathrm{ADV}(q) = \{B/h \mid B \in \mathrm{DTIME}(t) \text{ and } h \in \mathrm{ADV}(q)\}.$$

Definition (Lutz and Mayordomo [11], Lutz [12]). *For $t, q, \nu : \mathbb{N} \to \mathbb{N}$, a language A is* weakly (t, q, ν)-stochastic *if, for all $B, C \in \mathrm{DTIME}(t)/\mathrm{ADV}(q)$ such that $|C_{=n}| \geq \nu(n)$ for all sufficiently large n,*

$$\lim_{n\to\infty} \frac{|(A \triangle B) \bigcap C_{=n}|}{|C_{=n}|} = \frac{1}{2}.$$

We write $\mathrm{WS}(t, q, \nu)$ for the set of all weakly (t, q, ν)-stochastic languages.

The following result resembles the weak stochasticity theorems proved by Lutz and Mayordomo [11] and Lutz [12] but gives a more careful upper bound on the time complexity of the martingale.

Theorem 2.4 (weak stochasticity theorem). *Assume that $\alpha, \beta, \gamma, \tau \in \mathrm{R}$ satisfy $\alpha \geq 1, \beta \geq 1, \gamma > 0$, and $\tau > \alpha\beta$. Then there is an exact $2^{(\log n)^\tau}$-martingale d such that*

$$S^\infty[d] \cup \mathrm{WS}(2^{n^\alpha}, n^\beta, 2^{\gamma n}) = \mathbf{C}.$$

Proof. Assume the hypothesis, and assume without loss of generality that $\alpha, \beta, \gamma, \tau \in \mathbb{Q}$. Fix $\alpha', \tau', \tau'' \in Q$ such that $\alpha < \alpha'$ and $\alpha'\beta < \tau'' < \tau' < \tau$. Let $U \in \mathrm{DTIME}(2^{n^{\alpha'}})$ be a language that is universal for $\mathrm{DTIME}(2^{n^\alpha}) \times \mathrm{DTIME}(2^{n^\alpha})$ in the following sense. For each $i \in \mathbb{N}$, let

$$C_i = \{x \in \{0,1\}^* | \langle s_i, 0x\rangle \in U\},$$

$$D_i = \{x \in \{0,1\}^* | \langle s_i, 1x\rangle \in U\}.$$

Then $\mathrm{DTIME}(2^{n^\alpha}) \times \mathrm{DTIME}(2^{n^\alpha}) = \{(C_i, D_i) | i \in \mathbb{N}\}$.

Define a function $d' : \mathbb{N}^3 \times \{0,1\}^* \to \mathbb{Q} \cap [0, \infty)$ as follows. If k is not a power of 2, then $d'_{i,j,k}(w) = 0$. Otherwise, if $k = 2^n$, where $n \in \mathbb{N}$, then

$$d'_{i,j,k}(w) = \sum_{y,z \in \{0,1\}^{\leq n^\beta}} \Pr(Y_{i,j,k,y,z} | \mathbf{C}_w),$$

where the sets $Y_{i,j,k,y,z}$ are defined as follows. If $|(C_i/y)_{=n}| < 2^{\gamma n}$, then $Y_{i,j,k,y,z} = \emptyset$. If $|(C_i/y)_{=n}| \geq 2^{\gamma n}$, then $Y_{i,j,k,y,z}$ is the set of all $A \in \mathbf{C}$ such that

$$\left| \frac{|(A \triangle (D_i/z)) \cap (C_i/y)_{=n}|}{|(C_i/y)_{=n}|} - \frac{1}{2} \right| \geq \frac{1}{j+1}.$$

The definition of conditional probability immediately implies that, for each $i, j, k \in \mathbb{N}$, the function $d'_{i,j,k}$ is a martingale. Since $U \in \mathrm{DTIME}(2^{n^{\alpha'}})$ and $\alpha'\beta < \tau''$, the time required to compute each $\Pr(Y_{i,j,k,y,z}|\mathbf{C}_w)$ using binomial coefficients is at most $O(2^{(\log(i+j+k))^{\tau''}})$ steps, so the time required to compute $d'_{i,j,k}(w)$ is at most $O((2^{n^\beta}+1)^2 \cdot 2^{(\log(i+j+k))^{\tau''}}) = O(2^{(\log(i+j+k))^{\tau'}})$ steps. Thus d' is exactly $2^{(\log n)^{\tau'}}$-time-computable.

As in [11] and [12], the Chernoff bound tells us that, for all $i, j, n \in \mathbb{N}$ and $y, z \in \{0,1\}^{\leq n^\beta}$, writing $k = 2^n$,

$$\Pr(Y_{i,j,k,y,z}) \leq 2e^{-k^\gamma/2(j+1)^2},$$

whence

$$\begin{aligned} d'_{i,j,k}(\lambda) &\leq (2^{n^\beta}+1)^2 \cdot 2e^{-k^\gamma/2(j+1)^2} \\ &< e^{2n^\beta+3-k^\gamma/2(j+1)^2}. \end{aligned}$$

Let $a = \lceil \frac{1}{\gamma} \rceil$, let $\epsilon = \frac{\gamma}{4}$, and fix $k_0 \in \mathbb{N}$ such that

$$k^{2\epsilon} > k^\epsilon + 2(\log k)^\beta + 3$$

for all $k \geq k_0$. Define $g : \mathbb{N} \to \mathbb{N}$ by

$$g(j) = 4^a(j+1)^{4a} + k_0$$

for all $j \in \mathbb{N}$. Then g is a polynomial and, for all $i, j, n \in \mathbb{N}$, writing $k = 2^n$,

$$\begin{aligned} k \geq g(j) \Rightarrow \begin{cases} k^\gamma &= k^{2\epsilon}k^{2\epsilon} \\ &> [4^a(j+1)^{4a}]^{2\epsilon}(k^\epsilon + 2(\log k)^\beta + 3) \\ &\geq 2(j+1)^2(k^\epsilon + 2n^\beta + 3) \end{cases} \\ \Rightarrow d'_{i,j,k}(\lambda) < e^{-k^\epsilon}. \end{aligned}$$

It follows by Lemma 2.2 that the series $\sum_{k=0}^\infty d'_{i,j,k}(\lambda)$, for $i, j \in \mathbb{N}$, are uniformly p-convergent. Since $1 < \tau'' < \tau$, it follows by Theorem 2.3 that there is an exact $2^{(\log n)^\tau}$-martingale d such that

$$\bigcup_{i=0}^\infty \bigcup_{j=0}^\infty \bigcap_{t=0}^\infty \bigcup_{k=t}^\infty S^1[d'_{i,j,k}] \subseteq S^\infty[d]. \tag{2.4}$$

Now assume that $A \notin \mathrm{WS}(2^{n^\alpha}, n^\beta, 2^{\gamma n})$. Then, by the definition of weak stochasticity, we can fix $i, j \in \mathbb{N}$, functions $h_1, h_2 \in \mathrm{ADV}(n^\beta)$, and an infinite set $J \subseteq \mathbb{N}$ such that, for all $n \in J$, $A \in Y_{i,j,k,h_1(n),h_2(n)}$, where $k = 2^n$. For each $n \in J$, then, there is a prefix $w \sqsubseteq A$ such that $\mathbf{C}_w \subseteq Y_{i,j,k,h_1(n),h_2(n)}$, whence

$$d'_{i,j,k}(w) \geq \Pr(Y_{i,j,k,h_1(n),h_2(n)}|\mathbf{C}_w) = 1,$$

i.e., $A \in S^1[d'_{i,j,k}]$. This argument shows that

$$\bigcup_{i=0}^\infty \bigcup_{j=0}^\infty \bigcap_{t=0}^\infty \bigcup_{k=t}^\infty S^1[d'_{i,j,k}] \cup \mathrm{WS}(2^{n^\alpha}, n^\beta, 2^{\gamma n}) = \mathbf{C}.$$

If follows by (1.4) that

$$S^\infty[d] \cup \mathrm{WS}(2^{n^\alpha}, n^\beta, 2^{\gamma n}) = \mathbf{C}. \qquad \square$$

The technique of martingale dilation was introduced by Ambos-Spies, Terwijn, and Zheng [3]. It has also been used by Juedes and Lutz [9] and generalized considerably by Breutzmann and Lutz [6]. We use the notation of [9] here.

The *restriction* of a string $w = b_0 b_1 \cdots b_{n-1} \in \{0,1\}^*$ to a language $A \subseteq \{0,1\}^*$ is the string $w\restriction A$ obtained by concatenating the successive bits b_i for which $s_i \in A$. If $f : \{0,1\}^* \to \{0,1\}^*$ is strictly increasing and d is a martingale, then the *f-dilation* of d is the function $f\hat{}d : \{0,1\}^* \to [0,\infty)$ defined by

$$f\hat{}d(w) = d(w\restriction range(f))$$

for all $w \in \{0,1\}^*$.

LEMMA 2.5 (martingale dilation lemma—Ambos-Spies, Terwijn, and Zheng [3]). *If $f : \{0,1\}^* \to \{0,1\}^*$ is strictly increasing and d is a martingale, then $f\hat{}d$ is also a martingale. Moreover, for every language $A \in \{0,1\}^*$, if d succeeds on $f^{-1}(A)$, then $f\hat{}d$ succeeds on A.*

Finally, we summarize the most basic ideas of resource-bounded measures in E and E_2. A p-*martingale* is a martingale that is, for some $k \in \mathbb{N}$, an n^k-martingale. A p_2-*martingale* is a martingale that is, for some $k \in \mathbb{N}$, a $2^{(\log n)^k}$-martingale.

DEFINITION (Lutz [10]).

1. *A set X of languages has* p-measure 0*, and we write $\mu_{\mathrm{p}}(X) = 0$, if there is a* p-*martingale d such that $X \subseteq S^\infty[d]$.*
2. *A set X of languages has* p_2-measure 0*, and we write $\mu_{\mathrm{p}_2}(X) = 0$, if there is a* p_2-*martingale d such that $X \subseteq S^\infty[d]$.*

DEFINITION (Lutz [10]).

1. *A set X of languages has* measure 0 *in* E*, and we write $\mu(X|\mathrm{E}) = 0$, if $\mu_{\mathrm{p}}(X \bigcap \mathrm{E}) = 0$.*
2. *A set X of languages has* measure 0 *in* E_2*, and we write $\mu(X|\mathrm{E}_2) = 0$, if $\mu_{\mathrm{p}_2}(X \bigcap \mathrm{E}_2) = 0$.*
3. *A set X of languages has* measure 1 *in* E*, and we write $\mu(X|\mathrm{E}) = 1$, if $\mu(X^c|\mathrm{E}) = 0$. In this case, we say that X contains* almost every *element of* E*.*
4. *A set X of languages has* measure 1 *in* E_2*, and we write $\mu(X|\mathrm{E}_2) = 1$, if $\mu(X^c|\mathrm{E}_2) = 0$. In this case, we say that X contains* almost every *element of* E_2*.*
5. *The expression $\mu(X|\mathrm{E}) \neq 0$ means that X does* not *have measure* 0 *in* E*. Note that this does* not *assert that "$\mu(X|\mathrm{E})$" has some nonzero value. Similarly, the expression $\mu(X|\mathrm{E}_2) \neq 0$ means that X does not have measure* 0 *in* E_2*.*

It is shown in [10] that these definitions endow E and E_2 with internal measure structure. This structure justifies the intuition that, if $\mu(X|\mathrm{E}) = 0$, then $X \cap \mathrm{E}$ is a *negligibly small* subset of E (and similarly for E_2).

3. Results. The key to our main theorem is the following lemma, which says that languages that are $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$-reducible to nondense languages cannot be very stochastic.

LEMMA 3.1 (main lemma). *For all real numbers $\alpha < 1$ and $\beta > 1 + \alpha$,*

$$\mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c) \cap \mathrm{WS}(2^n, n^\beta, 2^{\frac{n}{2}}) = \emptyset.$$

Proof. Let $\alpha < 1$ and $\beta > 1+\alpha$, and assume without loss of generality that α and β are rational. Let $A \in \mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c)$. It suffices to show that A is not weakly $(2^n, n^\beta, 2^{\frac{n}{2}})$-stochastic.

Since $A \in \mathrm{P}_{n^\alpha-\mathrm{T}}(\mathrm{DENSE}^c)$, there exist a nondense language S, a polynomial $q(n)$, and a $q(n)$-time-bounded oracle Turing machine M such that $A = L(M^S)$ and, for every $x \in \{0,1\}^*$ and $B \subseteq \{0,1\}^*$, M makes exactly $\lfloor |x|^\alpha \rfloor$queries (all distinct) on input x with oracle B. Call these queries $Q^B(x,1), \ldots, Q^B(x, \lfloor |x|^\alpha \rfloor)$ in the order in which M makes them.

For each $B \in \{0,1\}^*$ and $n \in \mathbb{N}$, define an equivalence relation $\approx_{B,n}$ on $\{0,1\}^{\leq q(n)}$ by

$$u \approx_{B,n} v \Leftrightarrow (\forall w)[u \leq w \leq v \Rightarrow [\![w \in B]\!] = [\![u \in B]\!]]$$

and an equivalence relation $\equiv_{B,n}$ on $\{0,1\}^n$ by

$$x \equiv_{B,n} y \Leftrightarrow (\forall i)[1 \leq i \leq n^\alpha \Rightarrow Q^B(x,i) \approx_{B,n} Q^B(y,i)].$$

Note that $\approx_{B,n}$ has at most $2|B_{\leq q(n)}| + 1$ equivalence classes, so $\equiv_{B,n}$ has at most $(2|B_{\leq q(n)}| + 1)^{n^\alpha}$ equivalence classes.

Let $\epsilon = \frac{1-\alpha}{2}$, and let J be the set of all $n \in \mathbb{N}$ for which the following three conditions hold.

(i) $2|S_{\leq q(n)}| + 1 \leq 2^{n^\epsilon}$.
(ii) $n^{\alpha+\epsilon} \leq \frac{n}{2}$.
(iii) $n^\alpha(2n+1) \leq n^\beta$.

Since $\alpha + \epsilon < 1$ and $\beta > 1 + \alpha$, conditions (ii) and (iii) hold for all sufficiently large n. Since $\epsilon > 0$ and S is not dense, condition (i) holds for infinitely many n. Thus the set J is infinite.

Define an advice function $h : \mathbb{N} \to \{0,1\}^*$ as follows. If $n \notin J$, then $h(n) = \lambda$. If $n \in J$, then let D_n be a maximum-cardinality equivalence class of the relation $\equiv_{S,n}$. For each $1 \leq i \leq \lfloor n^\alpha \rfloor$, fix strings $y_{n,i}, z_{n,i} \in D_n$ such that, for all $x \in D_n$,

$$Q^S(y_{n,i}, i) \leq Q^S(x,i) \leq Q^S(z_{n,i}, i).$$

Let

$$\begin{aligned} h_1(n) &= y_{n,1} \cdots y_{n,\lfloor n^\alpha \rfloor}, \\ h_2(n) &= z_{n,1} \cdots z_{n,\lfloor n^\alpha \rfloor}, \\ h_3(n) &= [\![Q^S(y_{n,1}, 1) \in S]\!] \cdots [\![Q^S(y_{n,\lfloor n^\alpha \rfloor}, \lfloor n^\alpha \rfloor) \in S]\!], \\ h(n) &= h_1(n)h_2(n)h_3(n). \end{aligned}$$

Note that $|h(n)| = \lfloor n^\alpha \rfloor (2n+1) \leq n^\beta$ for all $n \in J$, so $h \in \mathrm{ADV}(n^\beta)$.

For each $n \in \mathbb{N}$, let $t = \lfloor n^\alpha \rfloor$, and let C_n be the set of all coded pairs

$$\langle x, y_1 \cdots y_t z_1 \cdots z_t b_1 \cdots b_t \rangle$$

such that $x, y_1, \ldots, y_t, z_1, \ldots, z_t \in \{0,1\}^n$, $b_1, \ldots, b_t \in \{0,1\}$, and, for each $1 \leq i \leq t$,

$$Q^{b_1 \cdots b_t}(y_i, i) \leq Q^{b_1 \cdots b_t}(x,i) \leq Q^{b_1 \cdots b_t}(z_i, i),$$

where $Q^{b_1 \cdots b_t}(w,i)$ denotes the ith query of M on input w when the successive oracle answers are $b_1, \ldots, b_t$. Let B_n be the set of all such coded pairs in C_n such that M

accepts on input x when the successive oracle answers are $b_1, \ldots, b_t$. Finally, define the languages

$$B = \{\langle x, v\rangle \mid v = \lambda \text{ or} \langle x, v\rangle \in B_{|x|}\},$$

$$C = \{\langle x, v\rangle \mid v = \lambda \text{ or} \langle x, v\rangle \in C_{|x|}\}.$$

It is clear that $B, C \in \mathrm{DTIME}(2^n)$. Also, by our construction of these sets and the advice function h, for each $n \in \mathbb{N}$, we have

$$(C/h)_{=n} = \begin{cases} D_n & \text{if } n \in J, \\ \{0,1\}^n & \text{if } n \notin J \end{cases}$$

and

$$(B/h)_{=n} = \begin{cases} A \cap D_n & \text{if } n \in J, \\ \{0,1\}^n & \text{if } n \notin J. \end{cases}$$

For each $n \in J$, if $\kappa(n)$ is the number of equivalence classes of $\equiv_{S,n}$, then

$$\kappa(n) \leq (2|S_{\leq q(n)}| + 1)^{n^\alpha} \leq (2^{n^\epsilon})^{n^\alpha} = 2^{n^{\alpha+\epsilon}},$$

so

$$|D_n| \geq \frac{2^n}{\kappa(n)} \geq 2^{n - n^{\alpha+\epsilon}} \geq 2^{\frac{n}{2}}.$$

It follows that $|(C/h)_{=n}| \geq 2^{\frac{n}{2}}$ for all $n \in \mathbb{N}$.

Finally, for all $n \in J$,

$$(A \triangle (B/h)) \cap (C/h)_{=n} = (A \triangle (A \cap D_n)) \cap D_n = \emptyset.$$

Since J is infinite, it follows that

$$\frac{|(A \triangle (B/h)) \cap (C/h)_{=n}|}{|(C/h)_{=n}|} \not\to \frac{1}{2}$$

as $n \to \infty$. Since $B, C \in \mathrm{DTIME}(2^n)$, $h \in \mathrm{ADV}(n^\beta)$, and $|(C/h)_{=n}| \geq 2^{\frac{n}{2}}$ for all $n \in \mathbb{N}$, this shows that A is not weakly $(2^n, n^\beta, 2^{\frac{n}{2}})$-stochastic. □

We now prove our main result.

THEOREM 3.2 (main theorem). *For every real number $\alpha < 1$,*

$$\mu_{\mathrm{p}}(\mathrm{P}_{n^{\alpha/2}\text{-T}}(\mathrm{DENSE}^c)) = \mu_{\mathrm{p}_2}(\mathrm{P}_{n^\alpha\text{-T}}(\mathrm{DENSE}^c)) = 0.$$

Proof. Let $\alpha < 1$, and let $\beta = \frac{3+\alpha}{2}$, so that $1 + \alpha < \beta < 2$. By Theorem 2.4, there is an exact $2^{(\log n)^2}$-martingale d such that

$$S^\infty[d] \cup \mathrm{WS}(2^n, n^\beta, 2^{\frac{n}{2}}) = \mathbf{C}.$$

By Lemma 3.1, we then have

$$\mathrm{P}_{n^\alpha\text{-T}}(\mathrm{DENSE}^c) \subseteq S^\infty[d].$$

Since d is a p_2-martingale, this implies that $\mu_{\mathrm{p}_2}(\mathrm{P}_{n^\alpha\text{-T}}(\mathrm{DENSE}^c)) = 0$.

Define $f : \{0,1\}^* \to \{0,1\}^*$ by

$$f(x) = 0^{|x|^2-|x|-1}1x.$$

Then f is strictly increasing, so $f\hat{}d$, the f-dilation of d, is a martingale. The time required to compute $f\hat{}d(w)$ is

$$O(|w|^2 + 2^{(\log |w'|)^2})$$

steps, where $w' = w\restriction range(f)$. (This allows $O(|w|^2)$ steps to compute w' and then $O(2^{(\log |w'|)^2})$ steps to compute $d(w')$.)

Now $|w'|$ is bounded above by the number of strings x such that $|x|^2 \leq |s_{|w|}| = \lfloor \log(1 + |w|) \rfloor$, so

$$|w'| < 2^{1+\sqrt{\log(1+|w|)}}.$$

Thus the time required to compute $f\hat{}d(w)$ is

$$O(|w|^2 + 2^{(1+\sqrt{\log(1+|w|)}\,)^2}) = O(|w|^2)$$

steps, so $f\hat{}d$ is an n^2-martingale.

Now let $A \in \mathrm{P}_{n^{\alpha/2}-\mathrm{T}}(\mathrm{DENSE}^c)$. Then $f^{-1}(A) \in \mathrm{P}_{n^{\alpha}-\mathrm{T}}(\mathrm{DENSE}^c) \subseteq S^\infty[d]$, so $A \in S^\infty[f\hat{}d]$ by Lemma 2.5. This shows that $\mathrm{P}_{n^{\alpha/2}-\mathrm{T}}(\mathrm{DENSE}^c) \subseteq S^\infty[f\hat{}d]$. Since $f\hat{}d$ is an n^2-martingale, it follows that $\mu_{\mathrm{p}}(\mathrm{P}_{n^{\alpha/2}-\mathrm{T}}(\mathrm{DENSE}^c)) = 0$. □

We now develop a few consequences of the main theorem. The first is immediate.

Corollary 3.3. *For every real number $\alpha < 1$,*

$$\mu(\mathrm{P}_{n^{\alpha/2}-\mathrm{T}}(\mathrm{DENSE}^c) \mid \mathrm{E}) = \mu(\mathrm{P}_{n^{\alpha}-\mathrm{T}}(\mathrm{DENSE}^c) \mid \mathrm{E}_2) = 0.$$

The following result on the density of weakly complete (or weakly hard) languages now follows immediately from Corollary 3.3.

Corollary 3.4. *For every real number $\alpha < 1$, every language that is weakly $\leq^{\mathrm{P}}_{n^{\alpha/2}-\mathrm{T}}$-hard for* E *or weakly $\leq^{\mathrm{P}}_{n^{\alpha}-\mathrm{T}}$-hard for* E_2 *is dense.*

Our final two corollaries concern consequences of the strong hypotheses $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ and $\mu_{\mathrm{p}_2}(\mathrm{NP}) \neq 0$. The relative strengths of these hypotheses are indicated by the known implications

$$\mu(\mathrm{NP} \mid \mathrm{E}) \neq 0 \Rightarrow \mu(\mathrm{NP} \mid \mathrm{E}_2) \neq 0 \Leftrightarrow \mu_{\mathrm{p}_2}(\mathrm{NP}) \neq 0 \Leftrightarrow \mu_{\mathrm{p}}(\mathrm{NP}) \neq 0 \Rightarrow \mathrm{P} \neq \mathrm{NP}.$$

(The leftmost implication was proven by Juedes and Lutz [9], and the fact that $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$ implies $\mu_{\mathrm{p}_2}(\mathrm{NP}) \neq 0$ was proven by Ambos-Spies, Terwijn, and Zheng [3]. The remaining implications follow immediately from elementary properties of resource-bounded measure.)

Corollary 3.5. *Let $\alpha < 1$. If $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$, then every language that is $\leq^{\mathrm{P}}_{n^{\alpha}-\mathrm{T}}$-hard for* NP *is dense.*

We conclude by considering the densities of languages to which SAT can be adaptively reduced.

Definition. *A function $g : \mathbb{N} \to \mathbb{N}$ is* subradical *if $\log g(n) = o(\log n)$.*

It is easy to see that a function g is subradical if and only if, for all $k > 0$, $g(n) = o(\sqrt[k]{n})$. (This is the reason for the name "subradical.") Subradical functions

include very slow-growing functions such as $\log n$ and $(\log n)^5$, as well as more rapidly growing functions such as $2^{(\log n)^{0.99}}$.

COROLLARY 3.6. *If* $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$, $g : \mathbb{N} \to \mathbb{N}$ *is subradical, and SAT* $\leq^{\mathrm{P}}_{g(n)-\mathrm{T}} H$, *then* H *is dense.*

Proof. Assume the hypothesis. Let $A \in \mathrm{NP}$. Then there is a $\leq^{\mathrm{P}}_{\mathrm{m}}$-reduction f of A to SAT. Fix a polynomial $q(n)$ such that, for all $x \in \{0,1\}^*$, $|f(x)| \leq q(|x|)$. Composing f with the $\leq^{\mathrm{P}}_{g(n)-\mathrm{T}}$-reduction of SAT to H that we have assumed to exist then gives a $\leq^{\mathrm{P}}_{g(q(n))-\mathrm{T}}$-reduction of A to H. Since g is subradical, $\log g(q(n)) = o(\log q(n)) = o(\log n)$, so for all sufficiently large n, $g(q(n)) \leq 2^{\frac{\log n}{4}} = n^{\frac{1}{4}}$. Thus $A \leq^{\mathrm{P}}_{n^{\frac{1}{4}}-\mathrm{T}} H$.

The above argument shows that H is $\leq^{\mathrm{P}}_{n^{\frac{1}{4}}-\mathrm{T}}$-hard for NP. Since we have assumed $\mu_{\mathrm{p}}(\mathrm{NP}) \neq 0$, it follows by Corollary 3.5 that H is dense. □

To put the matter differently, Corollary 3.6 tells us that if SAT is polynomial-time reducible to a nondense language with at most $2^{(\log n)^{0.99}}$ adaptive queries, then NP has measure 0 in E and in E_2.

4. Questions. As noted in the introduction, the relationships between weak hardness notions for E and E_2 under reducibilities such as $\leq^{\mathrm{P}}_{\mathrm{T}}, \leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$ and $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{tt}}$ remain to be resolved. Our main theorem also leaves open the question whether $\leq^{\mathrm{P}}_{n^\alpha-\mathrm{T}}$-hard languages for E must be dense when $\frac{1}{2} \leq \alpha < 1$. We are in the curious situation of knowing that the classes $\mathrm{P}_{n^{0.99}-\mathrm{tt}}(\mathrm{DENSE}^c)$ and $\mathrm{P}_{n^{0.49}-\mathrm{T}}(\mathrm{DENSE}^c)$ have p-measure 0, but not knowing whether the class $\mathrm{P}_{n^{0.50}-\mathrm{T}}(\mathrm{DENSE}^c)$ has p-measure 0. Indeed, at this time we cannot even prove that $\mathrm{E} \not\subseteq \mathrm{P}_{n^{0.50}-T}(\mathrm{SPARSE})$. Further progress on this matter would be illuminating.

REFERENCES

[1] K. AMBOS-SPIES, *Randomness, relativizations, and polynomial reducibilities*, in Proceedings of the First Structure in Complexity Theory Conference, Berkeley, CA, Springer-Verlag, 1986, pp. 23–34.

[2] K. AMBOS-SPIES AND E. MAYORDOMO, *Resource-bounded measure and randomness*, in Complexity, Logic, and Recursion Theory, Lecture Notes in Pure and Appl. Math. 187, A. Sorbi, ed., Marcel Dekker, New York, 1997, pp. 1–47.

[3] K. AMBOS-SPIES, S. A. TERWIJN, AND X. ZHENG, *Resource-bounded randomness and weakly complete problems*, Theoret. Comput. Sci., 172 (1997), pp. 195–207.

[4] C. H. BENNETT AND J. GILL, *Relative to a random oracle A,* $\mathrm{P}^A \neq \mathrm{NP}^A \neq co - \mathrm{NP}^A$ *with probability* 1, SIAM J. Comput., 10 (1981), pp. 96–113.

[5] L. BERMAN AND J. HARTMANIS, *On isomorphisms and density of* NP *and other complete sets*, SIAM J. Comput., 6 (1977), pp. 305–322.

[6] J. M. BREUTZMANN AND J. H. LUTZ, *Equivalence of measures of complexity classes*, SIAM J. Comput., 29 (1999), pp. 302–326.

[7] H. BUHRMAN AND D. VAN MELKEBEEK, *Hard sets are hard to find*, in Proceedings of the 13th IEEE Conference on Computational Complexity, Buffalo, NY, 1998, pp. 170–181.

[8] B. FU, *With quasilinear queries,* EXP *is not polynomial time Turing reducible to sparse sets*, SIAM J. Comput., 24 (1995), pp. 1082–1090.

[9] D. W. JUEDES AND J. H. LUTZ, *Weak completeness in* E *and* E_2, Theoret. Comput. Sci., 143 (1995), pp. 149–158.

[10] J. H. LUTZ, *Almost everywhere high nonuniform complexity*, J. Comput. System Sci., 19 (1990), pp. 1100–1131.

[11] J. H. LUTZ AND E. MAYORDOMO, *Measure, stochasticity, and the density of hard languages*, SIAM J. Comput., 23 (1994), pp. 762–779.

[12] J. H. LUTZ, *One-way functions and balanced* NP, Theoret. Comput. Sci., to appear.

[13] J. H. Lutz, *The quantitative structure of exponential time*, in L. A. Hemaspaandra and A. L. Selman, eds., Complexity Theory Retrospective II, Springer-Verlag, New York, 1997, pp. 225–260.

[14] S. R. Mahaney, *Sparse complete sets for* NP*: Solution of a conjecture of Berman and Hartmanis*, J. Comput. System Sci., 25 (1982), pp. 130–143.

[15] E. Mayordomo, *Contributions to the Study of Resource-Bounded Measure*, Ph.D. thesis, Universitat Politècnica de Catalunya, Barcelona, Spain, 1994.

[16] A. R. Meyer and L. Stockmeyer, *The equivalence problem for regular expressions with squaring requires exponential space*, in Proceedings of the 13th Annual IEEE Symposium on Switching and Automata Theory, IEEE Computer Society, Los Alamitos, CA, 1972, pp. 125–129.

[17] M. Ogiwara and O. Watanabe, *On polynomial time bounded truth-table reducibility of* NP *sets to sparse sets*, SIAM J. Comput., 20 (1991), pp. 471–483.

[18] O. Watanabe, *On the Structure of Intractable Complexity Classes*, Ph.D. thesis, Tokyo Institute of Technology, Tokyo, Japan, 1987.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1211–1230

A FINITE STATE VERSION OF THE KRAFT–MCMILLAN THEOREM*

FRÉDÉRIQUE BASSINO†, MARIE-PIERRE BÉAL†, AND DOMINIQUE PERRIN†

Abstract. The main result is a finite-state version of the Kraft–McMillan theorem characterizing the generating sequence of a k-ary regular tree. The proof uses a new construction called the *multiset construction*, which is a version with multiplicities of the well-known subset construction of automata theory.

Key words. generating series, regular trees, nonnegative matrices

AMS subject classifications. 68Q45, 68R10, 94A45, 37B10

PII. S0097539798343908

1. Introduction. The Kraft inequality $\sum_{n\geq 0} s_n k^{-n} \leq 1$ characterizes the generating sequences $(s_n)_{n\geq 0}$ of leaves in a k-ary tree. It is used in connection with the Huffman algorithm to build prefix codes or search trees and usually restricted to the case of finite trees. We are interested here in the case of infinite sequences corresponding to infinite trees. These infinite trees arise, for example, as search trees in infinite sets. They also appear in the context of finite automata having nested loops to represent the set of first returns to a given state. The tree thus obtained is called a *regular* tree. It has only a finite number of nonisomorphic subtrees since two subtrees corresponding to the same state of the automaton are isomorphic. The generating sequences of such infinite trees are of interest in the applications of finite automata to text compression or channel coding.

Our main result is a characterization of the generating sequences of leaves of regular k-ary trees. Its essence is that the two conditions of being the generating sequence of

(i) a k-ary tree, and

(ii) a regular tree

are independent in the sense that their conjunction is enough to guarantee that a sequence is the generating sequence of a regular k-ary tree.

The proof uses a new construction on graphs called the *multiset construction*, which is a counterpart for automata with multiplicities of the well-known subset construction of automata theory.

Our results have a connection with symbolic dynamics. Actually, in both cases, the emphasis is on the space of paths in a finite graph. Even if we do not use results from symbolic dynamics, some of the methods used, like state-splitting or the Perron theory, are similar. Using an expression of Lind and Marcus [15], our treatment is "dynamical in spirit." The relationship with symbolic dynamics is discussed more closely in [7] and [8].

The paper is organized as follows. Section 2 contains preliminary results and definitions on graphs, trees, regular sequences, and the Perron–Frobenius theory. In

*Received by the editors August 27, 1998; accepted for publication (in revised form) December 3, 1999; published electronically October 6, 2000. Part of the results of this paper were presented at the conference LATIN '98 [F. Bassino, M.-P. Béal, and D. Perrin, *LATIN* '98, Lecture Notes in Comput. Sci. 1380, Springer-Verlag, Berlin, 1998, pp. 42–52].

http://www.siam.org/journals/sicomp/30-4/34390.html

†Institut Gaspard Monge, Université de Marne-la-Vallée, 77454 Marne-la-Vallée, Cedex 2, France (bassino@univ-mlv.fr, beal@univ-mlv.fr, perrin@univ-mlv.fr).

section 3, we present the multiset construction. Section 4 contains the proof of our main result (Theorem 16). Section 5 treats a similar problem, with the set of leaves replaced by the set of all nodes.

The results contained in this paper represent the terminal point of a series of steps. In a previous paper [7] (with a preliminary version in [5]), we proved Theorem 16 in the particular case of a strict inequality. The proof uses the technique of state-splitting from symbolic dynamics. In the same paper, we also give a proof of Theorem 19 which is different from the proof given here, which is based on the multiset construction and is more simple. Finally, the survey paper [8] gives an overview of length distributions and regular sequences.

2. Definitions and background. In this section, we fix our notation concerning graphs, trees, and regular sequences. We also recall some notions concerning positive matrices.

We give a word on the terminology used here. We constantly use the term *regular* where a richer terminology is often used. In particular, what we call a regular sequence here is, in Eilenberg's terminology, an $\mathbb{N}$-rational sequence (see [11], [19], or [10]).

2.1. Graphs and trees. In this paper, we use directed multigraphs, i.e., graphs with possibly several edges with the same origin and the same end. We simply call them graphs in all of what follows. We denote $G = (Q, E)$ a graph with Q as a set of vertices and E as a set of edges. We also say that G is a graph on the set Q.

A *tree* T on a set of nodes N with a *root* $r \in N$ is a function $T : N - \{r\} \longrightarrow N$, which associates to each node distinct from the root its father $T(n)$, in such a way that, for each node n, there is a nonnegative integer h such that $T^h(n) = r$. The integer h is the *height* of the node n.

A tree is k-ary if each node has at most k children. A node without children is called a *leaf*. A node which is not a leaf is called *internal*. A node n is a *descendant* of a node m if $m = T^h(n)$ for some $h \geq 0$. A k-ary tree is *complete* if all internal nodes have exactly k children and have at least one descendant which is a leaf.

For each node n of a tree T, the *subtree* rooted at n, denoted T_n, is the tree obtained by restricting the set of nodes to the descendants of n.

Two trees S, T are isomorphic, denoted $S \equiv T$, if there is a map which transforms S into T by permuting the children of each node. Equivalently, $S \equiv T$ if there is a bijective map $f : N \to M$ from the set of nodes of S onto the set of nodes of T such that $f \circ S = T \circ f$. Such a map f is called an isomorphism.

If T is a tree with N as set of nodes, the *quotient graph* of T is the graph $G = (Q, E)$ where Q and E are defined as follows. The set Q is the quotient of N by the equivalence $n \equiv m$ if $T_n \equiv T_m$. Let $\bar{m}$ denote the class of a node m. The number of edges from $\bar{m}$ to $\bar{n}$ is the number of children of m equivalent to n.

Conversely, the set of paths in a graph with given origin is a tree. Indeed, let $G = (Q, E)$ be a graph. Let $r \in Q$ be a particular vertex, and let N be the set of paths in G starting at r. The tree T having N as a set of nodes and such that $T(p_0, p_1, \ldots, p_n) = (p_0, p_1, \ldots, p_{n-1})$ is called the *covering tree* of G starting at r.

Both constructions are mutually inverse in the sense that any tree T is isomorphic to the covering tree of its quotient graph starting at the image of the root.

PROPOSITION 1. *Let T be a tree with root r. Let G be its quotient graph, and let i be the vertex of G which is the class of the root of T. For each vertex q of G and for each $n \geq 0$, the number of paths of length n from i to q is equal to the number of nodes of T at height n in the class of q.*

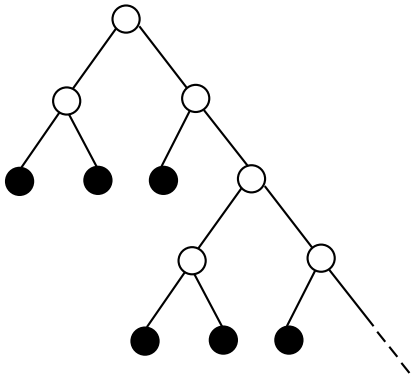


FIG. 1. *A regular tree.*

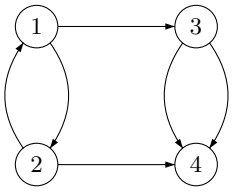


FIG. 2. *And its quotient graph.*

A tree is said to be *regular* if it admits only a finite number of nonisomorphic subtrees, i.e. if its quotient graph is finite.

For example, the infinite tree represented in Figure 1 is a regular tree. Its quotient graph is represented in Figure 2.

There is also a close connection between trees and sets of words on an alphabet. Let X be a set of words on the alphabet $\{0, 1, \dots, k-1\}$. The set X is said to be *prefix-closed* if any prefix of an element of X is also in X. When X is prefix-closed, we can build a tree $T(X)$ as follows. The set of nodes is X, the root is the empty word ϵ, and $T(a_1 a_2 \cdots a_n) = a_1 a_2 \cdots a_{n-1}$.

Let, for example, $X = \{\epsilon, 0, 1, 10, 11\}$. The tree $T(X)$ is represented in Figure 3.

2.2. Regular sequences. We consider sequences of natural integers $s = (s_n)_{n\geq 0}$. We shall not distinguish between such a sequence and the formal series $s(z) = \sum_{n\geq 0} s_n z^n$.

We usually denote a vector indexed by elements of a set Q, also called a Q-vector, with boldface symbols. For $\mathbf{v} = (v_q)_{q\in Q}$ we say that $\mathbf{v}$ is nonnegative, denoted $\mathbf{v} \geq 0$, (resp., positive, denoted $\mathbf{v} > 0$) if $v_q \geq 0$ (resp., $v_q > 0$) for all $q \in Q$. The same conventions are used for matrices. A nonnegative $Q \times Q$-matrix M is said to be *irreducible* if, for all indices p, q, there is an integer m such that $(M^m)_{p,q} > 0$. The matrix is *primitive* if there is an integer m such that $M^m > 0$.

The *adjacency matrix* of a graph $G = (Q, E)$ is the $Q \times Q$-matrix M such that, for each $p, q \in Q$, the integer $M_{p,q}$ is the number of edges from p to q. The adjacency matrix of a graph G is irreducible iff the graph is strongly connected. It is primitive if, moreover, the greatest commom divisor (g.c.d.) of lengths of cycles in G is 1.

Let G be a finite graph and let I, T be two sets of vertices. For each $n \geq 0$, let s_n

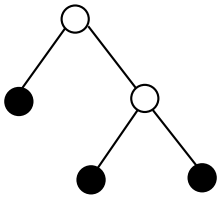


FIG. 3. *The tree* $T(X)$.

be the number of distinct paths of length n from a vertex of I to a vertex of T. The sequence $s = (s_n)_{n\geq 0}$ is called the sequence *recognized* by (G, I, T) or also by G if I and T are already specified. When $I = \{i\}$ and $T = \{t\}$, we simply denote (G, i, t) instead of $(G, \{i\}, \{t\})$.

A sequence $s = (s_n)_{n\geq 0}$ of nonnegative integers is said to be *regular* if it is recognized by such a triple (G, I, T), where G is finite. We say that the triple (G, I, T) is a *representation* of the sequence s. The vertices of I are called *initial* and those of T *terminal*. Two representations are said to be *equivalent* if they recognize the same sequence.

A representation (G, I, T) is said to be *trim* if every vertex of G is on some path from I to T. It is clear that any representation is equivalent to a trim one.

A well-known result in theory of finite automata allows one to use a particular representation of any regular sequence s such that $s_0 = 0$. One can always choose in this case a representation (G, i, t) of s with a unique initial vertex i and a unique final vertex $t \neq i$ such that no edge is entering vertex i and no edge is going out of vertex t. Such a representation is called a *normalized representation* (see, for example, [17, p. 14]).

Let (G, i, t) be a trim normalized representation. If we merge the initial vertex i and the final vertex t in a single vertex still denoted by i, we obtain a new graph denoted by $\overline{G}$, which is strongly connected. The triple $(\overline{G}, i, i)$ is called the *closure* of (G, i, t).

Let s be a regular sequence such that $s_0 = 0$. The *star* s^* of the sequence s is defined by

$$s^*(z) = \frac{1}{1 - s(z)}.$$

PROPOSITION 2. *If* (G, i, t) *is a normalized representation of* s*, its closure* $(\overline{G}, i, i)$ *recognizes the sequence* s^**.*

Proof. The sequence s is the length distribution of the paths of first returns to vertex i in $\overline{G}$, that is, of finite paths going from i to i without going through vertex i. The length distribution of the set of all returns to i is thus $1 + s(z) + s^2(z) + \cdots = 1/(1 - s(z))$. □

An equivalent definition of regular sequences uses vectors instead of the sets I, F. Let $\mathbf{i}$ be a Q-row vector of nonnegative integers, and let $\mathbf{t}$ be a Q-column vector of nonnegative integers. We say that $(G, \mathbf{i}, \mathbf{t})$ recognizes the sequence $s = (s_n)_{n\geq 0}$ if for each integer $n \geq 0$

$$s_n = \mathbf{i}M^n\mathbf{t},$$

where M is the adjacency matrix of G. The proof that both definitions are equivalent follows from the fact that the family of regular sequences is closed under addition (see

FIG. 4. *The Fibonacci graph.*

[11]). A triple $(G, \mathbf{i}, \mathbf{t})$ recognizing a sequence s is also called a representation of s, and two representations are called equivalent if they recognize the same sequence.

A sequence $s = (s_n)_{n\geq 0}$ of nonnegative integers is *rational* if it satisfies a recurrence relation with integral coefficients. Equivalently, s is rational if there exist two polynomials $p(z), q(z)$ with integral coefficients and with $q(0) = 1$ such that

$$s(z) = \frac{p(z)}{q(z)}.$$

Any regular sequence is rational. The converse is, however, not true (see section 5). For example, the sequence s defined by $s(z) = \frac{z}{1-z-z^2}$ is the sequence of Fibonacci numbers also defined by $s_0 = 0, s_1 = 1$, and $s_{n+1} = s_n + s_{n-1}$. It is recognized by the graph of Figure 4 with $I = \{1\}$ and $T = \{2\}$.

2.3. Regular sequences and trees. If T is a tree, its *generating sequence of leaves* is the sequence of numbers $s = (s_n)_{n\geq 0}$, where s_n is the number of leaves at height n. We also simply say that s is the *generating sequence* of T.

The following result is a direct consequence of the definitions.

THEOREM 3. *The generating sequence of a regular tree is a regular sequence.*

Proof. Let T be a regular tree, and let G be its quotient graph. Since T is regular, G is finite. The leaves of T form an equivalence class t. By Proposition 1, the generating sequence of T is recognized by (G, i, t), where i is the class of the root of T. □

We say that a sequence $s = (s_n)_{n\geq 1}$ satisfies the Kraft inequality for the integer k if

$$\sum_{n\geq 0} s_n k^{-n} \leq 1,$$

i.e., using the formal series $s(z) = \sum_{n\geq 0} s_n z^n$, if

$$s(1/k) \leq 1.$$

We say that s satisfies the strict Kraft inequality for k if $s(1/k) < 1$. The following result is well known (see [3, p. 35], for example).

THEOREM 4. *A sequence s is the generating sequence of a k-ary tree iff it satisfies the Kraft inequality for the integer k.*

Proof. Let first T be a k-ary tree, and let s be its generating sequence. It is enough to prove that, for each $n \geq 0$, the sequence $(s_0, \ldots, s_n)$ satisfies the Kraft inequality. It is the generating sequence of the finite tree obtained by restricting T to the nodes at height at most n. We may thus suppose T to be a finite tree. We have

$$s(z) = zt_1(z) + \cdots + zt_k(z),$$

where $t_1, \ldots, t_k$ are the generating sequences of leaves of the (possibly empty) subtrees rooted at the children of the root of T. By induction on the number of nodes, we have $t_i(1/k) \leq 1$, whence the desired result.

Conversely, we use an induction on n to prove that there exists a k-ary tree with generating sequence $(s_0, \dots, s_n)$. For $n = 0$, we have $s_0 \leq 1$ and T is either empty or reduced to one node. Suppose by induction hypothesis that we have already built a tree T with generating sequence $(s_0, s_1, \dots, s_{n-1})$. We have

$$\sum_{i=0}^{n} s_i k^{-i} \leq 1;$$

then

$$\sum_{i=0}^{n} s_i k^{n-i} \leq k^n,$$

and thus

$$s_n \leq k^n - \sum_{i=0}^{n-1} s_i k^{n-i}.$$

This allows us to add s_n leaves at height n to the tree T. □

Let us consider the Kraft equality case. If $s(1/k) = 1$, then any tree T having s as generating sequence is complete. The converse property is not true in general (see [11, p. 231]). However, it is a classical result that when T is a complete regular tree, its generating sequence satisfies $s(1/k) = 1$ (see Proposition 8).

For the sake of a complete description of the construction described above in the proof of Theorem 4, we have to specify the choice made at each step among the leaves at height n. A possible policy is to choose to give as many children as possible to the nodes which are not leaves and are of maximal height.

If we start with a finite sequence s satisfying the Kraft inequality, the above method builds a finite tree with a generating sequence equal to s. It is not true that this incremental method gives a regular tree when we start with a regular sequence, as shown in the following example.

Let $s(z) = z^2/(1-2z^2)$. Since $s(1/2) = 1/2$, we may apply the Kraft construction to build a binary tree with length distribution s. The result is the tree $T(X)$, where X is the set of prefixes of the set

$$Y = \bigcup_{n \geq 0} 01^n 0\{0,1\}^n,$$

which is not regular.

If s is a regular sequence such that $s_0 = 0$, there exists a regular tree T having s as a generating sequence. Indeed, let (G, i, t) be a normalized representation of s. The generating sequence of the covering tree of G starting at i is s. If s satisfies, moreover, the Kraft inequality for an integer k, it is, however, not true that the regular covering tree obtained is k-ary, as shown in the following example.

Let s be the regular sequence recognized by the graph of Figure 5 on the left with $i = 1$ and $t = 4$. We have $s(z) = 3z^2/(1 - z^2)$. Furthermore, $s(1/2) = 1$, and thus s satisfies Kraft's equality for $k = 2$. However, there are four edges going out of vertex 2 and its regular covering tree starting at 1 is 4-ary. A solution for this example is given by the graph of Figure 5 on the right. It recognizes s, and its covering tree starting at 1 is the regular binary tree of Figure 1.

The aim of section 4 is to build from a regular sequence s that satisfies the Kraft inequality for an integer k a tree with a generating sequence s which is both regular and k-ary.

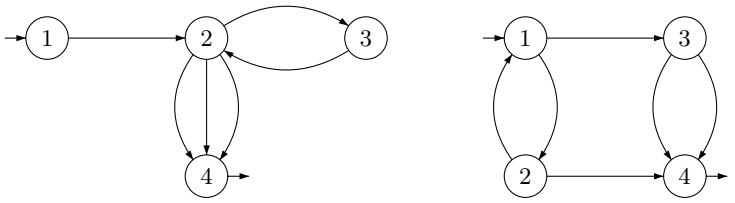


FIG. 5. *Graphs recognizing* $s(z) = 3z^2/(1-z^2)$.

2.4. Approximate eigenvector. Let M be the adjacency matrix of a graph G. By the Perron–Frobenius theorem (see [12] for a general presentation and [15], [14], or [9] for the link with graphs and regular sequences), the nonnegative matrix M has a nonnegative real eigenvalue of maximal modulus denoted by λ, also called the spectral radius of the matrix.

When G is strongly connected, the matrix is irreducible and the Perron–Frobenius theorem asserts that the dimension of the eigenspace of the matrix M corresponding to λ is equal to one, and that there is a positive eigenvector associated to λ.

Let k be an integer. A *k-approximate* eigenvector of a nonnegative matrix M is, by definition, an integral vector $\mathbf{v} \geq 0$ such that

$$M\mathbf{v} \leq k\mathbf{v}.$$

One has the following result (see [15, p. 152]).

PROPOSITION 5. *An irreducible nonnegative matrix M with spectral radius λ admits a positive k-approximate eigenvector iff $k \geq \lambda$.*

For a proof, see [15, p. 152]. When M is the adjacency matrix of a graph G, we also say that $\mathbf{v}$ is a k-approximate eigenvector of G. The computation of an approximate eigenvector can be obtained by the use of Franaszek's algorithm (see, for example, [15]). It can be shown that there exists a k-approximate eigenvector with elements bounded above by k^{2n}, where n is the dimension of M [4]. Thus the size of the coefficients of a k-approximate eigenvector is bounded above by an exponential in n and can be in the worst case of this order of magnitude.

The following result is well known. It links the radius of convergence of a sequence with the spectral radius of the associated matrix.

PROPOSITION 6. *Let s be a regular sequence recognized by a trim representation (G, I, T). Let M be the adjacency matrix of G. The radius of convergence of s is the inverse of the maximal eigenvalue of M.*

Proof. The maximal eigenvalue λ of M is $\lambda = \limsup_{n\geq 0} \sqrt[n]{\|M^n\|}$, where $\|\ \|$ is any of the equivalent matrix norms. Let ρ be the radius of convergence of s and, for each $p, q \in Q$, let ρ_{pq} be the radius of convergence of the sequence $u_{pq} = (M^n_{pq})_{n\geq 0}$. Then $1/\lambda = \min \rho_{pq}$. Since (G, I, T) is trim, we have $\rho_{pq} \geq \rho$ for all $p, q \in Q$. On the other hand, $\rho \geq \min \rho_{pq}$ since s is a sum of some of the sequences u_{pq}. Thus $\rho_s = \min \rho_{pq}$, which concludes the proof. □

As a consequence of this result, the radius of convergence ρ of a regular sequence s is a pole. Indeed, with the above notation, $s(z) = \mathbf{i}(1 - Mz)^{-1}\mathbf{t}$. Then $\det(I - Mz)$ is a denominator of the rational fraction s, and the poles of s are among the inverses of the eigenvalues of M. And since $1/\lambda$ is the radius of convergence of s, it has to be a pole of s. In particular, s diverges for $z = \rho$.

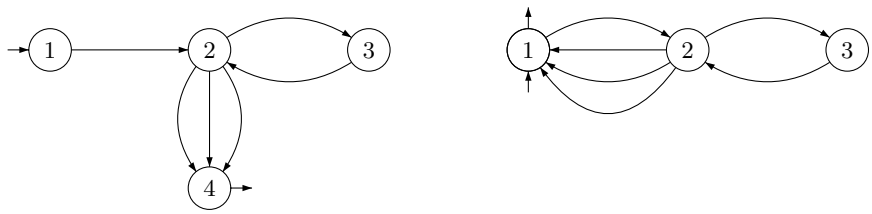


FIG. 6. *The graphs G and $\overline{G}$.*

The following result, due to Berstel, is also well known. It allows one to compute the radius of convergence of the star of a sequence.

PROPOSITION 7. *Let s be a regular sequence. The radius of convergence of the series $s^*(z) = 1/(1 - s(z))$ is the unique real number r such that $s(r) = 1$.*

For a proof, see [11, pp. 211–214], [10, p. 82], or [9, p. 84]. As a consequence, we obtain the following result.

PROPOSITION 8. *Let s be a regular sequence and let λ be the inverse of the radius of convergence of s^*. The sequence s satisfies the Kraft strict inequality $s(1/k) < 1$ (resp., equality $s(1/k) = 1$) iff $\lambda < k$ (resp., $\lambda = k$).*

We have thus proved the following result, which is the basis of the constructions of the next sections.

PROPOSITION 9. *Let s be a regular sequence satisfying Kraft's inequality $s(1/k) \leq 1$. Let (G, i, t) be a normalized representation of s and let $(\overline{G}, i, i)$ be the closure of (G, i, t). The adjacency matrix M of $\overline{G}$ admits a k-approximate eigenvector.*

Actually, under the hypothesis of Proposition 9, the graph G itself also admits a k-approximate eigenvector. Indeed, let $\overline{\mathbf{w}} = (\overline{w}_q)_{q \in Q-t}$ be a k-approximate eigenvector of $\overline{G}$. Then the vector $\mathbf{w} = (w_q)_{q \in Q}$, defined by $w_q = \overline{w}_q$ for $q \neq t$ and $w_t = \overline{w}_i$, is a k-approximate eigenvector of G. This is illustrated in the following example.

Let us, for example, consider again $s(z) = 3z^2/(1 - z^2)$ (see Figure 5). The sequence s is recognized by the normalized representation $(G, 1, 4)$, where G is the graph represented on the left of Figure 6. The graph $\overline{G}$ is represented on the right. The vectors

$$\mathbf{w} = \begin{bmatrix} 3 \\ 2 \\ 1 \\ 3 \end{bmatrix}, \overline{\mathbf{w}} = \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix}$$

are 2-approximate eigenvectors of G and $\overline{G}$, respectively.

3. The multiset construction. In this section, we present the main construction used in this paper. It can be considered as a version with multiplicities of the subset construction used in automata theory to replace a finite automaton by an equivalent deterministic one. We use only unlabeled graphs, but the construction can be easily generalized to graphs with edges labeled by symbols from an alphabet.

Our construction is also linked with one used by D. Lind to build a positive matrix with given spectral radius (see [15, especially Lemma 11.1.9]).

We use, for convenience, the term *multiset* of elements of a set Q as a synonym of Q-vector. If $\mathbf{u} = (u_q)_{q \in Q}$ is such a multiset, the coefficient u_q is also called the

multiplicity of q. The *degree* of $\mathbf{u}$ is the sum $\sum_{q\in Q} u_q$ of all multiplicities.

We start with a triple $(G, \mathbf{i}, \mathbf{t})$, where $G = (Q, E)$ is a finite graph and $\mathbf{i}$ (resp., $\mathbf{t}$) is a row (resp., column) Q-vector. We denote by M the adjacency matrix of G.

Let m be a positive integer. We define another triple $(H, \mathbf{J}, \mathbf{X})$ which is said to be obtained by the *multiset construction*. The graph H is called an *extension* of the graph G. The extension is not unique and depends as we shall see on some arbitrary choices. The set S of vertices of H is formed of multisets of elements of Q of total degree at most m. Thus, an element of S is a nonnegative vector $\mathbf{u} = (u_q)_{q\in Q}$ with indices in Q such that $\sum_{q\in Q} u_q \leq m$. This condition ensures that H is a finite graph.

We now describe the set of edges of the graph H by defining its adjacency matrix N. Let U be the $S \times Q$-matrix defined by $U_{\mathbf{u},q} = u_q$. Then N is any nonnegative $S \times S$-matrix which satisfies

$$NU = UM.$$

Equivalently, for all $\mathbf{u} \in S$,

$$\sum_{\mathbf{v}\in S} N_{\mathbf{u},\mathbf{v}}\mathbf{v} = \mathbf{u}M.$$

Let us comment informally on the above formula. We can describe the construction of the graph H as a sequence of choices. If we reach a vertex $\mathbf{u}$ of H, we partition the multiset $\mathbf{u}M$ of vertices reachable from the vertices composing $\mathbf{u}$ into multisets of degree at most m to define the vertices reachable from $\mathbf{u}$ in H. The integer $N_{\mathbf{u},\mathbf{v}}$ is the multiplicity of $\mathbf{v}$ in the partition. The formula simply expresses the fact that the result is indeed a partition. In general, there are several possible partitions. The matrix U is called the *transfer matrix* of the extension.

We further define the S-row vector $\mathbf{J}$ and the S-column vector $\mathbf{X}$. Let $\mathbf{J}$ be the S-row vector such that $J_{\mathbf{i}} = 1$ and $J_{\mathbf{u}} = 0$ for $\mathbf{u} \neq \mathbf{i}$. Let $\mathbf{X}$ be the S-column vector such that $X_{\mathbf{u}} = \mathbf{u} \cdot \mathbf{t}$.

Thus

$$\mathbf{J}U = \mathbf{i}, \quad \mathbf{X} = U\mathbf{t}.$$

To avoid unnecessary complexity, we only keep in S the vertices reachable from $\mathbf{i}$. Thus, we replace the set S by the set of elements $\mathbf{u}$ of S such that there is a path from $\mathbf{i}$ to $\mathbf{u}$.

The number of multisets of degree at most m on a set Q with n elements is $\frac{n^{m+1}-1}{n-1}$. Thus the number of vertices of a multiset extension is of order n^m. It is polynomial in n if m is taken as a constant.

Let, for example, G be the graph represented on Figure 7 on the left. The graph H represented on the right is a multiset extension of G with

$$\mathbf{i} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \quad \mathbf{j} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The matrices M, N, and U are

$$M = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}, N = \begin{bmatrix} 1 & 1 \\ 0 & 2 \end{bmatrix}, U = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}, \mathbf{J} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \mathbf{X} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

In this case, the matrix U is invertible, and the matrices M, N are conjugate.

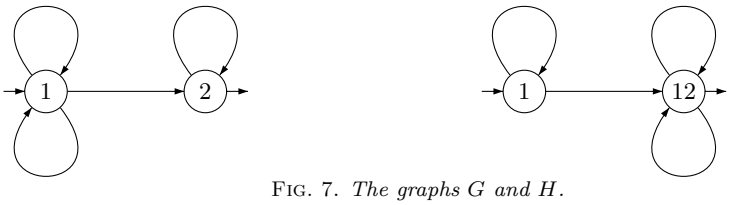


FIG. 7. *The graphs G and H.*

The basic property of an extension is the following one.

PROPOSITION 10. *Let H be an extension of G. The triple $(H, \mathbf{J}, \mathbf{X})$ is equivalent to $(G, \mathbf{i}, \mathbf{t})$.*

Proof. For each $n \geq 0$, we have

$$UM^n = N^n U.$$

Consequently, for each integer $n \geq 0$,

$$\begin{aligned} \mathbf{J}N^n\mathbf{X} &= \mathbf{J}N^nU\mathbf{t} \\ &= \mathbf{J}UM^n\mathbf{t} \\ &= \mathbf{i}M^n\mathbf{t}. \end{aligned}$$

This shows that $(H, \mathbf{J}, \mathbf{X})$ recognizes s. □

We will also make use of the following additional property of extensions.

PROPOSITION 11. *Let H be an extension of G. Let M (resp., N) be the adjacency matrix of G (resp., H), and let U be the transfer matrix. If $\mathbf{w}$ is a k-approximate eigenvector of M, the vector $\mathbf{W} = U\mathbf{w}$ is a k-approximate eigenvector of N. If $\mathbf{w}$ is positive, then $\mathbf{W}$ is also positive.*

Proof. We have

$$N\mathbf{W} = NU\mathbf{w} = UM\mathbf{w} \leq kU\mathbf{w} = k\mathbf{W}.$$

Since all rows of U are distinct from $\mathbf{0}$, the vector $\mathbf{W}$ is positive whenever $\mathbf{w}$ is positive. □

In the next section, we will choose a particular extension of the graph G called admissible which is defined as follows. Let $\mathbf{w}$ be a positive Q-vector, and let m be a positive integer. Let H be an extension of G, let U be the transfer matrix, and let $\mathbf{W} = U\mathbf{w}$. We say that H is *admissible* with respect to $\mathbf{w}$ and m if, for each $\mathbf{u} \in S$, all but possibly one of the vertices $\mathbf{v}$ such that $(\mathbf{u}, \mathbf{v})$ is an edge of H satisfy $W_{\mathbf{v}} \equiv 0 \bmod m$.

THEOREM 12. *For any graph G on Q, any positive Q-vector $\mathbf{w}$, and any integer $m > 0$, the graph G admits an admissible extension with respect to $\mathbf{w}$ and m.*

The proof relies on the following combinatorial lemma. This lemma is also used in a similar context by Adler, Coppersmith, and Hassner [1] and Marcus [16]. It is actually presented in [2] as a nice variant of the pigeon-hole principle.

LEMMA 13. *Let $w_1, w_2, \ldots, w_m$ be positive integers. Then there is a nonempty subset $S \subset \{1, 2, \ldots, m\}$ such that $\sum_{q \in S} w_q$ is divisible by m.*

Proof. The partial sums $w_1, w_1 + w_2, w_1 + w_2 + w_3, \ldots, w_1 + w_2 + \cdots + w_m$ either are all distinct (mod m), or two are congruent (mod m). In the former case,

at least one partial sum must be congruent to 0 (mod m). In the latter, there are $1 \leq p < r \leq m$ such that

$$w_1 + w_2 + \cdots + w_p \equiv w_1 + w_2 + \cdots + w_r (\text{mod } m).$$

Hence $w_{p+1} + w_{p+2} + \cdots + w_r \equiv 0 \pmod{m}$. □

Proof of Theorem 12. We build progressively the set of edges of H. Let $\mathbf{u}$ be an element of S. We prove by induction on the degree $d(\mathbf{u}M) = \sum_{q \in Q} (\mathbf{u}M)_q$ of $\mathbf{u}M$ that there exists $\mathbf{v}_1, \ldots, \mathbf{v}_n \in S$ such that $\mathbf{u}M = \sum_{i=1}^{n} \mathbf{v}_i$ and $W_{\mathbf{v}_i} \equiv 0 \bmod m$ for $1 \leq i \leq n-1$. If $\mathbf{u}M \in S$, i.e., if $d(\mathbf{u}M) \leq m$, we choose $n = 1$ and $\mathbf{v}_1 = \mathbf{u}M$. Otherwise, there exists a decomposition $\mathbf{u}M = \mathbf{v} + \mathbf{u}'$ such that $d(\mathbf{v}) = m$. Let $w_1, w_2, \ldots, w_m$ be the sequence of integers formed by the w_q repeated v_q times. By Lemma 13 applied to the sequence of integers w_i, there is a decomposition $\mathbf{v} = \mathbf{v}' + \mathbf{r}$ with $\mathbf{v}' \neq 0$ such that $W_{\mathbf{v}'} \equiv 0 \bmod m$. We have $\mathbf{u}M = \mathbf{v}' + \mathbf{w}'$ with $\mathbf{w}' = \mathbf{r} + \mathbf{u}'$. Since $d(\mathbf{w}') < d(\mathbf{u}M)$, we can apply the induction hypothesis to $\mathbf{w}'$, giving the desired result. □

For an S-vector $\mathbf{W}$, we denote by $\lceil \frac{\mathbf{W}}{m} \rceil$ the S-vector $\mathbf{Z}$ such that for each $\mathbf{u}$ in S,

$$Z_{\mathbf{u}} = \left\lceil \frac{W_{\mathbf{u}}}{m} \right\rceil.$$

Summing up the previous results, we obtain the following statement.

PROPOSITION 14. *Let H be an admissible extension of G with respect to $\mathbf{w}$ and m. Let M (resp., N) be the adjacency matrix of G (resp., H), let U be the transfer matrix, and let $\mathbf{W} = U\mathbf{w}$. If $\mathbf{w}$ is a positive k-approximate eigenvector of M, then $\lceil \frac{\mathbf{W}}{m} \rceil$ is a positive k-approximate eigenvector of N.*

Proof. By Proposition 3, the vector $\mathbf{W}$ is a positive k-approximate eigenvector of N. Thus

$$N\mathbf{W} \leq k\mathbf{W}.$$

Let $\mathbf{u}$ be an element of S. We have $W_{\mathbf{v}} \equiv 0 \bmod m$ for all indices $\mathbf{v}$ such that $N_{\mathbf{u},\mathbf{v}} > 0$ except possibly for an index $\mathbf{v}_0$. The previous inequality implies that

$$\sum_{\mathbf{v} \in S - \{\mathbf{v}_0\}} N_{\mathbf{u},\mathbf{v}} \frac{W_{\mathbf{v}}}{m} + N_{\mathbf{u},\mathbf{v}_0} \frac{W_{\mathbf{v}_0}}{m} \leq k \ \frac{W_{\mathbf{u}}}{m}.$$

Since $\frac{W_{\mathbf{v}}}{m}$ is a nonnegative integer for $\mathbf{v} \in Q - \{\mathbf{v}_0\}$, we get

$$\sum_{\mathbf{v} \in S - \{\mathbf{v}_0\}} N_{\mathbf{u},\mathbf{v}} \frac{W_{\mathbf{v}}}{m} + N_{\mathbf{u},\mathbf{v}_0} \left\lceil \frac{W_{\mathbf{v}_0}}{m} \right\rceil \leq k \ \left\lceil \frac{W_{\mathbf{u}}}{m} \right\rceil.$$

This proves that

$$N \left\lceil \frac{\mathbf{W}}{m} \right\rceil \leq k \left\lceil \frac{\mathbf{W}}{m} \right\rceil. \qquad \square$$

4. Generating sequence of leaves. In what follows, we state and prove, using the multiset construction, our main result concerning the generating sequences of regular trees. We begin with the following lemma, which is also used in the next section. We use the term leaf for a vertex of a graph without outgoing edges.

LEMMA 15. *Let G be a graph on a set Q of vertices. Let $i \in Q$ and $T \subset Q$. If G admits a k-approximate eigenvector $\mathbf{w}$, there is a graph G' and a set of vertices I' of G' such that the following hold.*

1. *G' admits the k-approximate eigenvector $\mathbf{w}'$ with all components equal to* 1.
2. *The triple $(G, i, \mathbf{w})$ is equivalent to the triple $(G', I', \mathbf{w}')$.*
3. *If $w_p = 1$ for all $p \in T$, there is a set of vertices T' of G' such that the triple (G, i, T) is equivalent to the triple (G', I', T'). Moreover, if T is the set of leaves of G, we can choose for T' the set of leaves of G'.*

Proof. We first show that one can replace G by a graph without multiplicities, i.e., such that the adjacency matrix has coefficients 0 or 1.

For this, let n be the maximal value of the coefficients of M. Let Q' be the set of all pairs (p, j) for $p \in Q$ and $1 \leq j \leq n$. Let E' be the set of all pairs $((p, j), (q, h)) \in Q' \times Q'$ such that $1 \leq j \leq n$ and $1 \leq h \leq M_{p,q}$. Let $i' = (i, 1)$ and $T' = \{(t, j) \mid t \in T, 1 \leq j \leq n\}$. Let $G' = (Q', E')$. The triple (G', i', T') recognizes the same sequence as (G, i, T). Let $w'_{(p,j)} = w_p$ for all $p \in Q$ and all $1 \leq j \leq n$. The triple $(G', i', \mathbf{w}')$ recognizes the same sequence as $(G, i, \mathbf{w})$. The vector $\mathbf{w}'$ is a k-approximate eigenvector of M'.

We may thus suppose that all coefficients of M are 0 or 1, i.e., that the set E of edges of G can be identified with a subset of $Q \times Q$. We now transform the graph G into a graph G' such that there are at most k edges going out of every vertex. For this, let Q' be the set of pairs (q, j) with $q \in Q$ and $1 \leq j \leq w_q$. For each $p \in Q$, we have

$$\sum_{q|(p,q)\in E} w_q \leq k w_p.$$

We may thus partition the pairs $(q, h) \in Q'$ in such a way that $(p, q) \in E$ in w_p groups $X_1, X_2, \ldots, X_{w_p}$ of at most k elements. The edges going out of (p, j) are all the pairs $((p, j), (q, h))$ such that $(q, h) \in X_j$. One can actually identify G with a multiset extension of G', where the set of multisets is $\{\bigcup_{1\leq j\leq w_p}(p, j) \mid p \in Q\}$ that we identify to Q. Let $I' = \{(i, j) \mid 1 \leq j \leq w_i\}$. Let $w'_{(p,j)} = 1$ for all (p, j) with $p \in Q$ and $1 \leq j \leq w_p$. Then, according to Proposition 10, the triple $(G', I', \mathbf{w}')$ recognizes the same sequence as $(G, i, \mathbf{w})$. Moreover, if $w_p = 1$ for all $p \in T$, let T' be set of all $(p, 1) \in Q'$ with $p \in T$. Then the triple (G', I', T') recognizes the same sequence as (G, i, T). If T is the set of vertices which have no outgoing edges, it is clear that the same holds for T'. ∎

We now come to our main result.

THEOREM 16. *Let $s = (s_n)_{n\geq 0}$ be a regular sequence of nonnegative integers, and let k be a positive integer such that $\sum_{n\geq 0} s_n k^{-n} \leq 1$. Then there is a k-ary rational tree having s as its generating sequence.*

Proof. Let us consider a regular sequence s and an integer k such that $\sum_{n\geq 0} s_n k^{-n} \leq 1$. Since the result holds trivially for $s(z) = 1$, we may suppose that $s_0 = 0$. Let (G, i, t) be a normalized representation of s, and let $\overline{G}$ be the closure of G as defined at the beginning of section 2.2. We denote by M (resp., $\overline{M}$) the adjacency matrix of G (resp., $\overline{G}$). Let $\overline{Q} = Q - \{t\}$ be the vertex set of $\overline{G}$. Let λ be the spectral radius of $\overline{M}$. By Proposition 8, the matrix $\overline{M}$ admits a positive k-approximate eigenvector $\overline{\mathbf{w}}$. By definition, we have $\overline{M}\overline{\mathbf{w}} \leq k\overline{\mathbf{w}}$.

Let $\mathbf{w}$ be the Q-vector defined by $w_q = \overline{w}_q$ for all $q \in \overline{Q}$ and $w_t = \overline{w}_i$. Then, since there is no edge going out of t in G, $\mathbf{w}$ is a positive k-approximate eigenvector of M. Let $\mathbf{t}$ be the Q-vector which is the characteristic vector of the vertex t. Let $m = w_i$.

By Theorem 12 there exists an admissible extension H of G with respect to $\mathbf{w}$ and m. Let U be the transfer matrix, and let $\mathbf{W} = U\mathbf{w}$. Since $w_t \equiv 0 \bmod m$, we

may choose H with the following additional property. For all $\mathbf{u} \in S$, either $u_t = 0$ or $\mathbf{u} = \mathbf{t}$.

According to Proposition 10, the sequence s is recognized by $(H, \mathbf{J}, \mathbf{X})$, where $\mathbf{J}$ is the characteristic row vector of $\mathbf{i}$ and $\mathbf{X}$ is the characteristic column vector of $\mathbf{t}$. This means that s is recognized by the normalized representation consisting of the graph H, the initial vertex $\mathbf{i}$ that we identify to i, and the terminal vertex $\mathbf{t}$ that we identify to t.

Let N be the adjacency matrix of H. By Proposition 14, the vector $\lceil \frac{\mathbf{W}}{m} \rceil$ is a positive k-approximate eigenvector of N. Remark that $\lceil \frac{\mathbf{W}}{m} \rceil_i = \lceil \frac{\mathbf{W}}{m} \rceil_t = 1$.

We may now apply Lemma 15 to construct a triple (H', I', T') equivalent to (H, i, t). The set T' is the set of leaves of H'. Since $\lceil \frac{\mathbf{W}}{m} \rceil_i = 1$, I' is reduced to one vertex i'. Since H' admits a k-approximate eigenvector with all components equal to one, the graph H' is of outdegree at most k. Finally, s is the generating sequence of the covering tree of H' starting at i'. This tree is k-ary and regular. □

Let us consider the above constructions in the particular case of the equality in Kraft's inequality. In this case, the result is a complete k-ary tree. Indeed, by Proposition 8, the matrix $\overline{M}$ admits a positive integral eigenvector $\mathbf{w}$ for the eigenvalue k. We have, for all $p \in \overline{Q}$,

$$\sum_{q \in \overline{Q}} M_{p,q} w_q = k w_p.$$

As a consequence, for any $\mathbf{u} \neq \mathbf{t}$, we have

$$\sum_{\mathbf{v} \in S} N_{\mathbf{u},\mathbf{v}} W_{\mathbf{v}} = k W_{\mathbf{u}}.$$

Then the graph constructed in Lemma 15 is of constant outdegree k. Thus the k-ary tree obtained is complete.

Let us consider the complexity of the construction used in the proof of Theorem 16. Let n be the number of vertices of the graph G giving a normalized representation of s. The size of the integer $m = w_i$ is exponential in n (see section 2.4). Thus the number of vertices of the graph H is bounded by a double exponential in n. The final regular tree is the covering tree of a graph whose set of vertices has the same size in order of magnitude.

Let, for example, s be the sequence defined by

$$s(z) = \frac{z^2}{(1 - z^2)} + \frac{z^2}{(1 - 5z^3)}.$$

Since $s(1/2) = 1$, it satisfies the Kraft equality for $k = 2$. The sequence s is recognized by (G, i, t), where $G = (Q, E)$ is the graph given in Figure 8 with $Q = \{1, 2, 3, 4, 5, 6, 7\}$, $i = 1$, $t = 4$. The adjacency matrix of G admits the 2-approximate eigenvector represented in Figure 8, where the coefficients of $\mathbf{w}$ are represented in squares beside the vertices. Thus $m = 3$.

An admissible extension H of G with respect to $\mathbf{w}$ and m is given in Figure 9. In this figure, each multiset of S is represented by a sequence of vertices with repetitions corresponding to the multiplicity. For example, the multiset $\mathbf{u} = (0, 0, 1, 0, 0, 2, 0)$ is represented by $(3, 6, 6)$. The sequence s is recognized by the normalized representation $(H, 1, 4)$, where the initial and final vertices are named as they appear in Figure 9. The coefficients of $\lceil \frac{\mathbf{W}}{m} \rceil$ are represented in squares beside the vertices.

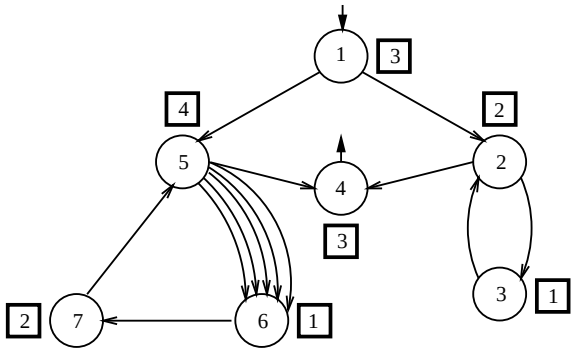


FIG. 8. *A normalized representation of s.*

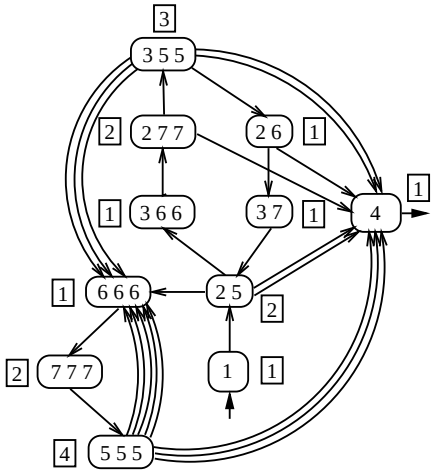


FIG. 9. *An admissible extension H.*

A regular binary tree T having s as a generating sequence of leaves is given in Figure 10. In this figure, the nodes have been renumbered, with the children of a node with a given label represented only once. The leaves of the tree are indicated by black boxes. The tree itself is obtained from the graph of Figure 9 by application of the construction of Lemma 15. For example, the vertex $(2, 5)$, which has coefficient 6 in $\mathbf{W}$, is split into two vertices named 2 and 3 in the tree.

This example was suggested to us by Christophe Reutenauer [18].

5. Generating sequence of nodes. In this section, we consider the generating sequence of the set of all nodes in a tree instead of just the set of leaves. This is motivated by the fact that in search trees, the information can either be carried by the leaves or by all the nodes of the tree. We will see that the complete characterization of the generating sequences of nodes in regular trees (Theorem 17) is more complicated than the one for leaves.

Soittola (see [19, p. 104]) has characterized the series which are the generating sequences of nodes in a regular tree. We characterize the ones that correspond to k-ary trees (Theorem 17). We also give a more direct construction in a particular case (Theorem 19).

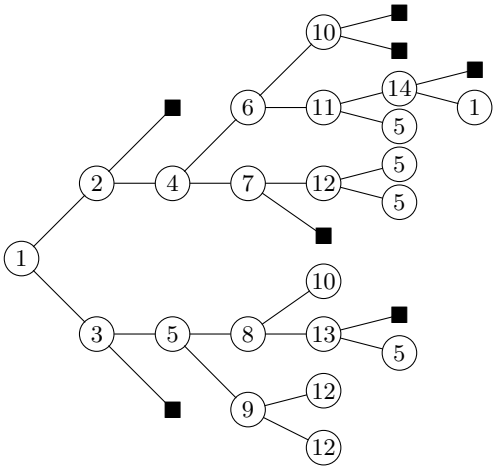


FIG. 10. *A regular binary tree with length distribution s.*

Let T be a tree. The generating sequence of nodes of the tree T is the sequence $t = (t_n)_{n\geq 0}$, where t_n is the number of nodes of T at height n. The sequence t satisfies $t_0 \leq 1$ and, moreover, if T is a k-ary tree, the condition

$$t_n \leq kt_{n-1}$$

for all $n \geq 1$. If T is a regular tree, then t is a regular sequence. We now completely characterize the regular sequences t that are the generating sequences of nodes of a k-ary regular tree.

THEOREM 17. *Let $t = (t_n)_{n\geq 0}$ be a regular sequence, and let k be a positive integer. The sequence $(t_n)_{n\geq 0}$ is the generating sequence of nodes of a k-ary regular tree iff it satisfies the following conditions.*

(i) *The convergence radius of t is strictly greater than $1/k$.*

(ii) *The sequence $s(z) = t(z)(kz-1)+1$ is regular.*

Proof. Let us first show that the conditions are necessary. Let $\overline{T}$ be the complete k-ary tree obtained by adding i new leaves to each node that has $k-i$ children. Since T is a regular tree, $\overline{T}$ is also regular.

Let s be the generating sequence of leaves of $\overline{T}$. Since $\overline{T}$ is complete, $s(1/k) = 1$. Since $kt_n = s_{n+1} + t_{n+1}$ for all $n \geq 0$, we have

$$1 - s(z) = t(z)(1 - kz).$$

Since s is a regular sequence, its radius of convergence is strictly larger than $1/k$ (see section 2.4). Since the value of the derivative of s at $z = 1/k$ is $kt(1/k)$, the same holds for t. This proves the necessity of the conditions.

Conversely, if t satisfies the conditions of the theorem, the regular series $s(z) = t(z)(kz-1)+1$ satisfies $s(1/k) = 1$. Thus, by Theorem 16, s is the generating sequence of leaves of a complete k-ary regular tree. The internal nodes of this tree form a k-ary regular tree whose generating sequence of nodes is t. □

The sequence s defined by condition (ii) is rational as soon as t is regular and therefore rational. Given a regular sequence t, condition (ii) is decidable in view of

a theorem of Soittola [19], also found independently in [13] and recalled below. We say that a rational sequence has a *dominating root*, either if it is a polynomial or if it has a real positive pole which is strictly smaller than the modulus of any other one. A sequence r is a *merge* of the sequences r_i if there is an integer p such that

$$r(z) = \sum_{i=0}^{p-1} z^i r_i(z^p).$$

THEOREM 18 (Soittola). *A sequence of nonnegative integers $r = (r_n)_{n\geq 0}$ is regular iff it is a merge of rational sequences having a dominating root.*

This result shows that it is decidable if a rational series is regular (see [19]). In the positive case, there is an algorithm computing a representation of the sequence.

We may observe that condition (ii) of the theorem implies the nonnegativity of the coefficients of the series s and thus the inequality for all $n \geq 1, t_n \leq kt_{n-1}$. It also implies that $t_0 \leq 1$.

We now show that there are regular sequences t satisfying $t_n \leq kt_{n-1}$ for all $n \geq 1$, and condition (i) of the theorem and such that the sequence $s(z) = t(z)(kz-1)+1$ is not regular. The example is based on an example of a rational sequence with nonnegative coefficients and which is not regular (see [10, p. 95]). Let

$$r_n = b^{2n}\cos^2(n\theta)$$

with $\cos(\theta) = \frac{a}{b}$, where the integers a, b are such that $b \neq 2a$ and $0 < a < b$. The sequence r is rational, has nonnegative integer coefficients, and is not regular. Its poles are $\frac{1}{b^2}$, $\frac{1}{b^2}e^{2i\theta}$, and $\frac{1}{b^2}e^{-2i\theta}$. We now define the sequence t as follows:

$$\begin{aligned} t_{2h} &= k^h, \\ t_{2h+1} &= k^h + r_h. \end{aligned}$$

We also assume that $b^2 < k$. By Soittola's theorem, the sequence t is regular since it is a merge of rational sequences having a dominating root. The convergence radius of t is $\frac{1}{\sqrt{k}} > \frac{1}{k}$. Therefore, the sequence t satisfies the first condition of Theorem 17. Let s be the sequence defined by $s(z) = t(z)(kz-1)+1$. If $h = 2p$ is even,

$$\begin{aligned} s_h &= kt_{h-1} - t_h \\ &= kk^{p-1} + kr_{p-1} - k^p + 1 = kr_{p-1} + 1. \end{aligned}$$

Thus the sequence s is not regular.

The above example does not work for the small values of k (the least value is $k = 10$). We do not know of similar examples for $2 \leq k \leq 9$.

We finally describe a particular case of Theorem 17 in which one has a relatively simple method, based on the multiset construction, to build the regular tree with a given generating sequence of nodes. This avoids the use of Soittola's characterization which leads to a method of higher complexity.

A *primitive* representation of a regular sequence s is a representation $(G, \mathbf{i}, \mathbf{t})$ such that the adjacency matrix of G is primitive. The following result is proved in [7] with a different proof using the state-splitting method of symbolic dynamics. The proof given here relies on a simpler construction.

THEOREM 19. *Let $t = (t_n)_{n\geq 0}$ be a regular sequence, and let k be a positive integer such that $t_0 = 1$, $t_n \leq kt_{n-1}$ for all $n \geq 1$, and such that*

(i) *the convergence radius of t is strictly greater than $1/k$, and*
(ii) *t has a primitive representation.*

Then $(t_n)_{n\geq 0}$ is the generating sequence of nodes by height of a k-ary regular tree.

We are going to give a proof of the theorem which uses the multiset construction. We shall use the following lemma that we establish first.

LEMMA 20. *Let M be a primitive matrix with spectral radius λ. Let $\mathbf{v}$ be a nonnull and nonnegative integral vector, and let k be an integer such that $\lambda < k$. Then there is a positive integer n such that $M^n\mathbf{v}$ is a positive k-approximate eigenvector of M.*

Proof. For a primitive matrix M with spectral radius λ, it is known that the sequence $((\frac{M}{\lambda})^n)_{n\geq 0}$ converges to $\mathbf{r}.\mathbf{l}$, where $\mathbf{r}$ is a positive right eigenvector and $\mathbf{l}$ is a positive left eigenvector of M for the eigenvalue λ with $\mathbf{l}\cdot\mathbf{r} = 1$ (see, for example, [15, p. 130]). Thus $(\frac{M^n}{\lambda^n}\mathbf{v})_{n\geq 0}$ converges to $\mathbf{r}.\mathbf{l}.\mathbf{v}$, which is equal to $\rho\mathbf{r}$, where ρ is a nonnegative real number. Since $M\mathbf{r} = \lambda\mathbf{r}$, we get, for a large enough integer n,

$$M\frac{M^n}{\lambda^n}\mathbf{v} \leq k\frac{M^n}{\lambda^n}\mathbf{v},$$

or, equivalently, $MM^n\mathbf{v} \leq kM^n\mathbf{v}$. If n is large enough, we moreover have $M^n\mathbf{v} > 0$ since M is primitive. □

We now give the proof of Theorem 19. It uses a shift of indices of the sequence to obtain a new sequence to which a simple application of the multiset construction can be applied.

Proof. Since t is regular, it is recognized by a triple $(G, \mathbf{i}, \mathbf{t})$, where $G = (Q, E)$ is a finite graph. Let M be the adjacency matrix of G.

For each $n \geq 0$, we have

$$t_n = \mathbf{i}M^n\mathbf{t}.$$

We denote by λ the spectral radius of M. By Proposition 6 the positive real number $1/\lambda$ is the radius of convergence of t. Thus $\lambda < k$ by hypothesis (i). Since M is a primitive matrix, by Lemma 20, there exists a positive integer n_0 such that $M^{n_0}\mathbf{t}$ is a positive k-approximate eigenvector of M.

Let $\mathbf{w} = M^{n_0}\mathbf{t}$, and let t' be the sequence defined by $t'_n = t_{n+n_0}$ for $n \geq 0$. Thus, for each $n \geq 0$,

$$t'_n = \mathbf{i}M^n\mathbf{w}.$$

The sequence t' is thus recognized by the triple $(G, \mathbf{i}, \mathbf{w})$. Note that $t'_0 = \mathbf{i}\cdot\mathbf{w}$.

Let $H = (S, R)$ be the extension of G obtained by the multiset construction in the following way. When we reach a vertex $\mathbf{u}$ of H, we partition $\mathbf{u}M$ in multisets $\mathbf{v}$ of degree 1, i.e., such that $\mathbf{v}$ is a 0, 1-vector with $v_q = 0$ for all $q \in Q$ except one. All elements of S are thus elements of Q except perhaps the initial vertex $\mathbf{i}$. If $\mathbf{i}$ is of degree 1, the number of elements of S is then equal to the number of elements of Q.

Let U be the transfer matrix of the extension. Since $\mathbf{w}$ is a positive k-approximate eigenvector of M, by Proposition 11, the vector $\mathbf{W} = U\mathbf{w}$ is a positive k-approximate eigenvector of the adjacency matrix of H. By Proposition 10, the triple $(H, \mathbf{i}, \mathbf{W})$ is equivalent to $(G, \mathbf{i}, \mathbf{w})$.

We now apply Lemma 15 to the graph H. We use $\mathbf{i}$ as initial vertex and the k-approximate eigenvector $\mathbf{W}$. Since we only use the first assertion of the lemma, we will not use any set T of terminal states. According to the lemma, we construct a graph H' and a set of vertices I' of H' such that H' admits the k-approximate vector

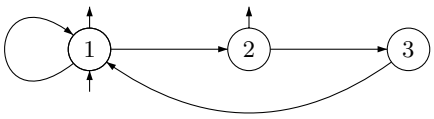


FIG. 11. *A primitive representation G of t.*

$\mathbf{W}'$ with all components equal to 1, and $(H', I', \mathbf{W}')$ is equivalent to $(H, \mathbf{i}, \mathbf{W})$. Thus H' is k-ary. Note that I' has $W_{\mathbf{i}} = \mathbf{i} \cdot \mathbf{w} = t_{n_0}$ elements.

Let T_p be the covering tree of H' starting at the state p of I'. Each T_p is a regular k-ary tree. Then t' is the sum of the generating sequences of nodes of the trees T_p for $p \in I'$.

Finally, we build a finite k-ary tree T' whose generating sequence of nodes is $(t_0, t_1, \ldots, t_{n_0})$. This can actually be done since $t_0 = 1$ and $t_n \leq kt_{n-1}$ for $n \geq 1$. We then identify bijectively each leaf at height n_0 of T' to the root of a tree T_j. We get a regular k-ary binary tree whose generating sequence of nodes is t. □

Let, for example, t be the series recognized by the graph G of Figure 11 with

$$\mathbf{i} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \text{ and } \mathbf{t} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}.$$

The adjacency matrix M of G is the primitive matrix

$$M = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Its spectral radius is less than 2. The hypotheses of Theorem 19 are thus satisfied. We apply the method described above. We have

$$M^2\mathbf{t} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix} \text{ and } M^3\mathbf{t} = \begin{bmatrix} 3 \\ 2 \\ 2 \end{bmatrix}.$$

Since $M^3\mathbf{t} \leq 2M^2\mathbf{t}$, $M^2\mathbf{t}$ is an approximate eigenvector of M. We thus set $n_0 = 2$ and $\mathbf{w} = M^2\mathbf{t}$. The graph H is the same as the graph G of Figure 11. The vector $\mathbf{W}$ is thus

$$\mathbf{W} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}.$$

The graph H' is represented on the left side of Figure 12. We finally obtain the binary regular tree T represented on the right side of Figure 12. (The nodes of the tree have been renumbered.)

Acknowledgments. The authors would like to thank Jean Berstel, Christophe Reutenauer, and the anonymous referees for their help to improve the presentation of our paper.

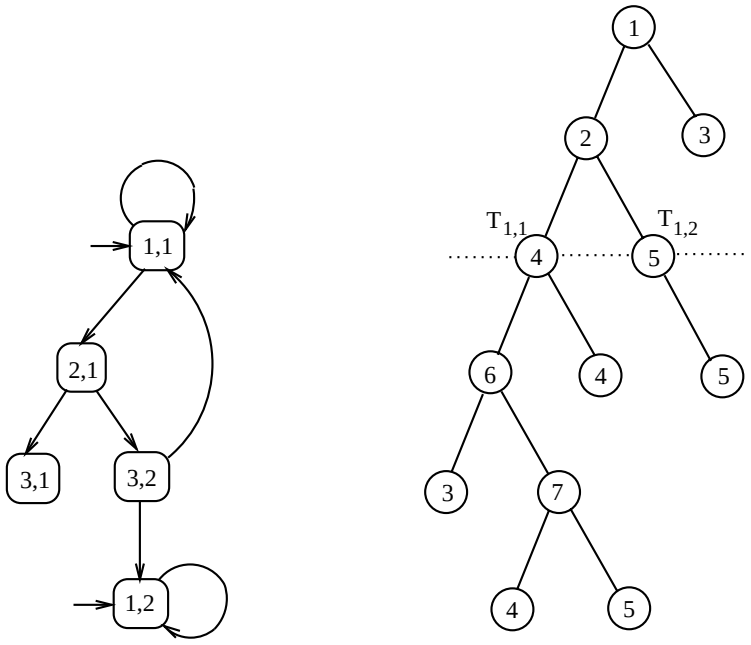


FIG. 12. *The graph H' and the tree T.*

REFERENCES

[1] R. L. ADLER, D. COPPERSMITH, AND M. HASSNER, *Algorithms for sliding block codes*, IEEE Trans. Inform. Theory, IT-29 (1983), pp. 5–22.

[2] M. AIGNER AND G. M. ZIEGLER, *Proofs from The Book*, Springer-Verlag, Berlin, 1998.

[3] R. B. ASH, *Information Theory*, Dover, New York, 1990.

[4] J. J. ASHLEY, *A linear bound for sliding-block decoder window size*, IEEE Trans. Inform. Theory, 3 (1988), pp. 389–399.

[5] F. BASSINO, M.-P. BÉAL, AND D. PERRIN, *Enumerative sequences of leaves in rational trees*, in Automata, Languages and Programming, Lecture Notes in Comput. Sci. 1256, Springer-Verlag, Berlin, 1997, pp. 76–86.

[6] F. BASSINO, M.-P. BÉAL, AND D. PERRIN, *Super-state automata and rational trees*, in LATIN'98, Lecture Notes in Comput. Sci. 1380, C. L. Lucchesi and A. V. Moura, eds., Springer-Verlag, Berlin, 1998, pp. 42–52.

[7] F. BASSINO, M.-P. BÉAL, AND D. PERRIN, *Enumerative sequences of leaves and nodes in rational trees*, Theoret. Comput. Sci., 221 (1999), pp. 41–60.

[8] F. BASSINO, M.-P. BÉAL, AND D. PERRIN, *Length distributions and regular sequences*, in Codes, Systems and Graphical Models, IMA Vol. Math. Appl., J. Rosenthal and B. Marcus, eds., Springer-Verlag, New York, 2000, to appear.

[9] M.-P. BÉAL, *Codage Symbolique*, Masson, Paris, 1993.

[10] J. BERSTEL AND C. REUTENAUER, *Rational Series and Their Languages*, Springer-Verlag, Berlin, New York, 1988.

[11] S. EILENBERG, *Automata, Languages and Machines*, A, Academic Press, New York, 1974.

[12] F. R. GANTMATCHER, *Matrix Theory*, Vol. II, Chelsea Publishing Company, New York, 1960.

[13] T. KATAYAMA, M. OKAMOTO, AND H. ENOMOTO, *Characterization of the structure-generating of regular sets and DOL growth functions*, Inform. Control, 36 (1978), pp. 85–101.

[14] B. P. KITCHENS, *Symbolic Dynamics. One-Sided, Two-Sided and Countable State Markov Shifts*, Springer-Verlag, Berlin, 1998.

[15] D. LIND AND B. H. MARCUS, *An Introduction to Symbolic Dynamics and Coding*, Cambridge University Press, Cambridge, UK, 1995.

[16] B. H. Marcus, *Factors and extensions of full shifts*, Monatsh. Math, 88 (1979), pp. 239–247.
[17] D. Perrin, *Finite automata*, in Handbook of Theoretical Computer Science, Vol. B, J. V. Leeuwen, ed., Elsevier, Amsterdam, MIT Press, Cambridge, MA, 1990, ch. 1.
[18] C. Reutenauer, *private communication*, Strasbourg University, Strasbourg, France, 1997.
[19] A. Salomaa and M. Soittola, *Automata-Theoretic Properties of Formal Power Series*, Springer-Verlag, New York, Heidelberg, 1978.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1231–1252

AN 8-APPROXIMATION ALGORITHM FOR THE SUBSET FEEDBACK VERTEX SET PROBLEM*

GUY EVEN†, JOSEPH (SEFFI) NAOR‡, AND LEONID ZOSIN§

Abstract. We present an 8-approximation algorithm for the problem of finding a minimum weight *subset feedback vertex set* (or SUBSET-FVS, in short). The input in this problem consists of an undirected graph $G = (V, E)$ with vertex weights $c(v)$ and a subset of vertices S called *special* vertices. A cycle is called *interesting* if it contains at least one special vertex. A subset of vertices is called a SUBSET-FVS with respect to S if it intersects every interesting cycle. The goal is to find a minimum weight SUBSET-FVS. The best previous algorithm for the general case provided only a logarithmic approximation factor. The minimum weight SUBSET-FVS problem generalizes two NP-complete problems: the minimum weight feedback vertex set problem in undirected graphs and the minimum weight multiway vertex cut problem. The main tool that we use in our algorithm and its analysis is a new version of multicommodity flow, which we call *relaxed multicommodity flow*. Relaxed multicommodity flow is a hybrid of multicommodity flow and multiterminal flow.

Key words. approximation algorithms, combinatorial optimization, feedback vertex set, multicommodity flow, multicut

AMS subject classifications. 05C85, 68R10, 68Q20, 68Q25, 68Q35, 90C05, 94C15, 68W25

PII. S0097539798340047

1. Introduction.

We consider in this paper the problem of finding a minimum weight *subset feedback vertex set* (or SUBSET-FVS, in short). The input in this problem consists of an undirected graph $G = (V, E)$ with vertex weights $c(v)$ and a subset of vertices S called *special* vertices. A cycle is called *interesting* if it contains at least one special vertex. A subset of vertices is called a SUBSET-FVS with respect to S if it intersects every interesting cycle. The goal is to find a minimum weight SUBSET-FVS.

The minimum weight SUBSET-FVS problem generalizes two NP-complete problems: When $S = V$, this is simply the minimum weight feedback vertex set problem in undirected graphs [16, 9], and when S contains a single special vertex, the SUBSET-FVS problem is equivalent to the minimum weight vertex multiway cut problem [5]. In the multiway cut problem, the input consists of a weighted graph and a set T of terminals, and the goal is to disconnect the terminals from each other by removing a set of vertices of minimum weight. The multiway cut problem is reduced to the SUBSET-FVS problem by adding a special (infinite-weight) vertex to the graph and connecting it to all the terminals. Conversely, the SUBSET-FVS problem with a single special vertex is reduced to a multiway cut problem by defining all the neighbors of the special vertex as terminals and removing the special vertex. Thus, the SUBSET-FVS problem

*Received by the editors June 9, 1998; accepted for publication (in revised form) January 5, 2000; published electronically October 11, 2000. A preliminary version of this paper appeared in *Proceedings of the 37th IEEE Conference on Foundations of Computer Science*, Burlington, VT, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 310–319.

http://www.siam.org/journals/sicomp/30-4/34004.html

†Department of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel (guy@eng.tau.ac.il).

‡Computer Science Department, Technion, Haifa 32000, Israel (naor@cs.technion.ac.il). The research of this author was supported in part by grant 92-00225 from the United States–Israel Binational Science Foundation (BSF), Jerusalem, Israel.

§ITG, Inc., 44 Farnsworth Street, 9th Floor, Boston, MA 02210 (lzosin@itginc.com). Most of this work was done while this author was a doctoral student at the Computer Science Department, Technion, Haifa 32000, Israel.

defines a full spectrum of feedback set problems which are categorized by the number of special vertices. Another motivation for the SUBSET-FVS problem is that heuristics for genetic linkage analysis require solving instances of the SUBSET-FVS problem [8].

Our main result in this paper is an algorithm that approximates the SUBSET-FVS problem by a factor of 8. This improves on the approximation algorithm for the SUBSET-FVS problem presented in [8] that achieves a factor of $\min\{\log|S|, \log\tau^*, 2\Delta\}$, where τ^* denotes the value of an optimal fractional solution, and Δ is the maximum degree. Our result almost matches the 2-approximation algorithms known for the vertex multiway cut problem [10] and for the feedback vertex problem [3, 2]. We remark that our algorithm requires solving linear programs exactly, and in this sense it is not a combinatorial algorithm.

The main tool that we use in our algorithm is a new version of multicommodity flow which we call *relaxed multicommodity flow*. Relaxed multicommodity flow is a hybrid of multicommodity flow and multiterminal flow. In multicommodity flow, for each edge, the capacity constraints apply to the total flow of all the commodities. In multiterminal flow, for each edge, the capacity constraints only apply to each commodity separately.

In relaxed multicommodity flow, there are two types of capacity constraints. First, the *intracommodity* constraints are single commodity capacity constraints that apply to each commodity separately. Second, the *intercommodity* constraints are constraints on the joint flow of the commodities. The intercommodity constraints used here are defined as follows: For every edge we consider the maximum flow, among all the commodities, which is shipped along it; we require that for every vertex, the sum of the maximum flows shipped along the edges incident to it is bounded by four times the weight of the vertex.

Even et al. [8] formulate the subset feedback set problem as an integer linear program and show that the gap between an optimal integral solution and an optimal (fractional) solution to the linear relaxation of this program can be as large as $\Omega(\log|S|)$. This means that an approximation algorithm which is based on rounding a fractional solution to an integral solution cannot achieve a constant approximation factor. Since the linear program for the subset feedback set problem is a "covering" problem, the dual problem is a "packing" problem, which in fact can be interpreted as a multicommodity flow problem. In the context of approximation algorithms for covering problems, the dual packing program usually serves as a lower bound on the optimal integral solution, and the approximation factor is measured with respect to this lower bound. This motivates replacing (standard) multicommodity flow by relaxed multicommodity flow as the dual problem of the SUBSET-FVS problem. This relaxation allows us to increase the total amount of flow shipped in the graph. We show that this provides a better lower bound on the optimal solution, yielding a constant approximation factor.

Suppose that a demand vector specifies for each source-sink pair (commodity) the amount of flow that needs to be shipped from the source to the sink. We say that a flow function *realizes* a demand vector if it ships for each commodity the amount of flow specified by the demand vector. A *tight* cut in the context of a relaxed multicommodity flow problem is a multiset of vertices which are tight with respect to the intracommodity constraints, called *intrasaturated vertices*, and also a set of vertices for which the intercommodity constraints are tight, called *intersaturated vertices*.

Tight cuts play an important role in identifying an approximate SUBSET-FVS

in our algorithm. Clearly, the weight of the intrasaturated vertices in a tight cut can be at most the sum of the demands. Thus, the key notion in the analysis of the approximation algorithm is bounding the weight of the intersaturated vertices in a tight cut. Indeed, our main theorem (Theorem 4.4) bounds the weight of the vertices that are intersaturated in every realization of a demand vector by the sum of the demands. Loosely speaking, Theorem 4.4 functions as an approximate "min-cut max-flow" theorem and implies that the total weight of the saturated vertices (both intersaturated and intrasaturated) in the cut is at most twice the sum of the demands.

1.1. Related work. Minimum weight feedback sets and subset feedback sets in *directed* graphs were considered in several papers [18, 20, 17, 6, 7]. Note that in the directed case, the vertex and edge versions are equivalent. The best known approximation factor for the subset feedback set in the directed case is $O(\min\{\log \tau^* \log\log \tau^*, \log |S| \log\log |S|\})$, where τ^* denotes the value of an optimal fractional solution [6]. Factor two approximation algorithms for the feedback vertex set problem in undirected graphs were given by [3, 2]. An approximation algorithm with a factor of two for the subset feedback edge set problem was given by [8]. Recently, Goemans and Williamson [11] considered a generalization of the SUBSET-FVS problem in undirected planar graphs and obtained a 9/4-approximation algorithm for their problem.

The approximation algorithm for the SUBSET-FVS problem turns out to be much more complicated than the approximation algorithm for the subset feedback edge problem presented by [8]. This is in accordance with conventional wisdom that in undirected graphs, optimization problems on vertices are harder than their counterparts on edges. For example, recall that the multiway cut problem is a special case of the subset feedback set problem. For the edge version, there is a rather simple 2-approximation algorithm [5]. (Recently, better bounds were obtained by [4, 15].) For the vertex version, [10] gives a *noncombinatorial* 2-approximation algorithm which is based on the observation that there exists an optimal solution in half-integers for the linear program for multiway cuts.

Recently, Naor and Zosin [19] improved the approximation factor for the directed multiway cut problem from $2 \log k$ to 2 using the relaxed multicommodity flow technique. However, [19] uses a completely different relaxation.

Organization. In section 2, a Gomory–Hu transformation and simplifying assumptions are presented. In section 3, the algorithm is overviewed. In section 4, the main theorem providing a bound on the sum of the weights of the vertices that are intersaturated by every relaxed multicommodity flow that supplies the demand vector is proved. In section 5, the approximation algorithm is presented. In section 6, the approximation factor is proved.

2. Preliminaries. In this section we define a transformation which is used by our algorithm and provide some simplifying assumptions. We henceforth refer sometimes to the weight of a vertex as its *capacity*, depending on the context.

2.1. A Gomory–Hu transformation. Let $G = (V, E)$ be an undirected graph where both edges and vertices are capacitated. Let C be a minimum cut separating source x from sink y in G. The cut C may contain both edges and vertices. Let $E(C)$ denote the edges in C, and let $V(C)$ denote the vertices in C. The removal of C from the graph partitions it into two parts, X and Y, where $x \in X$ and $y \in Y$ as can be seen in Figure 2.1.

Let R_x be the graph obtained from G by condensing the vertices in Y into a

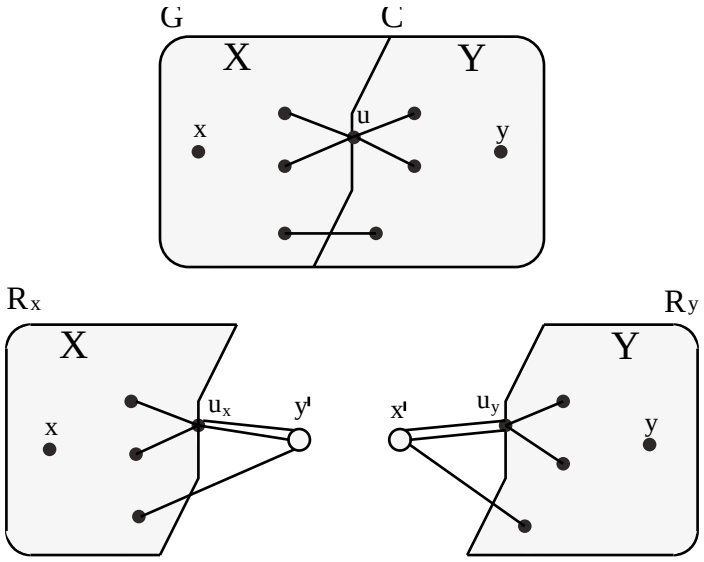


FIG. 2.1. *A Gomory–Hu transformation.*

single vertex denoted by y' which has infinite capacity. Let R_y be the graph obtained from G by condensing the vertices in X into a single vertex denoted by x' which has infinite capacity. Each vertex $u \in V(C)$ appears now in both R_x and R_y. We denote by u_x the copy of u in R_x, and by u_y the copy of u in R_y. We refer to u_x and u_y as *virtual copies* of u. We refer to the edges in R_x and R_y that correspond to edges in $E(C)$ or are adjacent to virtual copies of vertices in $V(C)$ as *virtual edges*. This construction is essentially the same as the condensation of a minimum cut performed by Gomory and Hu [12] (see also [14], and Granot and Hassin for the vertex version [13]).

The following two lemmas follow directly from [14, Lemma 5, p. 66; Lemma 6, p. 68] and from [13, Lemmas 4, 5, and 6].

LEMMA 2.1. *Let R_x and R_y denote the graphs obtained from applying a Gomory–Hu transformation to graph G with respect to minimum cut C between vertices x and y. Let $a \in X$ and $b \in Y$, and suppose that the capacity of C is at least d. Then, flow of value d can be shipped in G from a to b if and only if flow of value d can be shipped in R_x from a to y', and flow of value d can be shipped in R_y from b to x'.*

LEMMA 2.2 (under the same premises of Lemma 2.1). *Let $a, b \in X$ (or Y). Then, flow of value d can be shipped in R_x (or R_y) from a to b if and only if flow of value d can be shipped in G from a to b.*

2.2. Simplifying assumptions. For ease of presentation, we can assume without loss of generality the following assumptions.

(A) The set of finite weight vertices in the graph constitutes an independent set. This can easily be achieved by adding an infinite weight vertex on each edge.
(B) For each special vertex:
  (B1) It has infinite weight.
  (B2) It has degree two.
  (B3) Its two neighbors are not special vertices and have infinite weight.

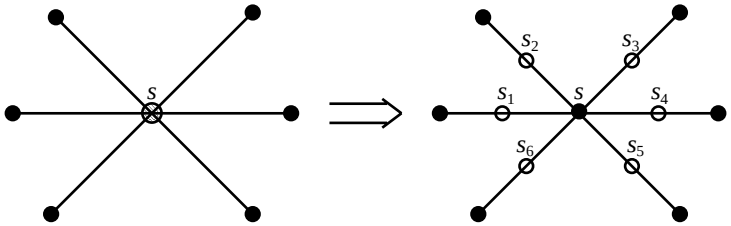


FIG. 2.2. *Transforming a special vertex with large degree.*

We justify Assumptions (B1), (B2), and (B3) as follows. Let s be a special vertex that does not satisfy (B1) or (B2). We insert a new vertex with infinite weight on each edge adjacent to s, add all new vertices to S, and delete s from S, as can be seen in Figure 2.2. Note that the new vertices are assigned infinite weight while vertex s retains its original weight.

Assumption (B3) can be satisfied by adding an infinite weight vertex on each edge in the graph. Thus, the set of special vertices constitutes an independent set, and each special vertex has degree two. Clearly, this transformation does not change the set of interesting cycles, nor does it change the optimal SUBSET-FVS.

Let $S = s_1, \ldots, s_k$ denote henceforth the set of special vertices. We denote the two neighbors of special vertex s_i for $1 \leq i \leq k$ by x_i and y_i.

3. Overview. In this section we overview the spirit of the algorithm and its proof. Our algorithm follows the basic framework suggested by [8] for approximating the SUBSET-FES problem. We first review the algorithm of [8].

Algorithm SUBSET-FES

Input: Graph $G = (V, E)$ with special vertices $s_1, \ldots, s_k$;
Output: SUBSET-FES M of G;

Notation: $V_i \triangleq V - \{s_i, \ldots, s_k\}$;
for $i = 1$ to k do
    $M_i \leftarrow$ minimum cut between x_i and y_i in the graph $G_i = (V_i, E - \cup_{j=1}^{i-1} M_j)$.
$M \leftarrow M_1 \cup \cdots \cup M_k$.

It is easy to see that the algorithm finds a feasible solution to the SUBSET-FES problem. Even et al. [8] show that the solution found by the algorithm approximates the optimal solution by a factor of 2. Algorithm SUBSET-FES can also be interpreted from a multicommodity flow perspective. Let each pair x_i, y_i, $1 \leq i \leq k$, define a commodity. The algorithm considers the commodities one-by-one and at each step maximizies the flow of the considered commodity. Notice that at the end of step i, $1 \leq i \leq k$, min-cut M_i is deleted from the graph and special vertex s_i is added back to the graph. In the end, we obtain a multicommodity flow function, where the flow of the different commodities is independent or nonaggregating.

The vertex version of Algorithm SUBSET-FES is not a good approximation algorithm for the SUBSET-FVS problem, as shown by the example depicted in Figure 3.1. To keep the drawing simple, the special vertices are not shown in Figure 3.1 (recall that special vertex s_i is connected to x_i and y_i). Assume that vertices $v_1, \ldots, v_k$ and $u_1, \ldots, u_k$ have unit weight. Let the weight of vertex w be $1+\epsilon$. The algorithm would

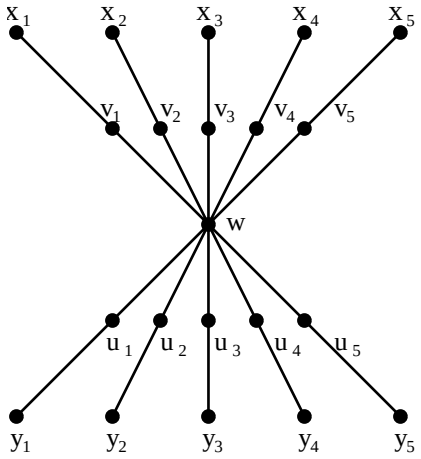


FIG. 3.1. *An example showing that the vertex version of Algorithm* SUBSET-FES *may only be a k-approximation for the* SUBSET-FVS *problem.*

proceed by assigning $M_i = \{v_i\}$, and the cost of the computed solution would be 5 (i.e., k, the number of special vertices), compared to the optimum solution that only contains vertex w.

Loosely speaking, we refine Algorithm SUBSET-FES by replacing the independence between the commodities by a relaxed multicommodity flow function. This means that in the ith iteration, the maximum flow (of value d_i) from x_i to y_i is computed such that $d^i = \{d_1, \ldots, d_i\}$ can be realized by a relaxed multicommodity flow function. A minimum cut (corresponding to the flow of the ith commodity) that disconnects x_i from y_i is computed with respect to the relaxed multicommodity flow function. The minimum cut contains vertices which are intersaturated by *every* relaxed multicommodity flow that realizes d^i, as well as vertices which are intrasaturated with respect to a commodity (one or more). The algorithm augments its solution by incrementally adding to the SUBSET-FVS the vertices belonging to the minimum cuts computed in each iteration.

It is crucial that the minimum cuts computed at each iteration are not deleted from the graph. Instead, a construction similar to the one used by Gomory and Hu [12] is employed with respect to the minimum cut computed at each iteration. This allows the flow which is computed in later iterations to either reuse a minimum cut computed at a previous iteration or go through the corresponding special vertex (but not both). We note that if we had deleted the minimum cuts from the graph (as in Algorithm SUBSET-FES), the value of the multicommodity flow function would have been much smaller than the weight of the minimum cuts $M_1, \ldots, M_k$.

The analysis rests on three key components. (a) The weight of the intersaturated vertices in a solution of a relaxed multicommodity flow problem is bounded by the sum of the demands (Theorem 4.4). (b) The relaxed flow which realizes $\boldsymbol{d}$ can be reconfigured so that the new source-sink pairs satisfy a separation property with respect to an optimal SUBSET-FVS (section 6.2). (c) The sum of the demands of a relaxed flow whose source-sink pairs satisfy a separation property with respect to a SUBSET-FVS is bounded by 4 times the weight of the SUBSET-FVS (Theorem 6.2).

The weight of the solution found by the algorithm equals the intersaturated vertices and intrasaturated vertices that are added to the solution. The weight of the intersaturated vertices, as well as the weight of the intrasaturated vertices, is bounded by the sum of the demands. The sum of the demands is then bounded by 4 times the weight of an optimal SUBSET-FVS using components (b) and (c), yielding an approximation factor of 8.

4. Relaxed multicommodity flow. In this section we define relaxed multicommodity flow which is a hybrid between multicommodity flow and multiterminal flow. In multicommodity flow, each commodity is subject to the capacity and preservation constraints, and the congestion constraints require that the sum of the flows of the different commodities do not exceed the edge/vertex capacities. In multiterminal flow, each commodity is subject to the capacity and preservation constraints, but there are no congestion constraints between them.

The source-sink pair of the ith commodity is denoted by a_i (source) and b_i (sink). We denote by $c(e)$ the capacity of edge e and by $c(v)$ the capacity of vertex v. We denote by $f_i(e)$ the amount of flow of commodity i on edge e. Denote by d_i, for $1 \le i \le k$, the amount of flow of commodity i shipped in the graph from a_i to b_i. Denote by $N(v)$ the set of edges adjacent to v, and for any $V' \subset V$, denote by $N(V')$ the set of edges which have at least one endpoint in V'. Denote by $f_{\max}(e)$ the value $\max_{1\le i\le k} f_i(e)$, i.e., $f_{\max}(e)$ equals the maximum flow along the edge e among all the commodities. To simplify notation, we also refer to the flow vector $(f_1, \ldots, f_k)$ as the flow f.

DEFINITION 4.1. *A multiterminal flow $f = (f_1, \ldots, f_k)$ between source-sink pairs $\{(a_i, b_i)\}_{i=1}^k$ is a* relaxed multicommodity flow *that realizes a demand vector $\boldsymbol{d} = (d_1, \ldots, d_k)$, if, for every $1 \le i \le k$, the flow supplied by f_i from a_i to b_i equals d_i, and the following constraints are satisfied.*

Intracommodity constraints. Each flow f_i is a single commodity flow. Namely,

(1) *for all $e \in E : f_i(e) \le c(e)$, and*

(2) *for all $v \in V : \sum_{e\in N(v)} f_i(e) \le 2 \cdot c(v)$.*

Intercommodity constraints. For all $v \in V : \sum_{e\in N(v)} f_{\max}(e) \le 4 \cdot c(v)$.

The intracommodity constraints are standard single commodity constraints. The sum of the flows of commodity i going through vertex v is bounded by $c(v)$. Note that in (2), each flow path passing through a nonterminal contributes twice its value to the left-hand side (LHS) and therefore the right-hand side (RHS) is $2 \cdot c(v)$. (As for sources and sinks, Assumption (B1) implies that they have infinite capacity.) The intercommodity constraints differ from (standard) multicommodity flow constraints in two respects. (i) For each edge e, the maximum, taken over all commodities on e, replaces the sum. (ii) The capacity of each vertex is multiplied by two and therefore the RHS is $4 \cdot c(v)$.

The relaxation of the intercommodity constraints is crucial for the success of the technique used in this paper in which flows of commodities are maximized one-by-one. The example depicted in Figure 4.1 shows that, without this relaxation, maximizing the flows of the commodities one-by-one could yield a total flow which is $1/(k-1)$ times smaller than the minimum cut that separates all source-sink pairs. Assume that vertices $v_1, v_2, \ldots, v_{k-1}$ have unit capacities. A unit flow between x_1 and y_1 would intersaturate the unit capacity vertices along the path, and no more flow could be pushed for the other commodities. This example shows that the intercommodity constraints need to be relaxed by a parameter α, where $\alpha > 1$. In section 6.4, we explain the choice $\alpha = 2$.

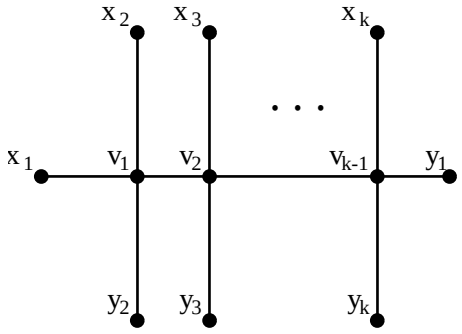


FIG. 4.1. *Relaxed multicommodity flow example.*

We define two types of *saturated* vertices with respect to f:

- Vertex $v \in V$ is called *intrasaturated* with respect to i if the intracommodity constraint corresponding to commodity i is tight with respect to v.
- Vertex $v \in V$ is called *intersaturated* if the intercommodity constraint with respect to v is tight.

Note that a vertex can be both intersaturated and intrasaturated with respect to more than one commodity.

Consider a network $N = (G, c)$, where G is a graph and c is a capacity function defined on the vertices and edges. We now define the notions of a *restricted* function $h : E \to R^{\geq 0}$ with respect to a network $N = (G, c)$ and the *restricted* network N^h with respect to a restricted function h.

DEFINITION 4.2. *Function h is called a restricted function for a network $N = (G, c)$ if*

(1) *for all $e \in E : h(e) \leq c(e)$, and*
(2) *for all $v \in V : \sum_{e \in N(v)} h(e) \leq 4 \cdot c(v)$.*

The restricted network N^h has the same vertex and edge set as N. The vertex capacities in N^h are the same as in N; the capacity of each edge e in N^h is set to $h(e)$.

Whenever it is clear which capacity function is attached to the edges of a network, we refer to the network as a graph. In particular, for simplicity we denote by G^f the restricted network N^f, where $N = (G, c)$.

The next claim follows directly from the previous definitions.

CLAIM 4.3. Let h be a restricted function for a graph G and suppose that for each $1 \leq i \leq k$, flow function f_i is a single commodity flow function in G^h that satisfies the capacity constraints. Then, $(f_1, \ldots, f_k)$ is a relaxed multicommodity flow in G.

4.1. Flow vs. intersaturated vertices. Consider source-sink pairs (a_i, b_i), $1 \leq i \leq k$, and a demand vector $\boldsymbol{d} = \{d_1, \ldots, d_k\}$. For simplicity of presentation, assume that all edges are uncapacitated, and there are no two finite weight vertices that are adjacent. Let I denote the set of intersaturated vertices that are saturated in *every* realization of $\boldsymbol{d}$. Note that in the case of single commodity flow, an edge that is saturated by every maximum flow function must belong to a minimum cut. We now state our main theorem which can be interpreted as an approximate "min-cut max-flow" theorem in the context of relaxed multicommodity flow.

THEOREM 4.4. *Suppose that a demand vector* $\boldsymbol{d} = \{d_1, \ldots, d_k\}$ *can be realized by a relaxed multicommodity flow function. Then,*

$$\sum_{v \in I} c(v) \leq \sum_{i=1}^{k-1} d_i.$$

The tightness of Theorem 4.4 follows from the example depicted in Figure 4.1. The relaxed multicommodity constraints enable one to push one unit of flow per commodity. Hence $\sum_{i=1}^{k-1} d_i = k-1$. The vertices $v_1, \ldots, v_{k-1}$ are all intersaturated by such a relaxed multicommodity flow, and hence $\sum_{v \in I} c(v) = k-1$, and tightness follows.

We need the following definition, claim, and lemma for proving the theorem.

DEFINITION 4.5. *A relaxed multicommodity flow function* f *is called simple if it has the following "minimal" property. For each commodity* i*, and for each edge* $e \in E$ *for which* $f_{\max}(e) = f_i(e)$*, if we decrease* $f_i(e)$ *by rerouting the flow (of possibly all commodities), then there exists an edge* $e' \in E$ *for which* $f_{\max}(e')$ *has to be increased.* Intuitively, a simple relaxed multicommodity flow function f does not contain spurious cycles for each commodity, and it "overlays" flow paths of different commodities as much as possible.

CLAIM 4.6. Let f be a relaxed multicommodity flow function that realizes demand vector $\boldsymbol{d}$. Then, there exists a simple relaxed multicommodity flow function f' that realizes demand vector $\boldsymbol{d}$ such that $f'_{\max}(e) \leq f_{\max}(e)$ for each edge $e \in E$.

Proof. Let g be a relaxed multicommodity flow function that realizes demand vector $\boldsymbol{d}$ and satisfies $g_{\max}(e) \leq f_{\max}(e)$ for all $e \in E$ such that it is impossible to decrease $g_{\max}(e)$ on some edge e without increasing $g_{\max}(e')$ on some other edge e'. We can obtain such a relaxed multicommodity flow function from f by subsequent minimization of $f_{\max}(e)$ on each edge $e \in E$ without increasing $f_{\max}(e')$ on some other edge e'.

Now we show that the following relaxed multicommodity flow function f' satisfies the claim. Flow function f' minimizes

$$\sum_{e \in E} |\{i | f'_i(e) = g_{\max}(e)\}|, \tag{4.1}$$

subject to the constraints that $f'_{\max}(e) \leq g_{\max}(e)$, and f' realizes demand vector $\boldsymbol{d}$.

We need only to show that f' is simple. Suppose that it is not simple. Then, there exists an edge e and commodity i for which $f'_{\max}(e) = f'_i(e)$ such that we can decrease $f'_i(e)$ by rerouting the flow without increasing $f'_{\max}(e')$ for any other edge $e' \in E$. Let f'' be the rerouted flow function, and let h denote the flow function which is the average (taken over each commodity) of f' and f''.

We claim that since h is the average of f' and f'', we have that for all edges $e' \in E$, $\{i | h_i(e') = g_{\max}(e')\} \subseteq \{i | f'_i(e') = g_{\max}(e')\}$. We also have that for edge e, $\{i | h_i(e) = g_{\max}(e)\} \subset \{i | f'_i(e) = g_{\max}(e)\}$. Clearly, h contradicts the objective function that f' minimizes. □

LEMMA 4.7. *Suppose that a simple relaxed multicommodity flow function* f *realizes demand vector* $\boldsymbol{d} = \{d_1, \ldots, d_k\}$*. Let* $V' \subseteq V$ *contain only finite weight vertices. (By Assumption* (A)*,* V' *is an independent set.) Then, there exists a relaxed multicommodity flow function* g *that also realizes demand vector* $\boldsymbol{d}$ *satisfying that for each*

edge $e \in E$, $g_{\max}(e) \leq f_{\max}(e)$, *and*

$$\sum_{e \in N(V')} g_{\max}(e) - 2 \cdot \sum_{v \in V'} c(v) \leq 2 \cdot \sum_{i=1}^{k-1} d_i. \tag{4.2}$$

Proof. The proof is by induction on k. For $k = 1$, f is a single commodity flow function, and thus the RHS of (4.2) is 0. The LHS of (4.2) is not positive because of the intracommodity constraints. We assume the induction hypothesis holds for $k-1$, and we prove it for k. Without loss of generality, let d_1 denote the minimum demand in $\boldsymbol{d}$. Let V'' contain all vertices $v \in V'$ for which there exists an edge $e \in N(v)$ such that $f_1(e) < f_{\max}(e)$. We show that it suffices to prove the induction hypothesis for vertices in V'' only. Let g be a relaxed multicommodity flow function that satisfies the induction hypothesis (or the lemma) for V''. Since $g_{\max}(e) \leq f_{\max}(e)$ for all $e \in E$, we have that $g_{\max}(e) \leq f_{\max}(e) = f_1(e)$ for all $e \in N(V' - V'')$. By the intracommodity constraint for commodity 1, for all $v \in V$, $\sum_{e \in N(v)} f_1(e) \leq 2 \cdot c(v)$. Therefore, g satisfies

$$\sum_{e \in N(V' - V'')} g_{\max}(e) - 2 \cdot \sum_{v \in V' - V''} c(v) \leq 0,$$

implying that g satisfies the induction hypothesis for V' too.

An edge $e = (u, v)$, where $c(v)$ is finite and $v \in V''$ is called *critical* with respect to commodity 1 if: (i) $f_{\max}(e) = f_1(e)$; (ii) $f_{\max}(e) > f_i(e)$ for all $i > 1$.

The *easy case* is when each flow path of commodity 1 contains at most two critical edges. In this case, remove the flow of commodity 1 from the graph. The relaxed multicommodity flow $(f_2, \ldots, f_k)$ is not necessarily simple. Let f' be a simple relaxed multicommodity flow function that realizes demand vector $\{d_2, \ldots, d_k\}$ and satisfies that for each edge $e \in E$, $f'_{\max}(e) \leq f_{\max}(e)$. By Claim 4.6, flow function f' exists.

The induction hypothesis states that there exists a relaxed multicommodity flow function g' that satisfies demand vector $\{d_2, \ldots, d_k\}$ such that for all $e \in E$, $g'_{\max}(e) \leq f'_{\max}(e)$. Flow function g' also satisfies, for V'',

$$\sum_{e \in N(V'')} g'_{\max}(e) - 2 \cdot \sum_{v \in V''} c(v) \leq 2 \cdot \sum_{i=2}^{k-1} d_i. \tag{4.3}$$

Now extend g' to g by adding commodity 1 back to the graph, i.e., $g_1(e) = f_1(e)$ for all $e \in E$. We claim that $g_{\max}(e) > g'_{\max}(e)$ only if e is critical with respect to commodity 1: if $g_{\max}(e) > g'_{\max}(e)$, then $f_1(e) = g_1(e) > g'_i(e)$. Since e is not critical, there exists a commodity $i > 1$ such that

$$f_i(e) = f_{\max}(e) \geq f_1(e).$$

Hence, $f_i(e) > g'_i(e) = g_i(e)$. Thus, the construction of g contradicts the fact that f is simple, yielding that if $g_{\max}(e) > g'_{\max}(e)$, then e is critical, as claimed.

The effect of extending g' to g in (4.3) is as follows. The RHS increases by $2d_1$. The LHS increases due to the critical edges: we change $g'_{\max}(e)$ to $f_1(e) = g_1(e)$ in $\sum_{e \in N(V'')} g'_{\max}(e)$ only in terms corresponding to critical edges. Since every flow path of commodity 1 contains at most two critical edges, the increase of the LHS is also bounded by $2d_1$.

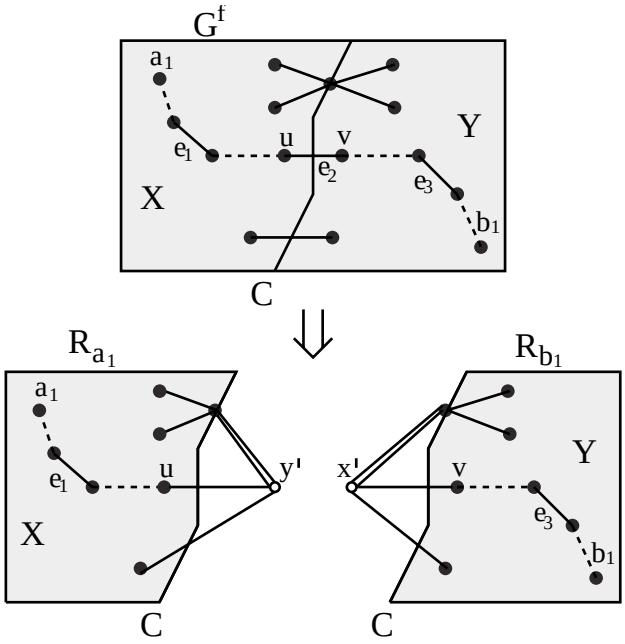


FIG. 4.2. *Construction of R_{a_1} and R_{b_1} from $G^{f_{\max}}$.*

However, the RHS of (4.3) is increased by $2d_1$. This completes the proof of the induction hypothesis for k in this case.

The *harder case* is when there exists a flow path of commodity 1 containing at least three critical edges, e_1, $e_2 = (u, v)$, and e_3, appearing in this order along the path from a_1 to b_1. By Definition 4.2, we may interpret $f_{\max}$ also as a restricted function for G. Construct the restricted graph $G^{f_{\max}}$. Commodity f_1 is a standard single commodity flow in $G^{f_{\max}}$. We claim that in $G^{f_{\max}}$, any standard single commodity flow function that ships d_1 units of flow from a_1 to b_1 must saturate edge e_2. Otherwise, in $G^{f_{\max}}$ there exists a single commodity flow f_1' that ships d_1 units of flow from a_1 to b_1 and $f_1'(e_2) < f_1(e_2)$. Therefore, by Claim 4.3, there exists in G a relaxed multicommodity flow $f' = (f_1', f_2, \ldots, f_k)$ that contradicts the fact that f is simple. Therefore, in $G^{f_{\max}}$ there exists a minimum cut C separating u from v (the endpoints of e_2) which is saturated by flow f_1. Note that cut C may contain both edges and vertices. The removal of C from the graph partitions it into two parts, X and Y, where $u \in X$ and $v \in Y$. Since cut C is saturated by f_1, it also separates a_1 from b_1 in $G^{f_{\max}}$. Apply now the Gomory–Hu transformation (see section 2.1) to cut C and graph $G^{f_{\max}}$. We obtain two graphs, R_{a_1} with condensed vertex y' and R_{b_1} with condensed vertex x', as depicted in Figure 4.2. We assign capacity to edges $V(C) \times \{y'\}$ and $V(C) \times \{x'\}$ equal to the flow of f_1 along these edges, and zero capacity to edges in $V(C) \times V(C)$.

Before we can apply the induction hypothesis to R_{a_1} and R_{b_1}, we prove the following claim.

CLAIM 4.8.

(a) Each source-sink pair (for $i > 1$) must belong to either R_{a_1} or R_{b_1}.

(b) Each of the graphs R_{a_1} and R_{b_1} must contain at least one source-sink pair (for $i > 1$).

Proof of Claim 4.8. We first prove that for each commodity $i > 1$ we have that

either $a_i, b_i \in R_{a_1}$, or $a_i, b_i \in R_{b_1}$. Otherwise, suppose $a_i \in R_{a_1}$ and $b_i \in R_{b_1}$. Since $f_1(e_2) > f_i(e_2)$ and cut C is saturated by commodity 1, we get that $d_1 > d_i$, which contradicts the assumption that d_1 is the smallest demand.

We prove that each side must contain a source-sink pair. Assume without loss of generality that source-sink pairs (a_i, b_i) for $2 \le i \le t$ are in R_{a_1}, and source-sink pairs (a_i, b_i) for $t+1 \le i \le k$ are in R_{b_1}. We first claim that from Lemma 2.2, there exists in R_{a_1} a relaxed multicommodity flow function f^x that satisfies demand vector $\{d_1, d_2, \ldots, d_t\}$, where the source and sink of commodity 1 are a_1 and y'. Similarly, in R_{b_1} there exists a relaxed multicommodity flow function f^y that satisfies demand vector $\{d_1, d_{t+1}, \ldots, d_k\}$, where the source and sink of commodity 1 are x' and b_1. We note that for all edges $e \in R_{a_1}$, $f^x_{\max}(e) \le f_{\max}(e)$, and for all edges $e \in R_{b_1}$, $f^y_{\max}(e) \le f_{\max}(e)$.

We now show that $1 < t < k$. Let z_1 and z_2 denote the endpoints of e_1; suppose that z_1 has finite weight. By the definition of critical edges, it follows that $z_1 \in V''$. Hence, there exists an edge $e' \in N(z_1)$ and commodity $j > 1$ such that $f_{\max}(e') = f_j(e') > f_1(e')$. We show that the source-sink pair $a_j, b_j \in R_{a_1}$. If $a_j, b_j \in R_{b_1}$, then some of the flow paths of commodity j pass through cut C. In this case, we contradict the simplicity of f by rerouting flow as follows. For each flow path P_j of commodity j that passes through the cut C, we can take two flow paths P_1 and P_1' of commodity 1 that pass through the same edges or vertices of the cut C that P_j uses. Now, we can reroute each flow path P_j which is crossing the cut C in the following way. It will go from the cut C to a_1 using path P_1 and continue from a_1 to the cut C using path P_1'. Thus, we can reroute commodity j in R_{a_1} using the flow paths of commodity 1, yielding that $f_j(e') \le f_1(e')$. Therefore, $f_j(e')$ is decreased without increasing $f_{\max}(e'')$ for any other edge e''. This contradicts the fact that f is simple and hence $a_j, b_j \in R_{a_1}$ as claimed. Similarly, using critical edge e_3, we can prove that there exists a commodity j' such that $a_{j'}, b_{j'} \in R_{b_1}$. (Note that the fact that each side of the cut must contain at least one terminal pair implies that in the special case of two commodities the hard case does not occur.)

Now we can apply the induction hypothesis to graphs R_{a_1} and R_{b_1} and vertex sets $V'' \cap R_{a_1}$ and $V'' \cap R_{b_1}$ with respect to simple flow functions obtained from f^x and f^y. (Denote by $V(C)$ the vertices in C; note that each vertex $z \in V(C)$ appears in both R_{a_1} and R_{b_1}.) Denote by g^x the flow function in R_{a_1} that exists by the induction hypothesis and for which $g^x_{\max}(e) \le f^x_{\max}(e)$ for all $e \in R_{a_1}$. We get from the induction hypothesis that

$$\sum_{e \in N(V(C) \cap V'')} g^x_{\max}(e) + \sum_{e \in N(X \cap V'')} g^x_{\max}(e) - 2 \cdot \sum_{v \in V(C) \cap V''} c(v) - 2 \cdot \sum_{v \in X \cap V''} c(v) \le 2 \cdot \sum_{i=1}^{t-1} d_i. \tag{4.4}$$

Denote by g^y the flow function in R_{b_1} that exists by the induction hypothesis and for which $g^y_{\max}(e) \le f^y_{\max}(e)$ for all $e \in R_{b_1}$. We now get from the induction hypothesis that

$$\sum_{e \in N(V(C) \cap V'')} g^y_{\max}(e) + \sum_{e \in N(Y \cap V'')} g^y_{\max}(e) - 2 \cdot \sum_{v \in V(C) \cap V''} c(v) - 2 \cdot \sum_{v \in Y \cap V''} c(v) \le 2 \cdot \left(\sum_{i=t+1}^{k-1} d_i + d_1 \right). \tag{4.5}$$

By rerouting flow, similarly to the rerouting of flow in the proof of Claim 4.8, we can obtain, without loss of generality, that

(a) neither g^x nor g^y deliver flow along edges in $V(C) \times V(C)$;
(b) if terminal pair (a_i, b_i) is in X, and $e \in V(C) \times \{y'\}$, then $g_1^x(e) \geq g_i^x(e)$. A similar property holds for a terminal pair in Y and an edge in $V(C) \times \{x'\}$.

We now form a relaxed multicommodity flow function g in graph G by gluing together g^x and g^y. For commodity 1, $g_1(e) = g_1^x(e)$ for all $e \in R_{a_1} \cap G$, $g_1(e) = g_1^y(e)$ for all $e \in R_{b_1} \cap G$, and $g_1(e) = g_1^x(e)$ for all edges in G corresponding to virtual edges in R_{a_1} and R_{b_1}. Recall that virtual edges correspond to edges in the minimum cut C, and hence, for every virtual edge e, $g_1^x(e) = g_1^y(e)$. Consider a source-sink pair (a_j, b_j), $j > 1$. Suppose that $a_j, b_j \in R_{a_1}$. By Lemma 2.2 and by rerouting flow, we may assume that for every edge $e \in R_{b_1}$, $g_j(e) \leq g_1^y(e)$. Moreover, the construction of g_j implies that $g_j(e) = g_j^x(e)$ for every $e \in R_{a_1}$. The analogous property holds when a source-sink pair is in R_{b_1}.

We now apply Claim 4.3 to the flow functions $g_1, \ldots, g_k$ in graph G to obtain relaxed multicommodity flow function g such that it realizes demands $\{d_1, \ldots, d_k\}$, and it satisfies $g_{\max}(e) \leq g_{\max}^x(e)$ if $e \in R_{a_1}$, and $g_{\max}(e) \leq g_{\max}^y(e)$ if $e \in R_{b_1}$.

Divide the edges $N(V(C) \cap V'')$ into groups:

Group (a): edges incident to y' or x';

Group (b_x): edges incident to vertices in X;

Group (b_y): edges incident to vertices in Y;

Group (c): edges in $V(C) \times V(C)$.

Our assumption on g^x implies that edges of group (a) in X are saturated by g_1^x. Therefore, for every vertex $v \in V(C)$,

$$\sum_{e \in N(v) \cap \text{ group(a)}} g_{\max}^x(e) = c(v).$$

In addition, our assumption on g^x implies that the flow of g^x along edges of group (c) is zero. Therefore,

$$\sum_{e \in N(V(C) \cap V'')} g_{\max}^x(e) = \sum_{v \in V(C) \cap V''} c(v) + \sum_{e \in \text{ group}(\mathrm{b}_x)} g_{\max}^x(e). \tag{4.6}$$

The same applies for the other side of the cut. Therefore,

$$\sum_{e \in N(V(C) \cap V'')} g_{\max}^y(e) = \sum_{v \in V(C) \cap V''} c(v) + \sum_{e \in \text{ group}(\mathrm{b}_y)} g_{\max}^y(e). \tag{4.7}$$

Summing up inequalities (4.4) and (4.5), substituting summations of g^x and g^y with the RHSs of (4.6) and (4.7), and replacing g^x and g^y by g, we get that

$$\begin{aligned} 2 \cdot \sum_{v \in V(C) \cap V''} c(v) + \sum_{e \in N(V'')} g_{\max}(e) - 2 \cdot \sum_{v \in V''} c(v) - 2 \cdot \sum_{v \in V(C) \cap V''} c(v) \\ \leq 2 \cdot \sum_{i=1}^{k-1} d_i + 2 \cdot (d_1 - d_t). \end{aligned}$$

Therefore, we get that

$$\sum_{e \in N(V'')} g_{\max}(e) - 2 \cdot \sum_{v \in V''} c(v) \leq 2 \cdot \sum_{i=1}^{k-1} d_i,$$

proving the lemma. □

Proof of Theorem 4.4. By Lemma 4.7 there exists a flow function g such that

$$\sum_{e \in N(I)} g_{\max}(e) - 2 \cdot \sum_{v \in I} c(v) \leq 2 \cdot \sum_{i=1}^{k-1} d_i,$$

where V' is chosen to be I. For each vertex $v \in I$, $\sum_{e \in N(v)} g_{\max}(e) = 4 \cdot c(v)$ by the intercommodity constraints. Substituting in the above yields the theorem. □

Remark 1. Let α denote the factor by which the intercommodity constraints are relaxed. (In this paper, $\alpha = 2$.) Then, the bound in Theorem 4.4 can be generalized to

$$\sum_{v \in I} c(v) \leq \frac{1}{\alpha - 1} \cdot \sum_{i=1}^{k-1} d_i.$$

5. The approximation algorithm. In this section we describe a primal-dual algorithm for approximating the SUBSET-FVS problem. The algorithm computes a relaxed multicommodity flow in the graph, iteratively, for k iterations. The algorithm constructs a sequence of graphs $H_0, H_1, \ldots, H_k$, where H_i is constructed in the end of the ith iteration. Initially, we start with a graph H_0 which is obtained from G by removing all special vertices from it. All edges in H_0 are assigned infinite capacity. The graph H_i is constructed from H_{i-1} by a Gomory–Hu transformation based on the minimum cut that is computed in the ith iteration. Note that this transformation replaces the source-sink pair x_i, y_i with a new source-sink pair denoted by x'_i, y'_i. Moreover, capacities are assigned to virtual edges in H_i as described later.

We now describe iteration $i+1$ for $0 \leq i \leq k-1$ of the algorithm. The input is graph H_i and demand vector $d^i = \{d_1, \ldots, d_i\}$ which was realized in H_{i-1} in the ith iteration.

1. Compute a relaxed multicommodity flow function $f = (f_1, \ldots, f_{i+1})$ in H_i that
   (i) realizes demand vector d^i for commodities $1, \ldots, i$, for source-sink pairs x'_j, y'_j, $1 \leq j \leq i$ and
   (ii) maximizes d_{i+1}, the amount of flow of commodity $i+1$ shipped from x_{i+1} to y_{i+1}.
2. Compute a minimum cut C and deduce from it a subset of vertices $F_{i+1} = A_{i+1} \cup B_{i+1}$, where A_{i+1} is a subset of the vertices intrasaturated by f_{i+1}, and B_{i+1} is a subset of the vertices intersaturated by f.
3. Derive graph H_{i+1} from graph H_i by applying the Gomory–Hu construction to minimum cut C between x_{i+1} and y_{i+1}, and by adding back special vertex s_{i+1}.

The algorithm returns as a SUBSET-FVS the set $F \triangleq F_1 \cup \cdots \cup F_k$.

We now elaborate on the steps of the algorithm. In step 1, we compute a relaxed multicommodity flow function f that realizes demand vector d^i and maximizes d_{i+1}, the amount of flow of commodity $i+1$. The flow function f can be computed using a linear program. The objective function maximizes d_{i+1}. The constraints of the linear program are standard and include constraints for encoding the intercommodity requirements. These constraints use a new variable $f_{\max}(e)$ for every edge and are defined as follows. For each vertex $v \in V$ for which $c(v)$ is finite, we have

$$\forall\, e \in N(v),\ \ \forall\, j,\ 1 \le j \le (i+1):\ \ f_j(e) \le f_{\max}(e),$$
$$\forall v:\ \ \sum_{e \in N(v)} f_{\max}(e) \le 4 \cdot c(v).$$

We are interested in identifying the set of vertices in H_i that are intersaturated by every relaxed multicommodity flow function that realizes demand vector $\{d_1, \ldots, d_{i+1}\}$. Let this set be denoted by I_{i+1}. The vertices belonging to I_{i+1} can be identified as follows. For each vertex $v \in V$ where $c(v)$ is finite, independently solve the linear program that minimizes $\sum_{e \in N(v)} f^v_{\max}(e)$ subject to the constraints that f^v is a legal relaxed multicommodity flow function that realizes demand vector $\{d_1, \ldots, d_{i+1}\}$. Clearly, if v is intersaturated in f^v, then $v \in I_{i+1}$. By taking the flow function f to be the average over all flows $\{f^v\}_{v \in V}$, we get that f intersaturates only vertices in I_{i+1}.

We now elaborate on step 2 in the algorithm. Consider graph H_i with a single source-sink pair, edge capacities, and vertex weights. We note that edge capacities in H_i (which will be described in step 3) are defined during the construction of $H_0, \ldots, H_{i-1}$, and only virtual edges have been assigned finite capacity (vertices maintain their original capacities). Our goal is to compute a minimum cut with respect to the source-sink pair x_{i+1}, y_{i+1}. We define additional temporary capacities for the sake of computing the minimum cut as follows: For every vertex v in I_{i+1} and edge $e \in N(v)$, define $c(e) = f_{\max}(e)$. Since d_{i+1} is maximum, it follows that f_{i+1} is a max-flow between x_{i+1} and y_{i+1} in H_i. Let C denote a minimum cut between x_{i+1} and y_{i+1} in H_i. The cut C consists of vertices, denoted by $V(C)$, and of edges, denoted by $E(C)$. Vertices in $V(C)$ must be intrasaturated by f_{i+1}. Define $A_{i+1} = V(C)$. Edges in $E(C)$ must be either virtual or incident to a vertex in I_{i+1}. Virtual edges in $E(C)$ contribute nothing to B_{i+1} (or F_{i+1}). However, for every nonvirtual edge in $E(C)$, we add its endpoint which belongs to I_{i+1} to the set B_{i+1}, unless this vertex is already in A_{i+1}. Thus, A_{i+1} consists of vertices that are intrasaturated by f_{i+1}, B_{i+1} consists of vertices that are intersaturated by $f = (f_1, \ldots, f_{i+1})$, and these sets are disjoint. The set F_{i+1} is the union of these two sets.

We now elaborate on step 3 in the algorithm. We apply the Gomory–Hu transformation (section 2.1) to the graph H_i with (a) the original vertex capacities and the edge capacities that have been assigned to the virtual edges in the previous iterations; (b) the cut C; and (c) source-sink pair (x_{i+1}, y_{i+1}). This reduction yields two graphs, $R_{x_{i+1}}$ with condensed vertex y'_{i+1} and $R_{y_{i+1}}$ with condensed vertex x'_{i+1}. Connect now $R_{x_{i+1}}$ with $R_{y_{i+1}}$ by inserting special vertex s_{i+1} and connecting it to its two neighbors $x_{i+1} \in R_{x_{i+1}}$ and $y_{i+1} \in R_{y_{i+1}}$, as can be seen in Figure 5.1.

The resulting graph is denoted by H_{i+1}. Edge capacities are assigned to edges in H_{i+1} as follows: If an edge is a virtual edge in one of the graphs $H_1, \ldots, H_i$ and not in H_{i+1}, then it keeps its capacity. If an edge is a virtual edge in H_{i+1}, then it is assigned capacity $c(e) = f_{\max}(e)$. All other (nonvirtual) edges are assigned infinite capacity.

5.1. Properties. The following two claims follow directly from the construction of H_{i+1}.

CLAIM 5.1. The graph induced by $G - \cup_{j=1}^{i+1} F_j - \{s_{i+2}, \ldots, s_k\}$ is a subgraph of H_{i+1}.

CLAIM 5.2. Graph H_{i+1} does not contain any interesting cycles.

The following theorem guarantees that the first step in each iteration of the algorithm is feasible.

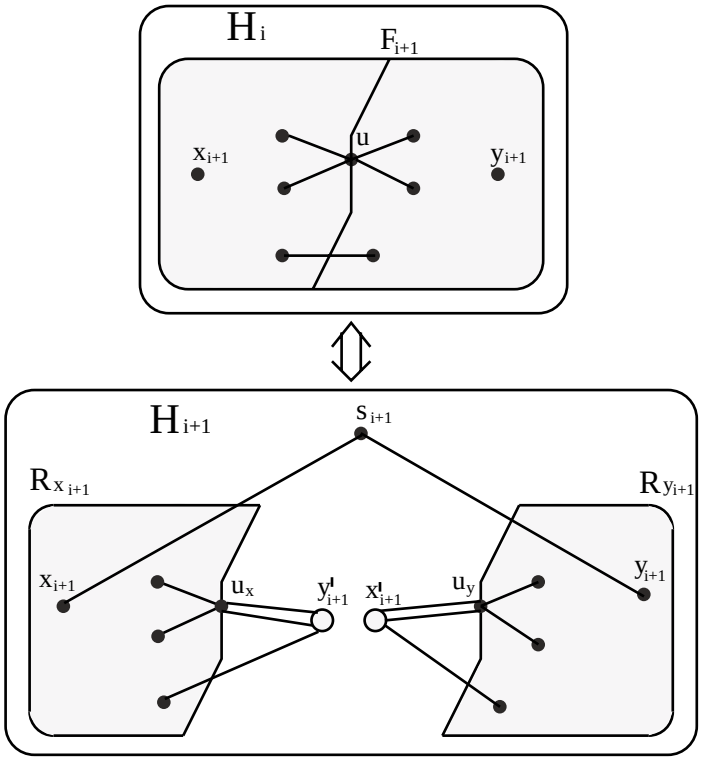


FIG. 5.1. *Obtaining graph H_{i+1} from H_i (and vice-versa).*

THEOREM 5.3. *Suppose there exists a relaxed multicommodity flow function in graph H_i that realizes demand vector $\{d_1, \ldots, d_{i+1}\}$ for source-sink pairs (x'_j, y'_j), $1 \le j \le i$, and (x_{i+1}, y_{i+1}). Then there exists a relaxed multicommodity flow function that realizes demand vector $\{d_1, \ldots, d_{i+1}\}$ for source-sink pairs (x'_j, y'_j), $1 \le j \le i+1$, in graph H_{i+1}.*

Proof. Let $f = (f_1, \ldots, f_{i+1})$ denote the relaxed multicommodity flow computed in iteration $i+1$, in graph H_i. It is obvious that $f_{\max}$ is a restricted function for both graph H_i and graph H_{i+1}. Let H^f_{i+1} denote the restricted graph with respect to graph H_{i+1} and the restricted function $f_{\max}$. We show that for each commodity j, $1 \le j \le i+1$, we can ship d_j units of flow from x'_j to y'_j, and therefore by Claim 4.3, we conclude that there exists a relaxed multicommodity flow function in H_{i+1} that realizes demand vector $\{d_1, \ldots, d_{i+1}\}$ for source-sink pairs (x'_j, y'_j), $1 \le j \le i+1$.

We start with commodity $i+1$. In graph H^f_i, the restricted graph with respect to graph H_i and function $f_{\max}(e)$, d_{i+1} units of flow from x_{i+1} to y_{i+1} went through the cut C. This means that in graph H^f_{i+1} we can ship d_{i+1} units of flow from x_{i+1} to y'_{i+1} and from x'_{i+1} to y_{i+1}. Thus, we can ship the flow of commodity $i+1$ in H^f_{i+1} from x'_{i+1} to y_{i+1}, then to x_{i+1} through s_{i+1}, and then from x_{i+1} to y'_{i+1}.

Suppose that $j < i+1$. We need to distinguish between two cases. The first case is when $x'_j \in R_{x_{i+1}}$ and $y'_j \in R_{y_{i+1}}$ (or vice-versa). We first observe that $d_j \le d_{i+1}$ since the flow of commodity j goes from x'_j to y'_j through cut C in H^f_i. In graph H^f_i, we can ship via cut C flow of commodity j of value d_j from x'_j to y'_j, and flow of commodity $i+1$ of value d_{i+1} from x_{i+1} to y_{i+1}. Therefore, by Lemma 2.1 in the graph H^f_{i+1}, we can ship flow of value d_j from x'_j to y'_{i+1}, and flow of value d_{i+1} from

x_{i+1} to y'_{i+1}. Hence, since $d_j \le d_{i+1}$, we can ship flow of value d_j from x'_j to x_{i+1}. Similarly, we can ship flow of value d_j from y'_j to y_{i+1}. Thus we can ship the flow of commodity j of value d_j from x'_j to y'_j via s_{i+1} in graph H^f_{i+1}. The second case is when both x'_j and y'_j belong to $R_{x_{i+1}}$ (or $R_{y_{i+1}}$). In this case, f_j is a flow of value d_j from x'_j to y'_j in H^f_i. By Lemma 2.2 we get that there exists a flow function of value d_j from x'_j to y'_j in H^f_{i+1}. □

5.2. Correctness. There are three issues that need to be addressed regarding the correctness of the algorithm. First, we need to show that step 1(i) of the algorithm can be implemented, i.e., demand vector $d^i = \{d_1, \ldots, d_i\}$ can be realized in H_i. This can be proved inductively. The basis ($i = 0$) is immediate. Theorem 5.3 guarantees that if d^i can be realized in H_{i-1}, then it can also be realized in H_i.

We prove that at the end of the kth iteration, the set F produced by the algorithm is a SUBSET-FVS in graph G. It follows from Claim 5.1 that $G - F$ is a subgraph of H_k. Claim 5.2 states that H_k does not contain interesting cycles. Therefore, F is a SUBSET-FVS in G.

5.3. Complexity. We show that there is no exponential blowup in the number of vertices in graph H_i as a result of the Gomory–Hu transformation. We claim that the number of virtual vertices in each graph H_i, $1 \le i \le k$, is at most $i \cdot |V|$. For vertex $u \in G$, define the set V_u recursively. Initially, V_u contains u. If, during the run of the algorithm, a vertex $x \in V_u$ generates two virtual copies of itself, then delete x from V_u and add the two virtual copies of x to V_u. We claim that $|V_u| \le k$ for all $u \in G$. This follows by observing that the set of edges in a minimum cut is contained in a biconnected component. The reason for this is as follows: consider two edges in a minimum cut; there are two paths that connect the source with those edges and two paths that connect the sink with those edges. Therefore, they have to be in the same biconnected component. Since every two vertices of V_u belong to different biconnected components of H_i for all $i \ge 0$, it follows that at most one vertex from V_u will belong to a minimum cut in some iteration of the algorithm. Thus, in each iteration, at most one vertex from V_u can generate two new virtual copies, yielding that $|V_u|$ grows by at most 1 at each iteration.

6. Analysis of the algorithm. The analysis of the algorithm is divided into four parts.

1. We define a separation property of source-sink pairs with respect to an optimal SUBSET-FVS. We show that if the separation property is satisfied, then the sum of the demands which can be supplied is bounded by four times the weight of the SUBSET-FVS.
2. We reconfigure the relaxed multicommodity flow in H_k to a relaxed multicommodity flow in $G - \{s_1, \ldots, s_k\}$ with respect to new source-sink pairs. Moreover, the reconfiguration causes the source-sink pairs to satisfy a separation property with respect to an optimal SUBSET-FVS.
3. We show that all the intersaturated vertices chosen in the course of the algorithm are also intersaturated in H_k. This property will allow us to use Theorem 4.4 in order to bound the weights of the chosen intersaturated vertices.
4. We combine the previous parts to obtain the 8-approximation factor.

6.1. Separation of source-sink pairs. Let F^* denote an optimal SUBSET-FVS in graph G. For $0 \le i \le k$, define the graph $G_i = (V_i, E_i)$ to be the subgraph of G

induced by $V - F^* - \{s_{i+1}, \ldots, s_k\}$. We define the following separation property.

DEFINITION 6.1. *Source-sink pairs* $\{(a_j, b_j)\}_{j=i+1}^{k}$ *are* separated *by* F^* *in the subgraph of* G *induced by* $V - \{s_{i+1}, \ldots, s_k\}$ *if they satisfy the following two conditions.*

1. *For every* $i + 1 \leq j \leq k$, *the vertices* a_j *and* b_j *belong to different connected components of graph* G_i.
2. *For every* $i+1 \leq j, j' \leq k$, *the vertices* a_j *and* $a_{j'}$ *belong to different connected components of graph* G_i.

The following theorem enables bounding the sum of the realized demands by the weight of a separating subset of vertices.

THEOREM 6.2. *Suppose that the source-sink pairs* $\{(a_i, b_i)\}_{i=1}^{k}$ *are separated by* F^* *in the subgraph* G' *of* G *induced by* $V - \{s_1, \ldots, s_k\}$. *Then, for every demand vector* $(d_1, \ldots, d_k)$ *in* G' *with respect to the source-sink pairs* (a_i, b_i), *which is realizable by a relaxed multicommodity flow function, the following holds:*

$$\sum_{i=1}^{k} d_i \leq 4 \cdot c(F^*).$$

Proof. Let $f = (f_1, \ldots, f_k)$ denote a relaxed multicommodity flow that realizes the demand vector $(d_1, \ldots, d_k)$. Let C_i denote the connected component in $G' - F^*$ that contains the source a_i. Let T_i denote the set of edges which have one endpoint in F^* and the other endpoint in C_i. By definition, $f_i(e) \leq f_{\max}(e)$ for each edge e. Since $b_i \notin C_i$, the flow of f_i along the edges of T_i cannot be less than d_i, namely, $d_i \leq \sum_{e \in T_i} f_i(e)$. Therefore,

$$d_i \leq \sum_{e \in T_i} f_{\max}(e).$$

Summing up over all commodities, we get that

$$\sum_{i=1}^{k} d_i \leq \sum_{e \in \cup_{i=1}^{k} T_i} f_{\max}(e).$$

Since the sets $\{T_i\}_{i=1}^{k}$ are disjoint,

$$\sum_{e \in \cup_{i=1}^{k} T_i} f_{\max}(e) \leq \sum_{v \in F^*} \sum_{e \in N(v)} f_{\max}(e).$$

By the intercommodity constraints, we get that

$$\sum_{i=1}^{k} d_i \leq 4 \cdot c(F^*). \qquad \square$$

Remark 2. Let α denote the factor by which the intercommodity constraints are relaxed. (In this paper, $\alpha = 2$.) Then, the bound in Theorem 6.2 can be generalized to

$$\sum_{i=1}^{k} d_i \leq 2\alpha \cdot c(F^*).$$

6.2. Reconfiguring flow paths. Let $f = (f_1, \ldots, f_k)$ denote the relaxed multicommodity flow with source-sink pairs $\{(x'_i, y'_i)\}_{i=1}^k$ and demand vector $\boldsymbol{d}$ in the graph H_k computed in the last iteration of the algorithm. Our goal is to define relaxed multicommodity flow $f = (f_1, \ldots, f_k)$ with (new) source-sink pairs $\{(a_i, b_i)\}_{i=1}^k$ and the same demand vector $\boldsymbol{d}$ in the subgraph of G induced by $V - \{s_1, \ldots, s_k\}$ such that the source-sink pairs $\{(a_i, b_i)\}_{i=1}^k$ are separated by F^* in this subgraph.

We construct this relaxed multicommodity flow inductively. For each i, $0 \leq i \leq k$, construct in the graph H_i a relaxed multicommodity flow $f = (f_1, \ldots, f_k)$ with source-sink pairs (x'_j, y'_j), for commodities $1 \leq j \leq i$, and with source-sink pairs (a_j, b_j), for commodities $i + 1 \leq j \leq k$ and demand vector $\boldsymbol{d}$, such that source-sink pairs $\{(a_j, b_j)\}_{j=i+1}^k$ are separated by F^* in the subgraph of G induced by $V - \{s_{i+1}, \ldots, s_k\}$.

Clearly for $i = k$ the relaxed multicommodity flow f is the flow computed in the last iteration of the algorithm. And for $i = 0$ the relaxed multicommodity flow f is the required relaxed multicommodity flow.

Assume that for $i + 1$ we construct such relaxed multicommodity flow $f = (f_1, \ldots, f_k)$ with source-sink pairs (x'_j, y'_j) for commodities $1 \leq j \leq i + 1$, and with source-sink pairs (a_j, b_j) for commodities $i + 2 \leq j \leq k$. Now we construct from it a relaxed multicommodity flow for i.

Overview. Our construction will consist of two steps. In the first step, we choose source-sink pairs (a_j, b_j) for $j \geq i+1$ that satisfy the following. (1) They are separated in G_i. (2) In graph H_{i+1}, demand vector $\{d_1, \ldots, d_k\}$ can be realized with source-sink pairs (x'_j, y'_j) for $1 \leq j \leq i + 1$, and (a_j, b_j) for $j \geq i + 2$, such that flow from (a_j, b_j) for $j \geq i + 2$ should not go through special vertex s_{i+1}.

In the second step, we restrict flow function $f = (f_1, \ldots, f_k)$ to flow paths from (new) source a_j to (new) sink b_j, for $i + 2 \leq j \leq k$. This flow function is henceforth referred to as f. Note that f satisfies, for (a_j, b_j), for $i + 2 \leq j \leq k$: All flow paths from a_j to b_j are subpaths of the flow paths in f prior to the modification. Hence, the amount of flow of commodity j shipped from a_j to b_j in f remains d_j. Thus, we generate the required flow f in H_i from flow function f in H_{i+1}.

Step 1. The construction proceeds by replacing systematically each flow path by a subpath of itself, so that all flow paths in f continue to share a common source and sink.

We assign $a_{i+1} \leftarrow x_{i+1}, b_{i+1} \leftarrow y_{i+1}$ or $a_{i+1} \leftarrow y_{i+1}, b_{i+1} \leftarrow x_{i+1}$. The choice is made so that the sources $a_{i+1}, \ldots, a_k$ belong to different connected components in the graph G_i. Note that since $s_{i+1} \in G_{i+1}$ and both x_{i+1} and y_{i+1} have infinite weight, x_{i+1} and y_{i+1} belong to the same connected component in graph G_{i+1}. Hence, at most one source a_t $(t > i + 1)$ may belong to this connected component. If no source belongs to this component, then both assignments of a_{i+1} and b_{i+1} are good. If a_t belongs to this connected component, then it is either separated from x_{i+1} or from y_{i+1} in graph G_i. In such a case, define a_{i+1} to be the neighbor of s_{i+1} that is separated from a_t in graph G_i.

The assignment of a_{i+1} and b_{i+1} satisfies the property that $\{a_j, b_j\}_{j=i+1}^k$ are separated by F^* in the subgraph of G_i induced by $V - \{s_{i+1}, \ldots, s_k\}$. However, the flow paths of some commodity j for $j > i + 1$ may not satisfy the requirement (2) that they use only vertices in $H_{i+1} - s_{i+1}$. Namely, the flow paths from a_j to b_j $(j > i + 1)$ might pass through special vertex s_{i+1}. (Note that at this stage, the flow paths pass only through vertices in H_{i+1}.) Otherwise, there is no need to update a_j and b_j. Since the special vertex s_{i+1} is about to be removed, this will disconnect all flow paths flowing through it. By construction, s_{i+1} is an articulation vertex of degree

two (i.e., the removal of s_{i+1} partitions H_{i+1} into two connected components), and thus, if a flow path from a_j to b_j passes through s_{i+1}, then all flow paths between a_j and b_j pass through s_{i+1} in H_{i+1}. Hence, it suffices to consider the case where all these paths go through s_{i+1}. We consider two cases.

1. All flow paths enter s_{i+1} from a_{i+1} and exit s_{i+1} to b_{i+1}. In this case, we assign $b_j \leftarrow a_{i+1}$. Since a_j and a_{i+1} do not belong to the same connected component in G_i, a_j and (the updated) b_j also belong to different connected components in G_i.
2. All flow paths enter s_{i+1} from b_{i+1} and exit s_{i+1} to a_{i+1}. There are two subcases: If a_j and b_{i+1} belong to different connected components in G_i, then we assign $b_j \leftarrow b_{i+1}$. Note that now, a_j and b_j are still in different components and that all subpaths of the flow paths that go from a_j to b_j avoid s_{i+1}. If a_j and b_{i+1} are in the same connected component, then a_{i+1} and b_j cannot belong to the same connected component since, otherwise, a_j and b_j would belong to the same connected component of G_{i+1}. In this case, we update $a_j \leftarrow a_{i+1}$; swap a_{i+1} and b_{i+1}, that is, $a_{i+1} \leftarrow b_{i+1}$; and $b_{i+1} \leftarrow a_{i+1}$. Note that now, a_j and a_{i+1} belong to different components, and so do a_j and b_j. Also, all subpaths of the flow paths that go from a_j to b_j avoid s_{i+1}. The swap between a_{i+1} and b_{i+1} does not conflict with separating a_{i+1} from the rest of the sources, because a_j is the only source that belongs to the connected component of a_{i+1} and b_{i+1} in the graph G_{i+1}.

Step 2. Now we generate the required flow f in H_i from flow function f in H_{i+1}. Recall the Gomory–Hu transformation through which graph H_{i+1} was obtained from H_i. Let C denote the minimum cut in the Gomory–Hu transformation. The removal of vertex s_{i+1} from H_{i+1}^f separates the graph into two parts denoted by $R_{x_{i+1}}$ and $R_{y_{i+1}}$. Note that flow function $f_{\max}$ in H_{i+1} is a restricted function for graph H_i. Thus, restricted graph H_i^f (for graph H_i and restricted function f) is a well-defined graph.

We now show that in H_i^f for $i+1 \le j \le k$, there exists a standard single commodity flow function that ships d_j units of flow from a_j to b_j, and for $1 \le j \le i$, there exists a standard single commodity flow function that ships d_j units of flow from x'_j to y'_j. We need to consider several cases.

The easiest case is when $j = i+1$. Recall that vertices x'_{i+1} and y'_{i+1} represent condensed subgraphs in H_i^f. In H_{i+1}, commodity $i+1$ is shipped from x'_{i+1} to y_{i+1} and then, via s_{i+1}, from x_{i+1} to y'_{i+1}. By Lemma 2.1, we can ship in H_i^f flow of value d_{i+1} from x_{i+1} to y_{i+1}. Vertices a_{i+1} and b_{i+1} were already matched to x_{i+1} and to y_{i+1} (or vice-versa) in the first part of the proof, and thus we show that d_{i+1} units of flow can be shipped from a_{i+1} to b_{i+1}.

Suppose that $j < i+1$. We need to distinguish between two cases. The first case is when $x'_j \in R_{x_{i+1}}$ and $y'_j \in R_{y_{i+1}}$ (or vice-versa). We first observe that $d_j \le d_{i+1}$ since the flow of commodity j went from x'_j to y'_j through cut C in H_i in iteration $i+1$ of the algorithm. By Lemma 2.1, in graph $R_{x_{i+1}}$, d_j units of flow can be shipped from x'_j to x_{i+1}. Also, in graph $R_{x_{i+1}}$, d_{i+1} units of flow can be shipped from x_{i+1} to y'_{i+1}. Thus, since $d_j \le d_{i+1}$, in graph $R_{x_{i+1}}$, d_j units of flow can be shipped from x'_j to y'_{i+1}. Similarly, in graph $R_{y_{i+1}}$, d_j units of flow can be shipped from x'_{i+1} to y'_j. Therefore, by Lemma 2.1, in graph H_i^f we can ship d_j units of flow of commodity j from x'_j to y'_j.

The second case is when both x'_j and y'_j belong to $R_{x_{i+1}}$ (or $R_{y_{i+1}}$). By Lemma 2.1, in graph $R_{x_{i+1}}$, there exists a flow of value d_j from x'_j to y'_j. Therefore, by

Lemma 2.2, in graph H_i^f, we can ship d_j units of flow of commodity j from x'_j to y'_j.

The last case is for commodities j for $i+1 < j \leq k$. It follows from the construction described in Step 1 that a_j and b_j are not separated by cut C in graph H_i^f. This case can be handled similarly to the previous subcase.

We have now established that in H_i^f for $i+1 \leq j \leq k$, there exists a standard single commodity flow function that ships d_j units of flow from a_j to b_j, and for $1 \leq j \leq i$, there exists a standard single commodity flow function that ships d_j units of flow from x'_j to y'_j. Notice that for each vertex $v \in H_i^f$, the capacity of the edges in $N(v)$ is either obtained from f in H_{i+1}, or, all the edges adjacent to v are virtual edges whose capacity is obtained from some legal relaxed multicommodity flow function in H_i. Therefore, we have that for each vertex $v \in H_i^f$, $\sum_{e \in N(v)} c(e) \leq 4 \cdot c(v)$. Now by Claim 4.3 we can superposition all the single commodity flow functions obtained for each commodity and get a legal relaxed multicommodity flow function f that realizes demand vector $d_1, \ldots, d_k$ in H_i. This completes the construction of required relaxed multicommodity flow function f in H_i.

6.3. Intersaturated vertices.

THEOREM 6.3. *Intersaturated vertices chosen by the algorithm are also intersaturated in H_k, namely,*

$$\bigcup_{i=1}^{k} B_i \subseteq I_k.$$

Proof. The proof follows by observing that for each edge e, $f_{\max}(e)$, which we constructed in the previous subsection in H_i, cannot be larger than $f_{\max}(e)$ in H_{i+1}. Thus, if a vertex is not intersaturated in H_{i+1}, then it is also not intersaturated in H_i. Therefore if a vertex is intersaturated in H_i for some $1 \leq i \leq k$, then it is intersaturated in H_k. ∎

6.4. The approximation factor. We are now ready to finish the analysis of the approximation factor of the algorithm. Our goal is to bound the weight of the chosen feedback set, namely, $c(F)$. To meet this end, we consider separately the chosen intrasaturated vertices and the chosen intersaturated vertices. Recall that in the ith iteration we add to F a subset $F_i = A_i \cup B_i$. The vertices in A_i are a subset of a minimum cut corresponding to the flow f_i, and hence, $c(A_i) \leq d_i$. Using Theorem 6.3, we bound $c(\cup_{i=1}^{k} B_i)$ by $c(I_k)$. We apply Theorem 4.4 to the graph H_k and obtain $c(I_k) \leq \sum_{i=1}^{k} d_i$. Therefore,

$$c(F) \leq 2 \cdot \sum_{i=1}^{k} d_i.$$

We now provide an upper bound on the sum of the demands. It follows from reconfiguration of the flow paths that in the graph $G - \{s_1, \ldots, s_k\}$ there exists a relaxed multicommodity flow function with source-sink pairs $\{(a_i, b_i)\}_{i=1}^{k}$ that realizes demand vector $(d_1, \ldots, d_k)$. Moreover, these source-sink pairs are separated by F^*, and thus by Theorem 6.2,

$$\sum_{i=1}^{k} d_i \leq 4 \cdot c(F^*),$$

yielding that $c(F) \leq 8 \cdot c(F^*)$.

We now can explain the choice of the relaxation factor of the intercommodity constraints. Let this factor be denoted by α. It follows from Remarks 1 and 2 that the approximation factor of the algorithm with respect to α is

$$2\alpha \cdot \left(1 + \frac{1}{\alpha - 1}\right).$$

This expression is minimized for $\alpha = 2$.

Acknowledgment. We would like to thank Naveen Garg for many helpful discussions.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanati, and J. B. Orlin, *Network Flows*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] V. Bafna, P. Berman, and T. Fujito, *A 2-approximation algorithm for the undirected feedback vertex set problem*, SIAM J. Discrete Math., 12 (1999), pp. 289-297.

[3] A. Becker A. and D. Geiger, *Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem*, Artificial Intelligence, 83 (1996), pp. 167–188.

[4] G. Calinescu, H. Karloff, and Y. Rabani, *An improved approximation algorithm for multiway cut*, in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 48–52.

[5] E. Dahlhaus, D. S. Johnson, C. H. Papadimitriou, P. D. Seymour, and M. Yannakakis, *The complexity of multi-terminal cuts*, SIAM J. Comput., 23 (1994), pp. 864–894.

[6] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multi-cuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[7] G. Even, J. Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading metrics*, in Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1995, pp. 62–71; J. ACM, to appear.

[8] G. Even, J. Naor, B. Schieber, and L. Zosin, *Approximating minimum subset feedback sets in undirected graphs with applications*, SIAM J. Discrete Math., 13 (2000), pp. 255–267.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* W. H. Freeman, San Francisco, California, 1979.

[10] N. Garg, V. V. Vazirani, and M. Yannakakis, *Multiway cuts in directed and node weighted graphs*, in Proceedings of the 21st International Colloquium on Automata, Languages and Programming (ICALP), Lecture Notes in Comput. Sci. 820, Springer, New York, 1994, pp. 487–498.

[11] M. X. Goemans and D. P. Williamson, *Primal-dual approximation algorithms for feedback problems in planar graphs*, Combinatorica, 18 (1998), pp. 37–59.

[12] R. E. Gomory and T. C. Hu, *Multi-terminal network flows*, J. SIAM, 9 (1961), pp. 551–570.

[13] F. Granot and R. Hassin, *Multi-terminal maximum flows in node-capacitated networks*, Discrete Appl. Math., 13 (1986), pp. 157–163.

[14] T. C. Hu, *Combinatorial Algorithms*, Addison-Wesley, Reading, MA, 1982.

[15] D. R. Karger, P. N. Klein, C. Stein, M. Thorup, and N. E. Young, *Rounding algorithms for a geometric embedding of minimum multiway cut*, in Proceedings of the 31st ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 668–678.

[16] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–104.

[17] P. N. Klein, S. A. Plotkin, S. Rao, and É. Tardos, *Approximation algorithms for Steiner and directed multicuts*, J. Algorithms, 22 (1997), pp. 241–269.

[18] T. Leighton and S. Rao, *Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms*, J. ACM, 46 (1999), pp. 787–832.

[19] J. Naor and L. Zosin, *A 2-approximation algorithm for the directed multiway cut problem*, in Proceedings of the 38th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 548–553.

[20] P. D. Seymour, *Packing directed circuits fractionally*, Combinatorica, 15 (1995), pp. 281–288.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1253–1298

COMPUTATIONALLY SOUND PROOFS*

SILVIO MICALI†

Abstract. This paper puts forward a new notion of a proof based on computational complexity and explores its implications for computation at large.

Computationally sound proofs provide, in a novel and meaningful framework, answers to old and new questions in complexity theory. In particular, given a random oracle or a new complexity assumption, they enable us to

1. prove that verifying is easier than deciding for all theorems;
2. provide a quite effective way to prove membership in computationally hard languages (such as $\mathcal{C}o$-$\mathcal{NP}$-complete ones); and
3. show that every computation possesses a short certificate vouching its correctness.

Finally, if a special type of computationally sound proof exists, we show that Blum's notion of program checking can be meaningfully broadened so as to prove that $\mathcal{NP}$-complete languages are checkable.

Key words. interactive proofs, probabilistically checkable proofs, random oracles, Merkle trees

PII. S0097539795284959

1. Introduction.

CS proofs. Proofs are fundamental to our lives, and as for all things fundamental we should expect that answering the question of what a proof is will always be an on going process. Indeed, we wish to put forward the new notion of a *computationally sound proof* (CS proof) which achieves new and important goals, not attained or even addressed by previous notions.

Informally, a CS proof of a statement S consists of a short string, σ, which (1) is as easy to find as possible, (2) is very easy to verify, and (3) offers a strong computational guarantee about the verity of S. By "as easy to find as possible" we mean that a CS proof of a *true* statement (i.e., for the purposes of this paper, *derivable* in a given axiomatic theory) can be computed in a time close to that needed to Turing-accept S. By "very easy to verify" we mean that the time necessary to inspect a CS proof of a statement S is substantially smaller than the time necessary to Turing-accept S. Finally, by saying that the guarantee offered by a CS proof is "computational" we mean that false statements either do not have any CS proofs, or such "proofs" are practically impossible to find.

Implementations of CS proofs. The value of a new notion, of course, crucially depends on whether it can be sufficiently exemplified. We provide two main implementations of our notion. The first is based on a random oracle and provably yields a CS proof system without any unproven assumption. The second relies on a new complexity conjecture: essentially, that it is possible to replace the random oracle of the first construction with a cryptographic function and obtain, mutatis mutandis, similar results.

Applications of CS proofs. In either implementation, CS proofs provide, in a new and meaningful framework, very natural answers to some of our oldest questions in complexity theory. In particular, they imply not only that the time necessary to verify is substantially smaller than the time necessary to accept, but, more importantly,

*Received by the editors April 21, 1995; accepted for publication (in revised form) October 28, 1999; published electronically October 11, 2000.
http://www.siam.org/journals/sicomp/30-4/28495.html
†Laboratory for Computer Science, MIT, Cambridge, MA 02139 (silvio@theory.lcs.mit.edu).

that this "speed-up" occurs for *all* theorems (rather than just *a few* theorems). In addition, they provide a quite effective way for proving membership in computationally hard languages (e.g., $\mathcal{C}o$-$\mathcal{NP}$ complete ones).

CS proofs also possess novel and important implications for computational correctness. In particular, in either implementation, they imply that every computation possesses a short certificate vouching for its correctness. In addition, if implementable in the second manner, CS proofs also imply that any heuristic or program for an $\mathcal{NP}$-complete problem is cryptographically checkable. This application at the same time extends and demonstrates the wide applicability of Blum's [11] original framework for checking program correctness.

Origins of CS proofs. In conceiving and constructing CS proofs, we have benefited from the research effort in interactive and zero-knowledge proofs. In particular, the notion of a probabilistically checkable proof [3, 17] and that of a zero-knowledge argument [13] have been the closest sources of inspiration in conceiving the new notion itself. In exemplifying the new notion, most relevant has been a construction of Kilian's [23], and, to a lesser extent, the works of [18] and [9].

Naturalness of CS proofs. We wish to emphasize that, from the above starting point, the mentioned applications of CS proofs to computation at large have been obtained by means of surprisingly simple arguments. Indeed, after setting up the stage for the new notion, the results about computational correctness follow quite naturally. This simplicity, in our opinion, lends support to our new perspective.

2. New goals for efficient proofs.

Proofs without demands for efficiency: Semirecursive languages. Truth and proofs have been traveling hand in hand. As formalized in the first half of this century by a brilliant series of works, the classical notion of a proof [1] is inseparable from that of a true statement. Given any finite set of axioms and inference rules, the corresponding true statements form a *semirecursive* set.[2] In the expressive and elegant approach of Turing, such sets possess two equivalent characterizations particularly important for our enterprise, one in terms of accepting algorithms and one in terms of verifying algorithms.

1. A language (set of binary strings) L is semirecursive if and only if there exists an *(accepting)* Turing machine A such that
$$L = \{x \ : \ A(x) = YES\}.$$
2. A language L is semirecursive if and only if there exists a *(verifying)* Turing machine V, halting on all inputs, such that
$$L = \{x \ : \ \exists \sigma \in \{0,1\}^* \text{ such that } V(x,\sigma) = YES\}.$$

Establishing the verity of a statement is thus a purely algorithmic process, and (at least formally) classical proofs—the σs of the second definition—are just strings. Because in this paper a "true" theorem simply is one derivable in a given theory, for

[1] Thinking that the intuitive notion of a proof has remained unchanged from the times of classic Greece (i.e., thinking that people like Peano, Zermelo, Frankel, Church, Turing, and Gödel have only contributed its rigorous formalization and the discovery of its inherent limitations) is certainly appealing, but unrealistic. Personally, we believe that no notion so fundamental and so human can remain, not even intuitively, the same across so different spiritual experiences and historical contexts. No doubt, our yearning for permanence (dictated by our intrinsically transient nature) predisposes us to perceive more continuity in our endeavors than may actually exist.

[2] Again, throughout this work, *true* is considered equivalent to *derivable*.

variety of discourse we may call a member of a semirecursive language a *theorem* or a *true statement*, and a classical proof a *derivation*.

Finally, we also wish to recall the definition of a recursive language. Namely,

3. a language L is recursive (decidable) if and only if there exists a *(deciding)* Turing machine D, halting on all inputs, such that

$$L = \{x \ : \ D(x) = YES\}.$$

Most of the semirecursive languages considered later on are actually recursive, in which case we may refer to their accepting algorithms as deciding algorithms.

Prior demands for efficiency: $\mathcal{NP}$, $\mathcal{IP}$, and $\mathcal{PCP}$. The two classical definitions of semirecursiveness are syntactically different, but they are not formally distinct from the point of view of "computational efficiency." Indeed, though in many natural cases verifying a classical proof is computationally preferable to finding it, classical proofs are more a way of expressing what is *in principle true* rather than a way of capturing what is *efficiently provable*. Providing a derivation is certainly a way to convince someone that a given theorem is true but not necessarily an efficient one: a classical proof may be arbitrarily long, or its relative verifying algorithm may take arbitrarily many computational steps to verify it.

Therefore, the now familiar notions of $\mathcal{NP}$ (due to Cook [16] and, independently, to Levin [24]) and $\mathcal{IP}$ (due Goldwasser, Micali, and Rackoff [21] and, independently, to Babai and Moran [4]) have been put forward in an effort to capture the essence of an efficient proof. Despite the notable differences between $\mathcal{NP}$ and $\mathcal{IP}$, in both cases this effort consists of *demanding that verifying be easy*.

Our notion too demands ease of verification (and in a *stronger* sense), but also broadens the perceived essence of an efficient proof by demanding some novel properties.

Another type of "proof efficiency" is provided by the notion of $\mathcal{PCP}$ [3, 17] (which we shall discuss in some detail later on). Quite succinctly, this notion consists of an explicit algorithm transforming an $\mathcal{NP}$-witness, σ, into a new proof (i.e., string), τ, which is polynomially longer, but whose correctness can be detected in probabilistic polylogarithmic time by random accessing (at unit cost) selected bits of τ. This immediately yields the following $\mathcal{NP}$-like proof system: the prover transforms an $\mathcal{NP}$-witness σ into a longer but samplable proof-string τ, and sends τ to the verifier, who then will verify τ by selectively sampling its bits.

In terms of overall verifying time, however, such a proof system is not more efficient than its $\mathcal{NP}$ counterpart (i.e., than just sending σ). Indeed, though few chosen bits of τ will be "truly checked," to ensure that he is truly dealing with, say, the ith bit of τ, the verifier must *read/receive* every bit of τ and keep precise track of the order in which it is read/received. And such read/receiving operations, according to any natural measure, have in themselves a cost proportional to τ's length, which is greater than the length of σ.[3]

(Notice that having the prover not send τ at all, but rather having him answer any question the verifier may ask about specific bits of τ, does not work: a dishonest prover may cheat successfully with probability 1.)

Our demands for efficiency. Our notion of an efficient proof system is expressed in relation to Turing acceptability. We believe this to be a necessary step.

[3]This continues to be true if the prover sends to the verifier a piece of randomly-accessible hardware containing τ, if such a transmission is deemed preferable.

Indeed, we perceive *accepting* as the solitary process of determining what is true and *proving* as the social process of conveying to others the results of this determination.

Let us begin with the desideratum common to prior notions of our proof: ease of verification. Both $\mathcal{NP}$ and $\mathcal{IP}$ interpret ease of verification in *absolute terms*, namely, by requiring that verifiers run in polynomial-time. This interpretation automatically and strictly narrows the class of the efficiently provable theorems. By contrast, we believe that efficient verifiability should be expressed in *relative* rather than absolute terms, namely, by comparing the complexities of (1) *verifying* the proof of a given statement S and (2) *accepting* S (i.e., establishing S's verity without any help).

In addition, we perceive two new desiderata. First, because a proof system specifies (implicitly or explicitly) two processes, that of verifying and that of *proving*, we believe that proving too should be efficient, and this latter efficiency should again be relative to the complexity of accepting. Second, while $\mathcal{NP}$ and $\mathcal{IP}$ narrow the provable theorems to a small subset of all true statements (e.g., $\mathcal{PSPACE}$), we believe that *all* true statements (i.e., all Turing-acceptable languages) should be efficiently provable.

Our main goals. In sum, at the highest and informal level, the objective of our new notion of a proof is *finding the right relationship between* accepting, efficiently proving, *and* efficiently verifying *a true statement.* We articulate this general objective in the following goals.

1. *(Relative) efficiency of verifying.* Construct proof systems so that, for *all* theorems, the complexity of verifying is substantially smaller than that of accepting.
2. *(Relative) efficiency of proving.* Construct proof systems so that the prover's complexity is close to that of accepting.
3. *(Recursive) universality.* Construct proof systems capable of efficiently proving membership in *every* semirecursive language.

As we shall point out in what follows, our notion of a CS proof system also achieves additional goals, but we do not consider them essential to the "right" notion of an efficient proof.

2.1. Efficiency of verifying.

The relative nature of efficient verifiability. As outlined above, we regard a proof system to be efficient if it makes verifying a given statement easier than Turing accepting it (i.e., easier than establishing its verity without the help from any prover). Ignoring a small, fixed polynomial, we demand that the complexity of verifying be polylogarithmic in that of accepting. Though somewhat arbitrary, the latter choice stems from two simple reasons: "logarithmic" because we wish the advantage of verifying over accepting to be *substantial* (whenever the accepting time is substantial!), and "poly" because we wish such an advantage to be reasonably *independent* from any specific computational model.

The ubiquitous nature of efficient verifiability. There is an additional and novel aspect to our goal of efficient verifiability, namely, that such efficiency should arise for *all* theorems and not for just some of them. Let us explain this point focusing on the $\mathcal{NP}$ proof system.

To begin with, within $\mathcal{P}$, the $\mathcal{NP}$ mechanism does not guarantee that verification is computationally easier than acceptance (which in this case coincides with decision). For instance, in principle, for infinitely many positive constants c there could be a language L_c decidable in time $O(n^c)$ but for which any type of $\mathcal{NP}$-witness needs $\Omega(n^c)$ steps to be verified. If this were the case, the $\mathcal{NP}$ proof system could not make

verifying membership in these languages any easier than deciding them.

In addition, in principle again, assuming $\mathcal{P} \neq \mathcal{NP}$ only entails the existence of a superpolynomial gap between the complexities of Turing accepting and verifying for some, *but possibly rare*, inputs, such as certain instances of satisfiability. By contrast, according to our present point of view, a proof system does not make verification sufficiently efficient unless it makes it polylogarithmically easier than accepting for essentially *all* theorems.

2.2. Efficiency of proving.

The relative nature of efficient provability. As mentioned already, implicitly or explicitly, proofs involve *two* agents, a prover and a verifier. We thus believe that the right notion of a proof should require efficiency for *both* agents and that the efficiency of a prover should not be measured in absolute terms but relatively to the complexity of Turing accepting the problem at hand.

Measuring prover efficiency relative to the complexity of accepting is a quite natural choice. Indeed, based on our intuition that the complexity of convincing someone else cannot be lesser than that of convincing ourselves (and based on our view that accepting is the process of convincing ourselves), the complexity of proving cannot be lower than that of accepting, while it could be much greater. We thus demand that our proof systems satisfy the following two properties.

(i) The prover must succeed in convincing the verifier whenever the theorem at hand is true (the old completeness property of an interactive proof system).

(ii) The amount of computation needed by the prover to convince the verifier must be polynomially close to that needed to accept that the given theorem is true.

In property (ii), we demand that the two amounts of computation be polynomially close to ensure a reasonable robustness.

We refer to the simultaneous holding of these two properties as *feasible completeness*. Feasible completeness is a novel *requirement* for proof systems. But do any of the prior proof systems "happen" to enjoy it anyway? Quite possibly, the answer is no. Consider, for instance, an $\mathcal{NP}$ language L (preferably not $\mathcal{NP}$-complete[4]) decidable by an algorithm D in, say, $n^{\log n}$ time. Then, in the $\mathcal{NP}$ mechanism, proving that a given string x belongs to L entails finding a polynomially long and polynomial-time inspectable witness w_x. But the complexity necessary to find such an *insightful* string may vastly exceed that of running algorithm D on input x for $|x|^{\log |x|}$ steps! Indeed, finding such an insightful string w_x might in the worst case require $O(2^{|x|})$ steps. In other words, while a few months of hard work may suffice for proving to ourselves (i.e., for accepting) that a given mathematical statement is true, it is conceivable that a lifetime may not be enough for finding an explanation followable by a verifier with a limited attention span.

Efficient provability might also not hold for the $\mathcal{IP}$ proof mechanism. Indeed, often the best way to prove membership in an $\mathcal{IP}$ language consists of invoking the general $\mathcal{IP} = \mathcal{PSPACE}$ protocol [26, 33], which is extremely wasteful of prover resources.

Realizing the importance of feasible completeness in a proof system allows us

[4]Above, we assume that L is not $\mathcal{NP}$-complete to avoid raising two issues at once. Indeed, due to our current complexity measures, $\mathcal{NP}$-proving membership in an $\mathcal{NP}$-complete language *appears* feasible. In fact, because of self-reducibility, if L is $\mathcal{NP}$-complete and decidable in $n^{\log n}$ time, then an $\mathcal{NP}$-witness of $x \in L$ is findable in $poly(|x|) \cdot |x|^{\log |x|}$ time. However, as we shall see in subsection 5.8, $\mathcal{NP}$ may not enjoy feasible completeness even when one focuses solely on $\mathcal{NP}$-complete languages.

to raise a variety of intriguing questions about $\mathcal{NP}$ and $\mathcal{IP}$.[5] But our point is not determining which proof systems *happen* to enjoy feasible completeness. Our point is that *feasible completeness must be* required *from any notion of a proof system that aims at achieving an adequate level of generality and meaningfulness.*

Related notions of feasible provability. Related notions of "feasible" provability have been considered in the past. In particular, Bellare and Goldwasser [6] discuss demonstrating membership in certain languages L by provers working in polynomial time and having access to an oracle for membership in L. Their notion, however, is weaker than ours because, in order to demonstrate membership in L of some given input x, their prover can query the oracle about other inputs x' for which accepting membership in L might be "much harder" than for x (despite the fact that such x' have been polynomial-time computed from x). Thus, if accessing the oracle for L were to be substituted with running an algorithm deciding L, then their provers may work much harder than needed for accepting that a specific x belongs to L.

Less relevantly, the protocols of [20] and [13] show that, if a prover were given for free an $\mathcal{NP}$ witness that an input x belongs to an $\mathcal{NP}$-complete language L, then proving in zero-knowledge that $x \in L$ only requires polynomial (in $|x|$) work. (In a sense, therefore, theirs is an example of feasible provability, but relative to the "nondeterministic complexity of x." That is, the prover complexity of a zero-knowledge proof system for $\mathcal{NP}$ is shown to be feasible relative to the prover complexity of another proof system: the $\mathcal{NP}$ one.) Such notion is nonetheless adequate when the prover is not handed the statement of a theorem (e.g., $x \in L$) as an input, but rather generates it together with suitable auxiliary information (e.g., an $\mathcal{NP}$-witness of $x \in L$) that enables feasible proving.

By contrast, our notion of feasible completeness refers to (1) *individual* inputs and (2) the *deterministic complexity* of these inputs.

2.3. Recursive universality. The previous proof systems discussed above have only a limited "range of action." For instance, an interactive proof system (P, V) is defined only with respect to proving membership in a *specific* language L. Different languages have, therefore, different interactive proof systems, or none. Moreover, as mentioned above, even considering the classes of all languages having an interactive proof system, one obtains a set of languages, $\mathcal{IP}$, that is quite small with respect to the set of all semirecursive languages.

We instead consider *universality* (i.e., the capability of handling the entire range of semirecursive languages) to be a necessary property of a "sufficiently right" proof system. By this we do not just mean that every semirecursive language should admit a proof system of the "right" type. We actually mean that a "right" proof system should be able to prove membership in *any* semirecursive language. That is, for any language L and any member x of L, on input x and a suitable description of L, a right proof system should be able to prove, *efficiently*, that x belongs to L. (As will be seen, we consider an accepting algorithm for L to be a suitable description of L

[5] For instance, in an intuitive language,

$\mathcal{Q}1$: *What is the computational complexity required from any $\mathcal{IP}$-prover of unsatisfiability?*

$\mathcal{Q}2$: (In light of better-than-exhaustive-search algorithms for graph isomorphism) *What is the complexity required from any $\mathcal{NP}$-prover of, say, graph isomorphism?*

$\mathcal{Q}3$: *Are there $\mathcal{NP}$-languages L, such that proving membership in L may require much less computation from an $\mathcal{IP}$-prover than from an $\mathcal{NP}$-prover?*
(i.e., can giving a prover "more freedom" save him much work?) In particular,

$\mathcal{Q}3'$: *What is the computational complexity required from any $\mathcal{IP}$-prover of satisfiability?*

and one that facilitates our establishing the efficient provability and verifiability of a proof system.)

3. CS proofs with a random oracle. To approximate our new goals, we put forward the notion of a CS proof system. As we shall see, this actually is a family of closely related notions. The first of such notions, that of a CS proof with a random oracle, will be presented and implemented in detail in this section; the others will be more briefly discussed in section 4. All these proof systems aim at proving membership in the following special language.

3.1. The CS language.

Encodings. Throughout this paper, we assume usage of a standard binary encoding, and often identify an object with its encoding. (In particular, if A is an algorithm, we may—meaningfully, if informally—give A as an input to another algorithm.) The length of an (encoded) object x is denoted by $|x|$. If q is a quadruple of binary strings, $q = (a, b, c, d)$, then our quadruple encoding is such that, for some positive constant c,

$$1 + |a| + |b| + |c| + |d| < |q| < c(1 + |a| + |b| + |c| + |d|).$$

Steps. If M is a Turing machine and x an input, we denote by $\#M(x)$ the number of steps that M takes on input x.

DEFINITION 3.1. *We define the* CS language*, denoted by $\mathcal{L}$, to be the set of all quadruples $q = (M, x, y, t)$, such that M is (the description of) a Turing machine, x and y are a binary strings, and t a binary integer such that*

1. $|x|, |y| \leq t$;
2. $M(x) = y$; and
3. $\#M(x) = t$.

Notice that, as long as M reads each bit of its inputs and writes each bit of its outputs, the above property 1 is not a real restriction. Notice too that, due to our encoding, if $q = (M, x, y, t) \in \mathcal{L}$, then $t < 2^{|q|}$.

3.2. The notion of a CS proof-system with a random oracle.

Oracles and oracle-calling algorithms. We denote the set of all binary strings having length i by Σ^i, and the set of all functions from a-bit strings to b-bit strings by $\Sigma^a \to \Sigma^b$. By an *oracle* we mean a function in $\Sigma^a \to \Sigma^b$, for some choice of a and b.

We consider algorithms making calls to one or two oracles. To emphasize that an algorithm A makes calls to a single oracle, we write $A^{(\cdot)}$. If A is such an algorithm and f an oracle, we write A^f to denote the algorithm obtained by answering A's queries according to function f, that is, by answering each query α with $f(\alpha)$. Similarly, to emphasize that an algorithm A makes calls to two oracles, we write $A^{(\cdot,\cdot)}$. If A is such an algorithm and (f_1, f_2) a pair of oracles, we write $A^{(f_1,f_2)}$ to denote the algorithm obtained by answering A's queries to the first oracle according to function f_1, and those to the second oracle according to function f_2.

For complexity purposes, in a computation of an oracle-calling algorithm, the process of writing down a query and receiving its answer from the proper oracle f is counted as a single step. No result of this paper would change in any essential way if this process costed $poly(a, b)$ steps whenever $f \in \Sigma^a \to \Sigma^b$.

An algorithm that, in any possible execution, makes exactly N calls to each of its oracles will be referred to as a *N-call algorithm.*

Integer presentation. If x is an integer given as an input to an algorithm A, unless otherwise clarified it is assumed that x is presented in binary to A. We only make an exception for the security paramenter, denoted by k, that is always presented in unary to all our algorithms: accordingly, we shall denote by 1^k (i.e., the concatenation of k 1s) the unary representation of integer k.[6]

DEFINITION 3.2. *Let $P^{(\cdot)}$ and $V^{(\cdot)}$ be two oracle-calling Turing machines, the second of which is running in polynomial time. We say that (P, V) is a* CS proof system with a random oracle *if there exists a sequence of six positive constants, $c_1, \ldots, c_6$ (referred to as the* fundamental constants *of the system), such that the following two properties are satisfied.*

1′. *Feasible completeness.* For all $q = (M, x, y, t) \in \mathcal{L}$, for all k, for all $f \in \Sigma^{k^{c_1}} \to \Sigma^{k^{c_1}}$, (1′.*i*) $P^f(q, 1^k)$ halts within $(|q|kt)^{c_2}$ computational steps, outputting a binary string $\mathcal{C}$ whose length is $\leq (|q|k)^{c_3}$, and (1′.*ii*) $V^f(q, 1^k, \mathcal{C}) = YES$.

2′. *Computational soundness.* For all $\widetilde{q} \notin \mathcal{L}$, for all k such that $2^k > |q|^{c_4}$, and for all (cheating) deterministic $2^{c_5 k}$-call algorithm $\widetilde{P}$, for a random oracle $\rho \in \Sigma^{k^{c_1}} \to \Sigma^{k^{c_1}}$,

$$Prob_\rho[V^\rho(\widetilde{q}, 1^k, \widetilde{P}^\rho(\widetilde{q}, 1^k)) = YES] \leq 2^{-c_6 k}.$$

Thus, an execution of (P, V) requires a common oracle f and two common inputs: a quadruple of binary strings q (allegedly a member of $\mathcal{L}$) and a unary-presented integer k. We refer to q as *the CS input*, and to k as *the security parameter*. Such an execution consists of first running P^f on inputs q and 1^k, so as to produce a binary output $\mathcal{C}$, and then running V^f on inputs q, 1^k and $\mathcal{C}$. If $q = (M, x, y, t)$ and $V^f(q, 1^k, \mathcal{C}) = YES$, we may call string $\mathcal{C}$ a *random-oracle CS proof of $M(x) = y$*, or, more precisely, a *random-oracle CS proof, of security k, of $M(x) = y$ in less than t steps*. For variation of discourse, we may sometimes refer to such a $\mathcal{C}$ as a CS *witness* or a CS *certificate*. If it is clear from the context that we are dealing with CS proof systems with a random oracle, we may simplify our language by dropping the qualification "random-oracle."

Discussion.

Controlled inconsistency. CS proofs (similarly to zero-knowledge arguments discussed later on) allow the existence of false proofs but ensure that these are computationally hard to find. That is, *false CS proofs may exist, but they will "never" be found.*

Equivalently, CS proof systems are deliberately inconsistent but practically indistinguishable from consistent systems. Indeed, each CS proof specifies a security parameter, controlling the amount of computing resources necessary to "cheat" in the proof, so that these resources can be made arbitrarily high. Accordingly, CS proofs are meaningful only if we believe that the provers who produced them, though more powerful than their corresponding verifiers, are themselves computationally bounded.[7] From a practical point of view, this is hardly a limitation. As long we restrict our attention to physically implementable processes, no prover in our universe can perform $2^{1,000}$ steps of computation, at least during the existence of the human race. Thus, "practically speaking" all provers are computationally bounded.

[6] This is to ensure that a polynomial-time algorithm is guaranteed to be able to make "*poly(k)*" steps when the security parameter is k.

[7] The transition from an interactive proof system to a CS proof system is analogous to the transition from a perfect zero-knowledge proof system to a computational zero-knowledge proof system [21], which has proved to be a more flexible and powerful notion [20].

Deterministic cheating. In the above definition of a CS proof system with a random oracle, we have considered cheating provers $\widetilde{P}$ to be deterministic because probabilism does not help in our context. Indeed, since we are not concerned about the size of the description of $\widetilde{P}$, nor about its running time (except for the number of oracle calls it may make), $\widetilde{P}$ may easily have built in any "lucky" sequence of coin tosses.

Security parameters. Informally, the security parameter k controls the probability of something going wrong, and CS proofs become more meaningful as k grows. But, at a minimum, we require that k be large enough so that $2^k > |q|^{c_4}$. This "mild" lower-bound is relied upon in our proof of Theorem 3.8.[8] At the same time, it is also reasonable in that, on input a quadruple q, the honest prover P is allowed at least $poly(|q|)$ steps of computation, and it would thus be strange not to assume that a cheating prover can make a similar number of steps.

Running time. A member of $\mathcal{L}$, $q = (M, x, y, t)$, includes the exact number of steps, t, in which M outputs y on input x. More simply, however, we could have demanded that t upperbounds $\#M(x)$. But since the CS proof system with a random oracle of section 3.4.2, $(\mathcal{P}, \mathcal{V})$, actually proves the exact value of $\#M(x)$, it would have been a pity to loose this exact information.[9]

A paradox. CS proofs are paradoxical in that a computationally bounded prover appears able to "prove more theorems" than an unbounded one. Indeed, if we choose k's value as a suitable function of the input length, then a properly-bounded CS prover can demonstrate membership in any $\mathcal{EXPTIME}$ language to a verifier whose running time is upperbounded by a fixed polynomial in the input length alone. By contrast, the unbounded prover of an interactive proof system can only prove membership in $\mathcal{PSPACE}$ languages to a polynomial-time verifier, and it is widely believed that $\mathcal{PSPACE}$ is a proper subset of $\mathcal{EXPTIME}$.

However, at a second thought, there is no paradox. Indeed, a prover, being someone more powerful than us, may be a potentially useful *ally* (willing to enlarge our state of knowledge by letting us verify that some very difficult theorems are indeed true), but may also be a potentially dangerous *enemy* (wishing to trick us into believing some false statements). It is thus not too surprising that, when a prover is powerful but not-too-powerful, we can "trust him to a larger extent" and can thus "critically receive" from him more theorems than before.

Achieving our goals. Let us now point out how CS proofs achieve our main goals for the notion of an efficient proof.

1. *Efficient verifiability.* Our first goal required that, for every theorem, verifying should be polylogarithmically easier than accepting. This goal is approximated by a CS proof system (P, V) in the following sense. Let L be a semirecursive language, x a member of L, and A an accepting algorithm for L. Then the theorem $x \in L$ can be verified by running A on input x and verifying that $A(x) = YES$. Assume now that the latter computation takes t steps and that (P, V) is a CS proof system with a random oracle. Then, by choosing a proper security parameter k and running P on inputs

[8] Roughly, for proving Theorem 3.5 it suffices that $2^k > poly(|t|)$, which it is implied by $2^k > poly(|q|)$, because $|t| < |q|$.

[9] In any case, he who considers more natural choosing t as an upperbound to $\#M(x)$ may notice that with minor changes all of the results of this paper, including those about CS checking, remain true. In particular, a CS proof that some $q = (M, x, y, T)$ belongs to the modified CS language is computable in $poly(|q|k\#M(x))$ even when $\#M(x) << T$.

$q = (A, x, YES, \#A(x))$ and 1^k, and access to a random oracle ρ, one obtains a CS certificate of $x \in L$, $\mathcal{C}$ that (a) is length-bounded by a fixed polynomial in $|q|$ and k, and (b) is accepted by V^ρ running on additional inputs q and 1^k. Therefore, because V is polynomial time (and because k is unary-presented), V accepts within a number of steps that are bounded by a fixed polynomial in $|q|$ and k. Thus, because $\#A(x)$ enters q in binary representation and because of our quadruple-encoding conventions, V accepts in time polynomial in $|x|$ and k (as well as in $|A|$) but polylogarithmic in $\#A(x)$.

2. *Efficient provability.* Our second goal called for the complexity of proving being polynomially close to that of accepting. This property is immediately guaranteed by the feasible completeness of a CS proof system. Feasible completeness in fact states that there exists a fixed constant c_2 such that, if an algorithm A accepts that a string x belongs to a semirecursive language L (in $\#A(x)$ steps), then a CS prover can, on inputs $q = (A, x, YES, \#A(x))$ and 1^k and with access to a random oracle, find a CS proof of $x \in L$ within $(|q|k\#A(x))^{c_2}$ steps. That is, a CS prover can find a CS proof of $x \in L$ in a time that is polynomial in $\#A(x)$ and k (as well as in $|A|$ and $|x|$).
3. *Recursive universality.* Our third goal called for proof systems capable of proving membership in all possible semirecursive languages. In apparent contrast with this requirement, a CS proof system is defined to prove membership only in the CS language $\mathcal{L}$. But $\mathcal{L}$ is designed so as to encode membership questions relative to any possible semirecursive language. In fact, to each semirecursive language L corresponds a Turing machine M_L so that $x \in L$ if and only if M_L, on input x, outputs YES in some number of steps t. Thus $x \in L$ if $(M_L, x, YES, t) \in \mathcal{L}$, thus achieving the third goal.

Efficient verifiability within $\mathcal{P}$. Note that a CS proof system with a random oracle makes verifying computationally preferable to accepting even for polynomial-time languages.

The process behind the curtains. In a classical proof system, what convinces us of the verity of a given statement is the existence of a string satisfying a proper syntactic property. By contrast, in an interactive proof system there are no strings that do the convincing: the "proof is in the process." Differently from both scenarios, in a noninteractive CS proof system, proofs are strings possessing a special property, but such strings may exist also for false statements. Therefore, what is convincing is not the existence of such strings but our belief that the *process* behind the generation/selection of such strings had computationally limited resources.[10]

Comparison with zero-knowledge arguments. Goldwasser, Micali, and Rackoff [21] introduced and first exemplified the notion of a zero-knowledge proof system, and Goldreich, Micali, and Wigderson[20] showed that all languages in $\mathcal{NP}$ possess

[10]Indeed, when debating whether a given statement is true, we do not have "serendipitous" access to some CS proof of it, if any. Thus, if we are given such a string, then there must have been an active process that generated/selected it. For instance, assume that, while walking on a beach pondering our favorite statement S, we encounter a sand pattern that looks like the sequence of bits of a CS proof, σ, of S. Then, we may consider that the grains of sand have been arranged in such a σ-shape by natural elements (such as wind, waves, and sun), and view the universe as a computer and its age as computing time, so that, in a final analysis, our σ has been found in a few billion years: an unlikely event if we have chosen our parameters so that the age of the universe is negligible with respect to the time necessary to find a good-looking CS proof of a false statement.

a (computationally) zero-knowledge proof, under a general complexity assumption.[11] Brassard, Chaum, and Crépeau [13] then put forward a related notion, that of a *zero-knowledge argument for* $\mathcal{NP}$, and proved a related theorem: under a specific complexity assumption, there exist zero-knowledge arguments for all $\mathcal{NP}$ languages.[12]

We wish to disregard the zero-knowledge component of the latter protocols and focus instead on their proof-system component, which we call the *argument (proper)*. Such arguments in fact provide an earlier example of proof systems which (as our CS proofs) are "convincing if its provers are *computationally bounded*." Let us explain.

In a (zero-knowledge) argument system for an $\mathcal{NP}$ language L, all agents, the prover P, the verifier V, and any possible malicious prover $\widetilde{P}$, are assumed to be polynomial-time machines. Before proving that a given input x belongs to L, P is *assumed* to have available, on a special tape inaccessible by V, an $\mathcal{NP}$-witness of $x \in L$, w. (Without this assumption, such a w might be uncomputable by the polynomial-time P.) During the protocol, P is provided by V with a special encryption scheme and uses it so as to convince V of the existence of w (without revealing it) by means of an interactive process that is less efficient than merely sending w. Vice versa, if $x \notin L$, no such w exists, and it is hard for a malicious $\widetilde{P}$ to convince V of the opposite. In fact, succeeding in such a malicious convincing entails "breaking" the provided encryption scheme, and the chance that a polynomial-time $\widetilde{P}$ may do that is quite remote.

Arguments, therefore, do not enlarge (nor aim at enlarging) the class of theorems that are efficiently provable. Rather, they constitute an alternative way of proving membership in $\mathcal{NP}$, a way that is less efficient than simply providing the witness but satisfies an additional property, *zero-knowledgeness* (which is in fact an integral part of their very definition).

Note that, leaving aside zero-knowledgeness and interaction (there are, after all, interactive CS proof systems), our CS proofs with a random oracle differ from the arguments of Brassard, Chaum, and Crépeau in the following ways.

- *Their arguments for $\mathcal{NP}$ may not enjoy efficient verifiability.* As discussed above, such $\mathcal{NP}$ arguments are less efficient than classical $\mathcal{NP}$ proof systems, and, as pointed out in subsection 2.1, the latter systems may not satisfy (ubiquitous efficiency, and thus) efficient verifiability.
- *Their arguments for $\mathcal{NP}$ may not enjoy efficient provability.* As discussed above, on input a member x of an $\mathcal{NP}$ language L, the prover of an $\mathcal{NP}$ argument is assumed to have "for free" (as an additional input) an $\mathcal{NP}$-witness w of $x \in L$. But, as we have pointed out in subsection 2.2, the time necessary for a prover to find such a witness w may vastly exceed that necessary to accept that $x \in L$.
- *Their arguments for $\mathcal{NP}$ do not enjoy recursive universality.* $\mathcal{NP}$ languages are a proper subset of all recursive languages.

3.3. The intuition behind our CS proof-system with a random oracle. Our construction is based on an earlier one of Kilian's [23], which is itself based on

[11] Informally, they exhibit interactive protocols enabling a prover to convince a polynomial-time verifier that an input belongs to an $\mathcal{NP}$ language L, but without conveying any more knowledge than the mere fact that a given witness of such a membership exists. Their protocols privilege the "proof aspect" rather than the "zero-knowledge aspect". Indeed, even if endowed with unbounded computational power, their provers cannot convince their verifiers that inputs outside L are in L (but with a negligible probability). However, for their verifiers to gain no information about inputs of L (other than their belonging to L) it is crucial that they be time-bounded.

[12] Differently from Goldreich, Micali, and Rackoff, rather then the proof aspect, they privilege the zero-knowledge aspect (see previous footnote).

Merkle's trees [27] and probabilistically checkable proofs [3, 17]. Let us thus start by recalling the latter two notions.

3.3.1. Probabilistically checkable proofs. Babai, Fortnow, Levin, and Szegedy [3], and Feige, Goldwasser, Lovasz, Safra, and Szegedy [17] have put forward, independently and with different aims,[13] some related and important ideas sharing a common technique: *proof-samplability.*[14] In essence, they present two algorithms. The first transforms an $\mathcal{NP}$-witness, w, into a slightly longer "samplable proof," w'. The second algorithm can check the correctness of such a string w' by "random" accessing (i.e., accessing at unit cost) selected few of its bits. In our paper, we refer to these two algorithms as the (sampling-enabling) prover and the (sampling) verifier, which we, respectively and consistently, denote by SP and SV, so as to differentiate them from other types of provers and verifiers.

The following version of their result has proved useful in most applications so far.

THEOREM 3.3 (samplable proofs: Version 0). *For all $\mathcal{NP}$ languages L there exist two polynomial-time algorithms, a deterministic SP, a probabilistic SV, and a polynomial Q, such that*

(a) *For all n-bit strings $x \in L$ and for all $\mathcal{NP}$-witness w of $x \in L$, $SP(x, w) = w'$, wherein string w' is such that, on input x and access to any $Q(\log n)$ bits of w' of its choice, algorithm SV accepts; and*

(b) *for all $x \notin L$ and for all w', algorithm SV, on input x and access to any $Q(\log n)$ bits of w' of its choice, rejects with probability $\geq 1/2$.*

We shall, however, rely on a more precise and general version of their result, namely the following theorem.

THEOREM 3.4 (samplable proofs: Version 1). *For any polynomial-time relation R over $\Sigma^* \times \Sigma^*$, there exist a deterministic polynomial-time algorithm $SP(\cdot, \cdot)$, a probabilistic polynomial-time algorithm $SV(\cdot, \cdot)$, and two polynomials $L(\cdot)$ and $\Lambda(\cdot)$, such that the following hold.*

1. *For all strings x and y such that $R(x, y)$ holds, $SP(x, y)$ outputs a string y' such that*
   1.1. *$y' < L(|x| + |y|)$, and*
   1.2. *$SV(x, |y'|)$, having a random tape of length $\Lambda(\log |y'|)$ and random access to y', accepts.*
2. *For all strings x such that for all y $R(x, y) = 0$, and for any string σ, the probability (computed over SV's coin tosses) that $SV(x, |\sigma|)$, having a random tape of length $\Lambda(\log |\sigma|)$, having random access to σ, and actually accessing $\Lambda(\log |\sigma|)$ bits of σ, accepts is $\leq 1/2$.*

In the statement of Theorem 3.4, as customary, inputs $|y'|$ and $|\sigma|$ are presented to SV in binary. (Thus the second input of SV is polylogarithmically shorter than $|x|+|y|$.) Note that, unlike in the case of $\mathcal{NP}$, $R(x, y) = 1$ may not imply any a priori

[13] The authors of [3] focus on proofs of membership in $\mathcal{NP}$ languages and show that it is possible to construct verifiers that work in time poly-logarithmic in the length of the input. (Since in such a short time the verifier could not even read the whole input—and thus check that the proof he is going to sample actually relates to the "right" theorem—these authors have devised a special error-correcting format for the input and assume that it is presented in that format. An input that does not come in that format can be put into it in polynomial-time.)

The authors of [17] use proof-samplability to establish the difficulty of finding approximate solutions to important $\mathcal{NP}$-complete problems. (With this goal in mind, these other authors do not mind verifiers working in time polynomial in the length of the input and do not use or need the fact that inputs appear in any special format.)

[14] Though improved in [2, 1, 35, 29], the original proof-samplability techniques of [3] and [17] suffice for our purposes.

bound for the length of y relative to that of x.

3.3.2. Merkle's trees. Recall that a binary tree is a tree in which every node has at most two children, hereafter called the 0-*child* and the 1-*child*. A *collision-free hash function* is, informally speaking, a polynomial-time computable function H mapping binary strings of arbitrary length into reasonably short strings, so that it is computationally infeasible to find any *collision (for H)*, that is, any two different strings x and y for which $H(x) = H(y)$. (Popular candidate collision-free hash function is the standardized *secure hash function* [32] and Rivest's MD4 [31].)

A *Merkle tree* [27] then is a binary tree whose nodes store (i.e., are associated to) values, some of which are computed by means of a collision-free hash function H in a special manner. A leaf node can store any value, but each internal node should store a value that is the one-way hash of the concatenation of the values in its children.[15] Thus, if the collision-free hash function produces k-bit outputs, each internal node of a Merkle tree, including the root, stores a k-bit value. Except for the root value, each value stored in a node of a Merkle tree is said to be a 0-value if it is stored in a node that is the 0-child of its parent, and a 1-value otherwise.

The crucial property of a Merkle tree is that, unless one succeeds in finding a collision for H, *it is computationally hard to change any value in the tree (and, in particular, a value stored in a leaf node) without also changing the root value.* This property allows a party A to "commit" to n values, $v_1, \ldots, v_n$ (for simplicity assume $n = 2^a$ for some integer a), by means of a single k-bit value. That is, A stores value v_i in the ith leaf of a full binary tree of depth d, and uses a collision-free hash function H to build a Merkle tree, thereby obtaining a k-bit value, rv, stored in the root. This root value rv "implicitly defines" what the n original values were. Assume in fact that, as some point in time, A gives rv, but not the original values, to another party B. Then, whenever, at a later point in time, A wants to "prove" to B what the value of, say, v_i was, he may just reveal all n original values to B, so that B can recompute the Merkle tree and then verify that the newly computed root-value indeed equals rv. More interestingly, A may "prove" what v_i was by revealing just $d+1$ (i.e., $\log n + 1$) values: v_i together with its *authentication path*, that is, the values stored in the siblings of the nodes along the path from leaf i (included) to the root (excluded), $Y_1, \ldots, Y_d$. Party B verifies the received alleged leaf-value v_i and the received alleged authentication path $Y_1, \ldots, Y_d$ as follows. She sets $X_1 = v_i$ and, letting $i_1, \ldots, i_d$ be the binary expansion of i, computes the values $X_2, \ldots, X_d$ as follows: if $i_j = 0$, she sets $X_{j+1} = H(Y_j X_j)$; otherwise, she sets $X_{j+1} = H(X_j Y_j)$. Finally, B checks whether the computed k-bit value X_d equals rv.

3.3.3. Kilian's construction. In [23], Kilian presents a special zero-knowledge argument for $\mathcal{NP}$, (P, V), exhibiting a polylogarithmic amount of communication, where prover P uses a Merkle tree in order to provide to V "virtual access" to a samplable proof.[16]

In essence, disregarding zero-knowledge aspects, the polynomial-time prover P, as in any zero-knowledge argument, possesses a polynomially long witness, w, proving

[15] i.e., if an internal node has a 0-child storing the value U and a 1-child storing a value V, then it stores the value $H(UV)$. If a child of an internal node does not exist, we assume by convention that it stores a special value, denoted by EMPTY.

[16] Essentially the same construction (minus its zero-knowledge aspects) was independently discovered by the author and privately comunicated to Shafi Goldwasser prior to Kilian's publication that same year. (It was not, however, written up or circulated until after Kilian's publication, and then only in the context of a broader notion of an efficient proof.)

that a given input x belongs to a given $\mathcal{NP}$-language L. In virtue of Theorem 3.3, P then transforms w into a longer, but still polynomially long in the length of x, "samplable proof" w' by running on inputs x and w the algorithm SP of Theorem 3.3. In order to yield more efficient verifiability, P cannot send V witness w, nor can he send him the longer samplable proof w'. Rather, P uses a Merkle tree with a collision-free hash function H, producing k-bit outputs, as above, to compute a k-bit string, rv, that commits him to w' and then sends rv to the verifier V. (For instance, disregarding further efficiency considerations, if w' is n-bit long and, for simplicity, n is a power of 2, the ith bit of w' is set to be the value v_i in the above described construction, the Merkle tree is a full binary tree of depth $\log n$, and rv is the k-bit value stored in its root.)

Verifier V runs as a subroutine the algorithm SV of Theorem 3.3. When SV wishes to consult the jth bit of w', V asks P for it, and P responds by providing the original value b_j together with its authentication path. V then checks whether b_j's authentication path is correct relative to rv, and, if so, he is assured that b_j is the original value because he trusts that P, being polynomial-time, cannot find a collision for H. V then feeds b_j to SV. The computation proceeds this way until V finds that an authentication path is incorrect, in which case it halts and rejects, or until SV halts, in which case V rejects if SV does and accepts otherwise. Because SV "virtually" accesses a polylogarithmic (in n) number of bits of w', and because each such a virtual access is answered by $k\,poly(\log n)$ bits of authentication path, the overall amount of communication is polylogarithmic in n and thus in the length of x.

Notice that the above construction only shows how a verifier can be given virtual access to w'. Let us reiterate that, in order to obtain a communication efficient zero-knowledge argument, Kilian's construction is actually more complicated, but the additional zero-knowledge constraint is irrelevant for our goals.

3.3.4. Our modifications. Like all prior argument systems, Kilian's is not a CS proof system (nor even an interactive one, as defined later on). To begin with, it only proves membership in $\mathcal{NP}$ languages and thus does not satisfy recursive universality. Further, even relative to the $\mathcal{NP}$ languages, it may not satisfy feasible completeness. Indeed, in his construction, in order to convince verifier V that x belongs to an $\mathcal{NP}$ language L, prover P needs an $\mathcal{NP}$-witness, w, of $x \in L$. But, again, the time necessary to compute w on input x may vastly exceed that necessary to accept (in this case, *decide*) that $x \in L$ (in a way that does not produce an $\mathcal{NP}$-witness).

We do, however, obtain a CS proof system with a random oracle, $(\mathcal{P}, \mathcal{V})$, by modifying his argument system. First, as a necessary step towards recursive universality, we assume that an input to $(\mathcal{P}, \mathcal{V})$ consists of a member of the CS language $\mathcal{L}$, (M, x, y, t).[17] On such a CS input, $(\mathcal{P}, \mathcal{V})$ works as follows. First, $\mathcal{P}$ runs machine M on input x so as to generate, in t steps, the history (i.e., sequence of instantaneous configurations), σ, of a computation of $M(x)$ in which string y is produced as an output. Such a history σ is then thought of as a proof that $M(x) = y$. This proof will not be insightful, and, because no restriction is put on M, can be arbitrarily long relative to x. (Notice that $\mathcal{P}$'s computation so far satisfies, by definition, feasible completeness.)

Next, $\mathcal{P}$ will put such a proof σ in samplable form. Consider in fact the following relation R.

[17] Again, in order to prove membership in a given semirecursive language L, M will then be a Turing machine accepting L, and y will be the special string YES.

$R(q, \sigma) = 1$ *if and only if* σ *is the* t*-step history of a computation of* M *outputting* y *on input* x.

Then, notice that R is $poly(|q|, |\sigma|)$-time computable. Thus, due to Theorem 3.4 proof σ can be put in a probabilistically checkable form τ by algorithm SP within $poly(|q|)$ —and thus $poly(t)$— steps. Given random access to the so obtained τ, algorithm SV, on inputs q and τ, then efficiently checks its correctness using polylog$(|\tau|)$ queries and a random tape, RT, whose length is $polylog(|\tau|)$. (Notice that also this second piece of $\mathcal{P}$'s computation satisfies feasible completeness.)

Though proving that "$M(x) = y$ in t steps," such τ is again too long. Thus, $\mathcal{P}$ "Merkle hashes" τ as in [23] using a collision-free hash function H and gives $\mathcal{V}$ only virtual access to it (something that still preserves feasible completeness). The verifier is thus guaranteed that he is properly accessing τ (i.e., that $\mathcal{P}$ is not choosing on-line the bits of τ based on the bit-locations that $\mathcal{V}$ wishes to access) *provided* that $\mathcal{P}$ is computationally incapable of finding a collision in H.[18] To provide such a "guarantee," for a *specific* input $q = (M, x, y, t) \in \mathcal{L}$, it is possible to choose the security parameter k big enough so that finding a collision in a k-bit-output H requires a number of steps enormously bigger than those required above from $\mathcal{P}$ on the input q at hand, but not too big so as to violate feasible completeness.[19] In sum, therefore, we propose to keep honest a prover working on a given *individual* problem by means of another much harder *individual* problem, that of finding a collision for H (though the latter problem may belong to a "much lower" complexity class than the first one[20]).

So far, our $(\mathcal{P}, \mathcal{V})$ satisfies both recursive universality and feasible completeness but still is interactive. Indeed, recall that, to give $\mathcal{V}$ virtual access to τ, $\mathcal{P}$ uses function H to Merkle-hash the samplable proof τ and sends $\mathcal{V}$ the resulting root value RV. In response, $\mathcal{V}$ runs the sampling verifier SV with a random tape RT. During this execution SV computes which bits of τ it wishes to see; CS verifier $\mathcal{V}$ then sends these requests to CS prover $\mathcal{P}$; and prover $\mathcal{P}$ replies with both the requested bits and their authentication paths relative to RV. Because RT is genuinely random, if the input $(M, x, y, t) \notin \mathcal{L}$ and if $\mathcal{V}$ interacts with a malicious prover $\widetilde{\mathcal{P}}$ that does not succeed in finding a collision for H, then SV (and thus $\mathcal{V}$) accepts with probability at most $1/2$.

Let us now introduce further modifications in order to dispense with any interaction between $\mathcal{P}$ and $\mathcal{V}$ during the proving process. We first decrease the probability of SV accepting a false statement to less than 2^{-k} by repeating the above process k times, each time using an independently-selected random tape RT. We then use a random oracle, as follows, to retain more or less this same probability, while eliminating any interaction between $\mathcal{P}$ and $\mathcal{V}$.

In some sense, we have the CS prover $\mathcal{P}$ "choose" the k random tapes of SV, so that $\mathcal{V}$ is no longer needed. In fact, given these tapes, $\mathcal{V}$ runs deterministically, and

[18] In Kilian's case such guarantee stemmed from the fact that the prover was polynomial-time, while collision finding is assumed not to be (and to enable him to prove membership in $\mathcal{NP}$-complete languages it was assumed that he had access to an $\mathcal{NP}$ witness for free). In our case, however, $\mathcal{P}$ cannot be assumed to be polynomial-time, because it ought to be able to run M on x for t steps for all possible $(M, x, y, t) \in \mathcal{L}$, and t may vastly exceed $|(M, x, y, t)|$.

[19] Assume, for instance, that the complexity of finding an H-collision for a k-bit output H is $\Omega(2^{k^d})$ for some constant d between 0 and 1. Then, because the honest prover works in time polynomial in t, setting $k = (\log t)^{2/d}$ seems a reasonable choice. This choice in fact increases only by a $poly((\log t)^{2/d})$ factor the amount of work of the honest prover but forces any malicious prover to work in time $2^{(\log t)^2}$.

[20] Accordingly, we view a prover working on a given input as an *individual* device, endowed with a fixed amount of computational resources, rather than a mechanism capable of handling all members of a given complexity class.

thus a prover $\mathcal{P}$ who knows them can simulate perfectly $\mathcal{V}$'s actions (and in particular compute the bit locations of τ that SV wishes to see). Of course, however, this a dangerous way of proceeding. In fact, it is already dangerous having a malicious CS prover $\widetilde{\mathcal{P}}$ simply *know* these k random tapes (let alone choose them), or even just predict the bit locations that SV wishes to see in its k runs. In fact:

> Letting λ denote the number of bit-locations of τ SV wishes to access in a single run, the total number of such bit-locations will be $k\lambda$. Now, if $k\lambda$ is sufficiently small with respect to the total number of bits in τ (which we would like to be our case in order to satisfy efficient verifiability), it is not hard to see that if $\widetilde{\mathcal{P}}$ knows in advance these $k\lambda$ bit-locations, then he could provide (1) a root value RV for the Merkle tree, (2) bit values for said locations, and (3) authetication paths for these values relative to RV, so as to cheat $\mathcal{V}$ with probability 1.

Notice too, however, that, if the k random tapes are selected *after* $\widetilde{\mathcal{P}}$ provides the root value RV, then, roughly said, unless he succeeds in finding at least one collision for H, his probability of cheating still is $< 2^{-k}$. (This continues to be true even if $\widetilde{\mathcal{P}}$ knows the so-selected k tapes in their entirety, rather than just the bit-requests that the sampling verifier computes from them.) This suggests replacing interaction in the above proof system as follows. On input $(M, x, y, t) \in \mathcal{L}$, prover $\mathcal{P}$, as before, (a) computes a classical proof of it by running M on input x and costructing a history σ of such computation, (b) puts σ in a samplable form τ, and (c) stores τ in the leaves of a suitable binary tree and constructs a corresponding Merkle tree, using a collision-free hash function H, so as to compute a root value RV. At this point, $\mathcal{P}$ uses the random oracle on input RV so as to compute k suitably-long random tapes, $RT_1, \ldots, RT_k$. He then runs ("in his head") verifier $\mathcal{V}$ and its subroutine SV as in the whole process described above for k times, using RT_i as SV's random tape in the ith iteration. Therefore, he computes (in his head) all the bit-locations of the samplable proof that SV requests to access. Then, it outputs, as a CS proof with a random oracle for $(M, x, y, t) \in \mathcal{L}$, the value RV and the requested bits, each with its own authentication path relative to RV. Such proof can be verified, in the obvious way, by using verifier $\mathcal{V}$ (with subroutine SV) and the same random oracle.

The intuition that this strategy works is quite strong. Consider a malicious prover trying to "CS-prove with a random oracle" a false statement. Of course, he can choose a root value RV' of his liking and consult the oracle so as to see whether he can produce a good-looking CS proof relative to RV'. However, roughly said, because for each RV' (as long as he does not succeed in finding a collision for the random oracle), his chance of finding a good-looking proof is at most 2^{-k}, we expect that he tries 2^k times before he succeeds. Thus, if k is large enough, and the running time of the malicious prover is properly and meaningfully upperbounded, his chance of finding a CS proof of a false statement is negligible. (Despite this simple and strong intuition, however, formally proving that this strategy works appears to be more difficult.)

Our strategy is reminiscent of a step used by Fiat and Shamir [18]. Indeed, they construct their digital signature scheme by starting with an interactive two-party protocol, in which the first party sends a first message to the second party and the second party responds with a random string, and then replacing the random message of the second party by evaluating a collision-free hash function on the first party's message. (By now, similar strategies have been discussed in the literature in many a context.)

As a final modification, in lieu of k-bit-output collision-free hash function H,

we construct our Merkle tree using a random oracle mapping $2k$-bit strings to k-bit ones. Indeed, finding collisions for random oracles is provably hard in a precisely quantifiable way, and adoption of such oracles also for this task dispenses us for relying on additional complexity assumptions (i.e., the existence of a collision-free H).

Note that the random oracle used for removing the interaction between prover and verifier and that used for building the Merkle tree had better be different. Alternatively, using standard techniques, one may use a single random oracle to "extract" two independent ones: one for each of these two tasks.

3.4. Description of $(\mathcal{P}, \mathcal{V})$: Our CS proof-system with a random oracle. Having presented all the ideas entering in our construction at an intuitive level, let us now proceed more formally.

3.4.1. Preliminaries.

From one oracle to two oracles. According to our definition, in a CS proof system with a random oracle, prover and verifier have oracle access to a single function f, where feasible completeness holds for any f, and computational soundness for a random f.

It will be easier, however, to exhibit a CS proof system with a random oracle $(\mathcal{P}, \mathcal{V})$ by having $\mathcal{P}$ and $\mathcal{V}$ have oracle access to two distinct functions, f_1 and f_2, where feasible completeness holds for any possible choice of f_1 and f_2, while computational soundness holds when f_1 and f_2 are random and independent.

Oracle access to these two functions can be simulated by accessing a single, properly selected, function f: to ensure that f_1 and f_2 are randomly and independently selected when f is random, it suffices to arrange that whenever $(i, x) \neq (j, y)$, no query made to f in order to compute $f_i(x)$ coincides with a query made to f in order to compute $f_j(y)$.[21]

From k runs to one run: Sampling proof systems and their length bounds. Rather than having the probability of successful cheating be less than $1/2$, let us restate Theorem 3.4 so as to reduce this probability to 2^{-k} by means of a sampling verifier that (at least formally) still uses a single random tape but receives an additional, independent security parameter.

THEOREM 3.5 (samplable proofs: Version 2). *There exists a deterministic polynomial-time algorithm $SP(\cdot, \cdot)$, a probabilistic polynomial-time algorithm $SV(\cdot, \cdot, \cdot)$, and two polynomials $L(\cdot)$ and $\Lambda(\cdot)$ such that, for any polynomial-time relation R over $\Sigma^* \times \Sigma^*$, the following two properties hold.*

1. *For all strings x and y such that $R(x, y)$ holds, $SP(x, y)$ outputs a string y' such that*
   - 1.1. *$y' < L(|x| + |y|)$, and*
   - 1.2. *for every security parameter k, $SV(x, |y'|, 1^k)$, having a random tape of length $k \cdot \Lambda(\log |y'|)$ and random access to y', accepts.*
2. *For all strings x such that for all y $R(x, y) = 0$ and for any string σ, the probability (computed over SV's coin tosses) that $SV(x, |\sigma|, 1^k)$, having a random tape of length $k \cdot \Lambda(\log |\sigma|)$, having random access to σ, and actually accessing $k \cdot \Lambda(\log |\sigma|)$ bits of σ, accepts is $\leq 2^{-k}$.*

The CS-history relation. In what follows we shall use Theorem 3.5 only for a specific relation $\mathcal{H}$, the *CS-history relation*, defined as follows.

[21] For instance, if, for $i = 1, 2$, $f_i : \{0,1\}^{a_i} \to \{0,1\}^{b_i}$ (for some positive integer values a_i and b_i, $i = 1, 2$), letting f map $\{0,1\}^{1+max(a_1+a_2)}$ into $\{0,1\}^{max(b_1+b_2)}$ allows us to achieve our goal quite straightforwardly.

$\mathcal{H}(q,h) = 1$ *if and only if string* $q = (M,x,y,t) \in \mathcal{L}$ *and string* h *is an encoding of the history of the execution of* M *on input* x.

Notice that, assuming the use of a proper encoding for these histories, not only is $\mathcal{H}$ polynomial-time computable, but there also is a fixed polynomial Q such that $\mathcal{H}(q,h) = 1$ implies $|h| \leq Q(2^{|q|})$. (In fact, our quadruple conventions imply that $|t| < |q|$, and thus that $t < 2^{|q|}$.) Let us now use the CS-history relation to restate Theorem 3.5 in the following form more directly useful to us.

THEOREM 3.6 (samplable proofs: Version 3). *There exists a deterministic polynomial-time algorithm* $SP(\cdot,\cdot)$, *a probabilistic polynomial-time algorithm* $SV(\cdot,\cdot)$, *and two polynomials* $\ell(\cdot)$ *and* $\lambda(\cdot)$ *such that, letting* $\mathcal{H}$ *be the history relation, the following two properties hold.*

1. *For all strings* q *and* h *such that* $\mathcal{H}(q,h) = 1$, $SP(q,h)$ *halts within* $\ell(|q|)$ *steps outputting a string* h' *such that*
   1.1. $\log |h'| < \ell(|q|)$, *and*
   1.2. *for every security parameter* k, $SV(q,|h'|,1^k)$, *having a random tape of length* $k \cdot \lambda(|q|)$ *and random access to* h', *accepts.*
2. *For all strings* q *such that for all* h $\mathcal{H}(x,h) = 0$, *for any security parameter* k, *and for any string* σ, *the probability (computed over* SV*'s coin tosses) that* $SV(q,|\sigma|,1^k)$, *having a random tape of length* $k \cdot \lambda(|q|)$, *having random access to* σ, *and actually accessing* $k \cdot \lambda(|q|)$ *bits of* σ, *accepts is* $\leq 2^{-k}$.

DEFINITION 3.7. *Let* SP, SV, ℓ, *and* λ *be as in Theorem* 3.6. *Then, we shall refer to* (SP,SV) *as a* sampling proof system (for the CS-history relation)*, and to* ℓ *and* λ *as its* length bounds *(respectively, for the samplable proof produced by* SP *and the number of queries and length of the random tape used by* SV*).*

Notation.

- *Basics.* We denote the empty word by ε, the set $\{0,1\}$ by Σ, the set of all natural numbers by $\mathcal{N}$, the set of all positive integers by Z^+, the concatenation of two strings x and y by $x|y$ (or more simply by xy), and the complement of a bit b by $\bar{b}$.
- *Strings.* If α is a binary string, then $|\alpha|$ denotes α's length; $\alpha_1 \cdots \alpha_i$ denotes α's i-bit prefix; and $\alpha_1 \cdots \bar{\alpha}_i$ denotes α's i-bit prefix with the last bit complemented.
- *Labeled trees.* If N is a power of two, we let $\mathcal{T}_N$ denote the complete binary tree with N leaves, whose vertices are labeled by binary strings whose lengths range from 0 to $\log N$ as follows. Vertex v_ε is the root, v_0 and v_1 are, respectively, its the left and right child, and, more generally, for all $i \in [0, \log N)$ and for all $\alpha \in \Sigma^i$, $v_{\alpha 0}$ and $v_{\alpha 1}$ are, respectively, the left and right child of node v_α. (Consequently, $v_{\alpha_1 \cdots \alpha_j}$ and $v_{\alpha_1 \cdots \bar{\alpha}_j}$ are siblings whenever $0 < |\alpha| \leq \log N$ and $0 < j \leq |\alpha|$.)
  The leaves of $\mathcal{T}_N$ are thought to be ordered "from left to right." Within the context of a tree $\mathcal{T}_N$, we denote by $[j]$ the $(\log_2 N)$-bit binary representation of integer j, with possible leading 0s. (Accordingly, the jth leaf of $\mathcal{T}_N$ is node $v_{[j]}$.)

3.4.2. Algorithms $\mathcal{P}$ and $\mathcal{V}$. Let us now describe two oracle-calling algorithms, $\mathcal{P}$ and $\mathcal{V}$, and then prove that they are, respectively, the prover and verifier of a CS proof system with a random oracle, $(\mathcal{P},\mathcal{V})$.

Common inputs: $q = (M,x,y,t)$, an n-bit (alleged) member of $\mathcal{L}$, and 1^k, a security parameter.

{*Comment:* For the purpose of the code of (honest) prover $\mathcal{P}$, $q = (M, x, y, t) \in \mathcal{L}$ and thus $t = \#M(x)$.}

Common subroutines: (SP, SV), a sampling proof system—as per Definition 3.7—with length bounds ℓ and λ.

Common oracles: $f_1 \in \Sigma^{2k} \to \Sigma^k$ and $f_2 \in \Sigma^{k+n} \to \Sigma^{k \cdot \lambda(n)}$.

{*Comment:* The first oracle, when randomly selected, is used as a collision-free hash function of a Merkle tree, by which $\mathcal{P}$ commits to a samplable proof of $q \in \mathcal{L}$. The second oracle, when randomly selected, is used to generate the random tape of sampling verifier SV.}

$\mathcal{P}$'s output: $\mathcal{C}$, a CS certificate that $q \in \mathcal{L}$.

$\mathcal{V}$'s additional input: $\mathcal{C}$.

ALGORITHM $\mathcal{P}$

$P1$. (Commit to a samplable proof of $q \in \mathcal{L}$.)

$P1.1$ (Find a proof, denoted by σ, of $x \in \mathcal{L}$.)
Run machine M on input x so as to output y in $\#M(x)$ steps and generate an encoding, σ, of M's computational history.
{*Comment:* σ can be considered a proof that $x \in \mathcal{L}$.}

$P1.2$ (Compute a samplable form, denoted by τ, of proof σ.)
$\tau \leftarrow SP(q, \sigma)$.
{*Comment:* Theorem 3.6, our quadruple encoding, and $n = |q|$ imply $|\tau| \le \ell(n)$.}

$P1.3$ (Commit to τ by means of a k-bit value R_ε.)
Assume, for simplicity only, that $|\tau|/k = N$, where N is an integral power of 2. Then, we shall associate to (figuratively speaking, "store in") each node v_α of a labeled tree $\mathcal{T}_N$ a value R_α computed as follows. Subdivide τ into the concatenation of N substrings, each k-bit long, $\tau = \tau_1 \cdots \tau_N$, and for $0 \le j < N$, assign to the jth leaf, $v_{[j]}$, the k-bit value

$$R_{[j]} = \tau_j. \tag{3.1}$$

Then, in a bottom-up fashion, assign to each interior node v_α of $\mathcal{T}_N$ the k-bit value

$$R_\alpha = f_1(R_{\alpha 0} | R_{\alpha 1}). \tag{3.2}$$

{*Comment:* R_ε thus is the k-bit value assigned to the root of $\mathcal{T}_N$. R_ε is considered a commitment to all values stored in the vertices of $\mathcal{T}_N$ and thus a comitment to all of τ.}

$P2$. (Build a CS certificate, $\mathcal{C}$, of $q \in \mathcal{L}$.)

$P2.1$ (Start building the CS certificate with the k-bit commitment R_ε as prefix.)
$\mathcal{C} \leftarrow R_\varepsilon$.

$P2.2$ (Choose a random tape, T, for SV.)
$T \leftarrow f_2(q | R_\varepsilon)$.

$P2.3$ (Run SV with random tape T and virtual access to τ.)
Run SV with random tape T, inputs q and $|\tau|$, and virtual access to τ. Whenever SV wishes to access bit-location i of τ, perform the following instructions.

$P2.3.1$ (Find the index, I, of the substring of τ containing b_i.)
$I \leftarrow \lceil i/k \rceil$;

$P2.3.2$ (Add leaf I to the CS certificate.)
$\mathcal{C} \leftarrow \mathcal{C}|R_{[I]}$; and

$P2.3.3$ (Add to the certificate the authentication path of leaf I.)
Set $\alpha = [I]$;
{*Comment:* $|\alpha| = \log_2 N$.}
Set $AUTHPATH_I = R_{\alpha_1 \cdots \bar{\alpha}_{\log N}} | \cdots | R_{\alpha_1 \bar{\alpha}_2} | R_{\bar{\alpha}_1}$;
{*Recall:* The authentication path of leaf I consists of the values stored in the siblings of the vertices of the path from leaf I to the root.}
$\mathcal{C} \leftarrow \mathcal{C}|AUTHPATH_I$.
{*Example:* if $N = 8$ and $I = 3$, then $[I] = 011$ and $AUTHPATH_I = (R_{010}, R_{00}, R_1)$.}

$P3$. (Output a certificate for $q \in \mathcal{L}$.)
Output $\mathcal{C}$.
{*Comment:* $\mathcal{C}$'s k-bit prefix is R_ε.}

ALGORITHM $\mathcal{V}$

$V1$. (Read and delete R_ε from certificate $\mathcal{C}$, and compute SV's random tape T.)
$ALLEGEDROOT \leftarrow \mathcal{C}_1 \cdots \mathcal{C}_k$;
$\mathcal{C} \leftarrow \mathcal{C}_{k+1} \cdots$; and
$T \leftarrow f_2(q|R_\varepsilon)$.

$V2$. (Run SV with random tape T, inputs q, $\ell(n)$ and 1^k, and virtual access to samplable proof τ.)
Execute $SV(q, \ell(n), 1^k)$ with random tape T. Whenever SV wishes to access bit-location i of the samplable proof, do the following.

$V2.1$ (Find the index, I, of the k-bit segment of τ containing b_i, and read the value of leaf I from the certificate.)
$I \leftarrow \lceil i/k \rceil$; $\alpha \leftarrow [I]$; and $R_\alpha \leftarrow \mathcal{C}_1 \cdots \mathcal{C}_k$.
Provide SV with the $(i - kI)$th bit of R_α.

$V2.2$ (Delete the value of leaf I from the certificate.) $\mathcal{C} \leftarrow \mathcal{C}_{k+1} \cdots$.

$V2.3$ (Check and remove from the certificate the authentication path of leaf I.)
For $m = 1$ to $\log N$,
$R_{\alpha_1 \cdots \bar{\alpha}_m} \leftarrow \mathcal{C}_1 \cdots \mathcal{C}_k$ and
$\mathcal{C} \leftarrow \mathcal{C}_{k+1} \ldots$.
For $m = \log N, \ldots, 1$, compute $R_{\alpha_1 \cdots \alpha_{m-1}}$ as follows:

$$R_{\alpha_1 \cdots \alpha_{m-1}} \leftarrow \begin{cases} f_1(R_{\alpha_1 \cdots \alpha_m} | R_{\alpha_1 \cdots \alpha_m^-}) & \text{if } \alpha_m = 1, \\ f_1(R_{\alpha_1 \cdots \alpha_m^-} | R_{\alpha_1 \cdots \alpha_m}) & \text{if } \alpha_m = 0 \end{cases}$$

and check whether the computed value R_ε equals the value $ALLEGEDROOT$.
{*Example:* If $N = 8$ and $I = 3$, then $[I] = 011$ and the verifier computes

$$\begin{aligned} R_{01} &= f_1(R_{010}|R_{011}), \\ R_0 &= f_1(R_{00}|R_{01}), \text{ and} \\ R_\varepsilon &= f_1(R_0|R_1), \end{aligned}$$

where values R_{011}, R_{010}, R_{00}, and R_1 are retrieved from $\mathcal{C}$.}

$V3$. (Accept if and only if SV accepts and the authentication path of each leaf is correct.)
If SV accepts and each $V2.3$ check is passed, output YES. Otherwise, output NO.

3.5. $(\mathcal{P}, \mathcal{V})$ works.

THEOREM 3.8. *$(\mathcal{P}, \mathcal{V})$ is a CS proof system with a random oracle.*

As per our Definition 3.2, to prove Theorem 3.8 we need to prove that $(\mathcal{P}, \mathcal{V})$ satisfies both feasible completeness and computational soundness.

3.5.1. Proof of feasible completeness. Adopting the two-oracle formulation of Definition 3.2 and recalling the domain and range of each of the two oracles in our construction of $(\mathcal{P}, \mathcal{V})$, what we need to prove is the following.

> *There exists $c_1, c_2, c_3 > 0$ such that for all $q = (M, x, y, t) \in \mathcal{L}$, for all k, for all $f_1 \in \Sigma^{2k} \to \Sigma^k$, and for all $f_2 \in \Sigma^{k+|q|} \to \Sigma^{k \cdot \lambda(|q|)}$:*
> (i) *$\mathcal{P}^{f_1, f_2}(q, 1^k)$ halts within $(|q|kt)^{c_2}$ computational steps, outputting a binary string $\mathcal{C}$ whose length is $\leq (|q|k)^{c_3}$, and*
> (ii) *$\mathcal{V}^{f_1, f_2}(q, 1^k, \mathcal{C}) = YES$.*

It is immediately seen that subproperty (ii) of feasible completeness holds. That is, for all $q = (M, x, y, t) \in \mathcal{L}$, for all security parameter k, and for all oracles f_1 and f_2, the certificate output by $\mathcal{P}$ convinces $\mathcal{V}$. Subproperty (i), that is, the fact that $\mathcal{P}$ performs only polynomially many (in n, k, and t) steps for producing a certificate, follows as easily. Indeed, prover $\mathcal{P}$ performs the following operations: (1) initially invests t steps of computation for running M on input x; (2) takes a number of steps polynomial in q's length (i.e., n) and t for computing the samplable proof τ; (3) makes less than $t \log t$ queries (each at unit cost) to the second random oracle for generating the Merkle tree; and, finally, (4) makes additional polynomially many steps for running $\mathcal{V}$ "in his head" and answering its queries so as to build the desired CS certificate.

Finally, let us argue that the length of $(\mathcal{P}, \mathcal{V})$'s certificates are in accordance to Definition 3.2. Namely, letting $q = (M, x, y, t)$ be a member of $\mathcal{L}$ and $\mathcal{C} = \mathcal{P}^{f_1, f_2}(q, 1^k)$, then $\mathcal{C}$'s length is polynomial in $|q|$ and k. To this end, notice that $\mathcal{C}$ contains a k-bit root value, plus one authentication path (in the constructed Merkle tree) for each bit that the samplable verifier SV wishes to access when run on inputs q and $|\tau|$ and (virtual) access to the samplable proof τ of "$M(x) = y$ in t steps." Now, because τ can be computed in a number of steps upperbounded by a fixed polynomial in $|q|$ and t, and because according to our conventions $t < 2^{|q|}$, it follows that the length of the binary representation of $|\tau|$ is upperbounded by some other fixed polynomial in $|q|$ alone. Therefore, because SV runs in polynomial time, the number of bits of τ it accesses (i.e., the number of authentication paths included in $\mathcal{C}$) is polynomial in $|q|$ alone. The claim about the length of $\mathcal{C}$ then follows from the fact that each authentication path contains a k-bit value for each level of the constructed Merkle tree and thus $k \log \tau < k|q|$ bits overall. □

3.5.2. Proof of computational soundness. Adopting the two-oracle formulation of Definition 3.2 and recalling the domain and range of each of the two oracles in our construction of $(\mathcal{P}, \mathcal{V})$, what we need to prove is the following.

> *There exist positive constants c_4, c_5, and c_6 such that for all $\widetilde{q} \notin \mathcal{L}$, for all k such that $2^k > |\widetilde{q}|^{c_4}$, and for all (cheating) deterministic $2^{c_5 k}$-call algorithm $\widetilde{P}$, for random oracles $\rho_1 \in \Sigma^{2k} \to \Sigma^k$ and $\rho_2 \in \Sigma^{k+|\widetilde{q}|} \to \Sigma^{k \cdot \lambda(|\widetilde{q}|)}$,*

$$Prob_{\rho_1;\rho_2}[V^{\rho_1,\rho_2}(\widetilde{q}, 1^k, \widetilde{P}^{\rho_1,\rho_2}(\widetilde{q}, 1^k)) = YES] \leq 2^{-c_6 k}.$$

We shall actually prove the following theorem.

THEOREM 3.9. *$(\mathcal{P}, \mathcal{V})$ satisfies the above condition for the following choice of c_4, c_5, and c_6:*

$$c_4 \stackrel{\text{def}}{=} 64c, \qquad c_5 \stackrel{\text{def}}{=} 1/8, \quad \text{and} \quad c_6 \stackrel{\text{def}}{=} 1/16,$$

where $c \stackrel{\text{def}}{=}$ the smallest positive integer C such that $n^C > \ell(n)$ for all integers $n > 1$.

Recall that ℓ is the first length-bound of our underlying sampling proof system (SP, SV).

Proof idea. In essence, the proof is by contradiction. Assume that $q \notin \mathcal{L}$, and that, nonetheless, we are given a cheating prover $\widetilde{\mathcal{P}}$ that has a nonnegligible probability of outputting a CS proof for $q \in \mathcal{L}$. Then, we derive a contradiction by using such $\widetilde{\mathcal{P}}$ to build a *samplable* proof for $q \in \mathcal{L}$. Let us explain.

By hypothesis, for many oracles ρ_1 and ρ_2, $\widetilde{\mathcal{P}}$ should be able to produce a CS proof $\mathcal{C}_{\rho_1,\rho_2}$ for $q \in \mathcal{L}$. Such string $\mathcal{C}_{\rho_1,\rho_2}$ allegedly includes the contents of a few bit-locations of an underlying samplable proof, τ_{ρ_1,ρ_2}. By varying ρ_1 and ρ_2, and looking at their corresponding $\mathcal{C}_{\rho_1,\rho_2}$, we obtain the contents of more and more bit-locations, until we discover the bits in all the locations having a nonnegligible probability of being queried by the sampling verifier on input q. Despite the fact that such contents are pieced together from different CS proofs (and thus potentially from different underlying samplable proofs), we shall prove that, with high probability, the discovered contents are consistent with a *single* samplable proof τ.

Local definition 1.

- *Probabilities.* Let $S_1, S_2, \ldots,$ be finite sets, and let E be an event. Then, by $PROB_{x_1 \in S_1; x_2 \in S_2; \ldots}[E]$ we denote the probability of E in the experiment consisting of selecting elements $x_1 \in S_1, x_2 \in S_2, \ldots$ randomly and independently. If, for some x_i, it is already clear that x_i ranges in S_i, we may omit specifying S_i and more simply denote the same probability by $PROB_{\ldots;x_i;\ldots}[E]$.
- *Pseudoexecutions.* We shall consider executing a cheating, N-call, prover $\widetilde{\mathcal{P}}^{(\cdot,\cdot)}$ by answering its queries to the first oracle by means of a function f, and its queries to the second oracles by means of a predetermined, N-long, sequence S (i.e., the ith query to the second oracle will be answered with the ith element of S). We shall call such a process a *pseudoexecution (of $\widetilde{\mathcal{P}}$)*, or an execution of *algorithm* $\widetilde{\mathcal{P}}^{f,S}$. When $\widetilde{\mathcal{P}}^{f,S}$ is run on an n-bit input ($q \notin \mathcal{L}$) and security parameter k, then each element of S will consist of a $k\lambda(n)$-bit string (i.e., a possible random tape for $\mathcal{V}$). If σ is a string and m an integer between 1 and N, by the expression $\widetilde{\mathcal{P}}^{f,S_m=\sigma}$ we denote the algorithm identical to $\widetilde{\mathcal{P}}^{f,S}$, except that the mth query to the second oracle is answered by σ (i.e., the mth element of S—no matter what it originally was—is "forced" to be σ). By the expression $\widetilde{\mathcal{P}}^{f,S_m=\sigma_1,\sigma_2}$ we denote the algorithm that first executes $\mathcal{P}^{f,S_m=\sigma_1}$ and then $\mathcal{P}^{f,S_m=\sigma_2}$.
- *Collisions.* Let f be an oracle, $A(\cdot)$ an oracle-calling algorithm, and z an input. Then, by the expression *an f-collision in* $A^f(z)$, we mean that executing A on z with oracle f, A queries f about two distinct strings a and b and obtains the same string, c $(= f(a) = f(b))$, in response.
- *Pseudocertificates and pseudoroots.* Without loss of generality, we assume that a cheating prover $\widetilde{\mathcal{P}}$ never asks the same query twice to the same oracle. Again, without loss of generality, we assume that each cheating prover $\widetilde{\mathcal{P}}$ verifies all its nonempty outputs. That is, if $\widetilde{\mathcal{C}}$ is a nonempty string and $\widetilde{\mathcal{P}}^{f_1,f_2}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{\mathcal{C}}$, then, prior to outputting $\widetilde{\mathcal{C}}$, $\widetilde{\mathcal{P}}$ runs $\mathcal{V}$ making all required calls to f_1 and f_2 so as to verify that $\mathcal{V}^{f_1,f_2}(\widetilde{q}, 1^{\widetilde{k}}, \widetilde{\mathcal{C}}) = YES$, and thus that $\widetilde{\mathcal{C}}$ is a CS certificate for $\widetilde{q}$. If $\widetilde{q}$ does not belong to the CS language $\mathcal{L}$, to emphasize this fact we refer to $\widetilde{\mathcal{C}}$ itself as a *pseudocertificate (for $\widetilde{q}$)* and to

its $\widetilde{k}$-bit prefix as a *pseudoroot*.

In an execution of a cheating prover $\widetilde{\mathcal{P}}$ producing a pseudocertificate $\widetilde{\mathcal{C}}$ for $\widetilde{q}$, we say that $\widetilde{\mathcal{C}}$ is *relative to tape-number* m if (1) the pseudoroot of $\widetilde{\mathcal{C}}$ is a string $\widetilde{R}_\varepsilon$ such that the mth query of $\widetilde{\mathcal{P}}$ to its second oracle consists of the pair $(\widetilde{q}, \widetilde{R}_\varepsilon)$.

To indicate a generic pseudocertificate having pseudoroot $\widetilde{R}_\varepsilon$, we write $\widetilde{R}_\varepsilon \ldots$.

A proof by contradiction. We proceed by contradiction. Assume that Theorem 3.9 is incorrect; then, because $\widetilde{\mathcal{P}}$ verifies all its nonempty outputs, the following proposition holds.

PROPOSITION 3.10. *There exist an integer $\widetilde{n} > 1$, a $\widetilde{n}$-bit string $\widetilde{q} \notin \mathcal{L}$, an integer $\widetilde{k}$ such that $2^{\widetilde{k}} > \widetilde{n}^{64c}$, and a deterministic, $2^{\widetilde{k}/8}$-call, cheating prover $\widetilde{\mathcal{P}}$ such that, for random oracles $\rho_1 \in \Sigma^{2\widetilde{k}} \to \Sigma^{\widetilde{k}}$ and $\rho_2 \in \Sigma^{\widetilde{k}+\widetilde{n}} \to \Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}$,*

(P1) $PROB_{\rho_1;\rho_2}[\widetilde{\mathcal{P}}^{\rho_1,\rho_2}(\widetilde{q}, 1^{\widetilde{k}}) \neq \varepsilon] > 2^{-\widetilde{k}/16}$.

We now show that Proposition 3.10 contradicts the fact that $(\mathcal{P}, \mathcal{V})$'s subroutine (SP, SV) is a sampling proof system. We start by stating without proof some easy probabilistic facts.

Basic lemmas. Let A and B be finite sets, $A \times B$ their Cartesian product, and E a subset of $A \times B$. Then, in the following two lemmas $PROB_{a;b}[(a,b) \in E]$ denotes the probability that (a,b) belongs to E by selecting uniformly and independently a in A and b in B, and $PROB_b[(a,b) \in E]$ denotes the probability that (a,b) belongs to E by selecting uniformly b in B.

LEMMA 3.11. *Assume $PROB_{a;b}[(a,b) \in E] > x$, and let $G = \{a : PROB_b[(a,b) \in E] > 2^{-1} \cdot x\}$. Then,*

$$PROB_a[a \in G] > x/2.$$

LEMMA 3.12. *Assume $PROB_{a;b}[(a,b) \in E] < x$, and let $L = \{a : PROB_b[(a,b) \in E] < nx\}$. Then,*

$$PROB_a[a \in L] > 1 - n^{-1}.$$

LEMMA 3.13. *For all positive integers k and N, for all N-call algorithms $\mathcal{A}(\cdot)$, and for all inputs z,*

$$PROB_{f \in \Sigma^{2k} \to \Sigma^k}[\text{ an } f\text{-collision in } \mathcal{A}^f(z)] < N^2 2^{-k}.$$

Note that Lemma 3.13 continues to hold if A has additional inputs and oracles, provided that f is randomly selected independently of them.

An averaging argument.

LEMMA 3.14. *Let $\widetilde{n}$, $\widetilde{q}$, and $\widetilde{k}$ be as in Proposition 3.10. Then, there exist an oracle $f_1 \in \Sigma^{2\widetilde{k}} \to \Sigma^{\widetilde{k}}$, a $2^{\widetilde{k}/8}$-long sequence S of $\widetilde{k} \cdot \lambda(\widetilde{n})$-bit strings, an integer $m \in [1, 2^{\widetilde{k}/8}]$, and a $\widetilde{k}$-bit (pseudoroot) $\widetilde{R}_\varepsilon$ such that*

(L1.1) $PROB_{\sigma \in \Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots] > 2^{-1} \cdot 2^{-3\widetilde{k}/16}$, and

(L1.2) $PROB_{\sigma_1, \sigma_2 \in \Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma_1}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots = \widetilde{\mathcal{P}}^{f_1, S_m = \sigma_2}(\widetilde{q}, 1^{\widetilde{k}}) \wedge f_1\text{-collision}$ in $\mathcal{P}^{f_1, S_m = \sigma_1, \sigma_2}(\widetilde{q}, \widetilde{k})] < 32 \cdot 2^{-9\widetilde{k}/16}$.

Proof. Let ρ_1 and ρ_2 be oracles as in Proposition 3.10. Then, because $\widetilde{\mathcal{P}}^{\rho_1,\rho_2}$ verifies its nonempty outputs (and because it is $2^{\widetilde{k}/8}$-call), each of its pseudocertificates

is relative to some tape-number between 1 and $2^{\widetilde{k}/8}$. Thus, inequality P1 implies that there exists a positive integer $m \in [1, 2^{\widetilde{k}/8}]$ such that

(L1.3) $PROB_{\rho_1;\rho_2}[\widetilde{\mathcal{P}}^{\rho_1,\rho_2}(\widetilde{q}, 1^{\widetilde{k}}) \neq \varepsilon \ \wedge \ \widetilde{\mathcal{P}}^{\rho_1,\rho_2}(\widetilde{q}, 1^{\widetilde{k}})$ is relative to tape-number $m] \geq 2^{-\widetilde{k}/16}/2^{\widetilde{k}/8} = 2^{-3\widetilde{k}/16}$.

Therefore, by focusing our attention on tape-number m (i.e., on the mth answer of our random oracle $\rho_2 \in \Sigma^{\widetilde{k}+\widetilde{n}} \to \Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}$) and by averaging, inequality L1.3 implies that there exists a $2^{\widetilde{k}/8}$-long sequence, S, of $\widetilde{k}\cdot\lambda(\widetilde{n})$-bit strings (i.e., of possible second-oracle answers), such that

(L1.4) $PROB_{\rho_1;\sigma\in\Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}}) \neq \varepsilon \ \wedge \ \widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}})$ is relative to tape-number $m] > 2^{-3\widetilde{k}/16}$.

Define now an oracle $\rho_1 : \Sigma^{2\widetilde{k}} \to \Sigma^{\widetilde{k}}$ to be *good* if

$PROB_{\sigma\in\Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}}) \neq \varepsilon \ \wedge \ \widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}})$ is relative to tape-number $m] > 2^{-1}\cdot 2^{-3\widetilde{k}/16}$.

Then, inequality L1.4 and Lemma 3.11 imply that

(L1.5) $PROB_{\rho_1}[\rho_1 \text{ good }] > 2^{-1}\cdot 2^{-3\widetilde{k}/16}$.

Note now that because both $\widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma_1}$ and $\widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma_2}$ make at most $2^{\widetilde{k}/8}$ oracle calls, algorithm $\widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma_1,\sigma_2}$ makes at most twice as many calls to ρ_1. Thus, Lemma 3.13 implies that

(L1.6) $PROB_{\rho_1\in\Sigma^{2\widetilde{k}}\to\Sigma^{\widetilde{k}};\sigma_1,\sigma_2\in\Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\rho_1\text{-collision in } \widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma_1,\sigma_2}(\widetilde{q}, 1^{\widetilde{k}})] < 4\cdot 2^{\widetilde{k}/4}\cdot 2^{-\widetilde{k}} = 4\cdot 2^{-3\widetilde{k}/4}$.

Define now an oracle $\rho_1 : \Sigma^{2\widetilde{k}} \to \Sigma^{\widetilde{k}}$ to be *lucky* if

$$PROB_{\sigma_1,\sigma_2\in\Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\rho_1\text{-collision in } \widetilde{\mathcal{P}}^{\rho_1,S_m=\sigma_1,\sigma_2}(\widetilde{q}, 1^{\widetilde{k}})] < (8\cdot 2^{3\widetilde{k}/16})\cdot(4\cdot 2^{-3\widetilde{k}/4}) = 32\cdot 2^{-9\widetilde{k}/16}.$$

Then, inequality L1.6 and Lemma 3.12 imply that

(L1.7) $PROB_{\rho_1}[\rho_1 \text{ lucky }] > 1 - 8^{-1}\cdot 2^{-3\widetilde{k}/16} > 1 - 2^{-1}\cdot 2^{-3\widetilde{k}/16} \geq 1 - PROB_{\rho_1}[\rho_1 \text{ good}] = PROB_{\rho_1}[\rho_1 \text{ not good}]$.

Because inequality L1.7 implies that there exist oracles in $\Sigma^{2\widetilde{k}} \to \Sigma^{\widetilde{k}}$ that are both good and lucky, let f_1 be one such oracle. Then, because F_1 is good, letting F_1 be oracle f_1 of Lemma 3.14 satisfies inequality L1.1. In fact, L1.1 simply states that f_1 is a good oracle. Let us now show that there exists a k-bit value $\widetilde{R}_\varepsilon$ such that letting F_1 be oracle f_1 of Lemma 3.14 satisfies inequality L1.2. In fact, notice that, for any possible choice of σ, the computation of $\widetilde{\mathcal{P}}^{F_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}})$ is always identical up to its mth query to the second oracle, and thus that there exists a $\widetilde{k}$-bit value $\widetilde{R}_\varepsilon$ such that all mth queries consist of the same pair $(\widetilde{q}, \widetilde{R}_\varepsilon)$. Therefore, whenever an execution of $\widetilde{\mathcal{P}}^{f_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}})$ produces a nonempty pseudocertificate with respect to tape-number m, $\widetilde{R}_\varepsilon$ will be the pseudoroot of this certificate. Now, because F_1 is lucky, we have

$PROB_{\sigma_1,\sigma_2\in\Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[\widetilde{\mathcal{P}}^{F_1,S_m=\sigma_1}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots = \widetilde{\mathcal{P}}^{F_1,S_m=\sigma_2}(\widetilde{q}, 1^{\widetilde{k}}) \ \wedge \ F_1\text{-collision in } \widetilde{\mathcal{P}}^{F_1,S_m=\sigma_1,\sigma_2}(\widetilde{q}, 1^{\widetilde{k}})]$

$< PROB_{\sigma_1,\sigma_2\in\Sigma^{\widetilde{k}\cdot\lambda(\widetilde{n})}}[F_1\text{-collision in } \widetilde{\mathcal{P}}^{F_1,S_m=\sigma_1,\sigma_2}(\widetilde{q}, 1^{\widetilde{k}})] <$ (because F_1 is lucky) $32\cdot 2^{-9\widetilde{k}/16}$. □

A collision argument.

Local definition 2. If $\widetilde{\mathcal{C}}$ is a pseudocertificate, we write $\widetilde{\mathcal{C}} \ni (i,b)$ if $\widetilde{\mathcal{C}}$ indicates that location i (in the samplable proof that is allegedly Merkle-hashed to the pseudoroot of $\widetilde{\mathcal{C}}$) contains bit b.

LEMMA 3.15. *Let $\widetilde{n}$, $\widetilde{q}$, $\widetilde{k}$, $\widetilde{\mathcal{P}}$, f_1, S, m, and $\widetilde{R}_\varepsilon$ be as in Lemma* 3.14, *let σ_0 and σ_1 be two distinct $k \cdot \lambda(\widetilde{n})$-bit strings, and let E_0 be the execution of $\widetilde{\mathcal{P}}^{f_1,S_m=\sigma_0}(\widetilde{q}, 1^{\widetilde{k}})$ and E_1 the execution of $\widetilde{\mathcal{P}}^{f_1,S_m=\sigma_1}(\widetilde{q}, 1^{\widetilde{k}})$. Now, if E_0 and E_1 produce two pseudocertificates, respectively, $\mathcal{C}_0$ and $\mathcal{C}_1$, such that* (1) *both $\mathcal{C}_0$ and $\mathcal{C}_1$ have pseudoroot $\widetilde{R}_\varepsilon$ and* (2) *for some common location i, $\mathcal{C}_0 \ni (i,0)$ and $\mathcal{C}_1 \ni (i,1)$, then an f_1-collision occurs in $\widetilde{\mathcal{P}}^{f_1,S_m=\sigma_0,\sigma_1}(\widetilde{q}, 1^{\widetilde{k}})$.*

Proof. Recall that, supposedly, the underlying samplable-proof of $\widetilde{q} \in \mathcal{L}$ has been divided into N substrings, each $\widetilde{k}$-bit long and stored in a separate leaf of a Merkle tree of depth $\log N$. Thus bit-location i should be stored in the Ith leftmost leaf of the Merkle tree, where $I = \lceil i/\widetilde{k} \rceil$, or, equivalently, in the node whose $\log N$-bit name is $[I] = \alpha_1 \cdots \alpha_{\log N}$. According to our notation, the value stored in this node is denoted by $R_{[I]}$. Now, because they indicate different bit-values for location i, pseudocertificates $\mathcal{C}_0$ and $\mathcal{C}_1$ also indicate different $\widetilde{k}$-bit values for the content of node $[I]$, respectively, $R^0_{[I]}$ and $R^1_{[I]}$.

Pseudocertificates $\mathcal{C}_0$ and $\mathcal{C}_1$ also include two authentication paths for these values. Let them be, respectively,

$$R^0_{\alpha_1\cdots\alpha_{\log N-1}\bar{\alpha}_{\log N}} | R^0_{\alpha_1\cdots\bar{\alpha}_{\log N-1}} | \cdots | R^0_{\bar{\alpha}_1} | \widetilde{R}_\varepsilon$$
and
$$R^1_{\alpha_1\cdots\alpha_{\log N-1}\bar{\alpha}_{\log N}} | R^1_{\alpha_1\cdots\bar{\alpha}_{\log N-1}} | \ldots, R^1_{\bar{\alpha}_1} | \widetilde{R}_\varepsilon.$$

Because cheating prover $\widetilde{\mathcal{P}}$ verifies all its outputs, for each $j \in [1, \log N]$ it queries oracle f_1 about string S^0_j in the first execution, and about string S^1_j in the second execution, where $S^0_j = R^0_{\alpha_1\cdots\alpha_j} | R^0_{\alpha_1\cdots\bar{\alpha}_j}$ and $S^1_j = R^1_{\alpha_1\cdots\alpha_j} | R^1_{\alpha_1\cdots\bar{\alpha}_j}$.

This implies that there exists a value $j \in [1, \log N]$ such that S^0_j and S^1_j are different queries but oracle f_1 returns the same answer on them. In fact, for $j = 1$, the two queries $S^0_1 (= R^0_{\alpha_1} | R^0_{\bar{\alpha}_1})$ and $S^1_1 (= R^2_{\alpha_1} 1 | R^1_{\bar{\alpha}_1})$ are both answered by $\widetilde{R}_\varepsilon$, and for $j = \log N$ the two queries $S^0_{\log N} (= R^0_{[I]})$ and $S^1_{\log N} (= R^1_{[I]})$ are different, because they coincide with the two different values for leaf I. □

Reaching the desired contradiction.

Local definition 3. Letting $\widetilde{n}$, $\widetilde{q}$, $\widetilde{k}$, S, m, and $\widetilde{R}_\varepsilon$ be as in Lemma 3.14, define the following.

- *Conditional probabilities $P_{i,0}$ and $P_{i,1}$.* For each bit-location $i \in [1, \ell(\widetilde{n})]$, define
  $$P_{i,0} = PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}} [\widetilde{\mathcal{P}}^{f_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots \wedge \widetilde{R}_\varepsilon \cdots \ni (i,0)]$$
  and
  $$P_{i,1} = PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}} [\widetilde{\mathcal{P}}^{f_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots \wedge \widetilde{R}_\varepsilon \cdots \ni (i,1)].$$
- *String τ.* For each location $i \in [1, \ell(\widetilde{n})]$, define
  $\tau_i = 0$ if $P_{i,0} \geq P_{i,1}$ and $\tau_i = 1$ otherwise.
- *Probabilities P_i and $P_{\bar{i}}$.* For each bit-location $i \in [1, \ell(\widetilde{n})]$, define
  $P_i = P_{i,\tau_i}$ and $P_{\bar{i}} = P_{i,\bar{\tau}_i}$.
- *Event "all queries answered by τ."*
  If $\sigma \in \widetilde{\Sigma^{k \cdot \lambda(\widetilde{n})}}$, in a pseudoexecution of $\widetilde{\mathcal{P}}^{f_1,S_m=\sigma}(\widetilde{q}, 1^{\widetilde{k}})$ with no f_1-collision and producing a pseudocertificate $\widetilde{\mathcal{C}}$ with pseudoroot $\widetilde{R}_\varepsilon$, "all queries answered

by τ" denotes the event that, for all bit-locations $i \in [1, \ell(\widetilde{n})]$ and for all bit b, $\widetilde{\mathcal{C}} \ni (i,b)$ implies $b = \tau_i$.

LEMMA 3.16. *Let* $\widetilde{n}$, $\widetilde{q}$, $\widetilde{k}$, $\widetilde{\mathcal{P}}$, f_1, S, m, *and* $\widetilde{R}_\varepsilon$ *be as in Lemma* 3.14 *Then,*

$$PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots \wedge \text{ all queries answered by } \tau] > 2^{-\widetilde{k}}.$$

Proof. We start by claiming that

(L3.1) $P_{\bar{i}} < 4 \cdot 2^{-9\widetilde{k}/32}$.

To prove our claim, consider selecting two $\widetilde{k} \cdot \lambda(\widetilde{n})$-bit strings, σ_0 and σ_1, and then executing $\widetilde{\mathcal{P}}^{f_1, S_m = \sigma_0, \sigma_1}(\widetilde{q}, 1^{\widetilde{k}})$. That is, consider executing first $\widetilde{\mathcal{P}}^{f_1, S_m = \sigma_0}(\widetilde{q}, 1^{\widetilde{k}})$ and then $\widetilde{\mathcal{P}}^{f_1, S_m = \sigma_1}(\widetilde{q}, 1^{\widetilde{k}})$. Then, by definition of P_i and $P_{\bar{i}}$, with probability $\geq 2P_iP_{\bar{i}}$, one of the latter two executions outputs a pseudocertificate $\mathcal{C}_0$ and the other a pseudocertificate $\mathcal{C}_1$ such that (1) $\mathcal{C}_0$ and $\mathcal{C}_1$ are nonempty pseudocertificates having pseudoroot $\widetilde{R}_\varepsilon$, and (2) $\mathcal{C}_0 \ni (i,0)$ and $\mathcal{C}_1 \ni (i,1)$. By Lemma 3.15, (1) and (2) imply that, with probability $\geq 2P_iP_{\bar{i}}$ (taken over the choices of σ_1 and σ_2), (3) an f_1-collision occurs in $\widetilde{\mathcal{P}}^{f_1, S_m = \sigma_0, \sigma_1}(\widetilde{q}, 1^{\widetilde{k}})$. Thus, by Lemma 3.14 (inequality L1.2), we have $2P_iP_{\bar{i}} < 32 \cdot 2^{-9\widetilde{k}/16}$. Now, because $P_{\bar{i}} \leq P_i$ by definition, we have $2P_{\bar{i}}^2 \leq 2P_iP_{\bar{i}} < 32 \cdot 2^{-9\widetilde{k}/16}$, and thus $P_{\bar{i}} < 4 \cdot 2^{-9\widetilde{k}/32}$ as initially claimed.

Define now $P_{\bar{\tau}}$ as the probability that "$\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots$, but *not* all queries answered by τ", that is,

$$P_{\bar{\tau}} \stackrel{\text{def}}{=} PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots \wedge \exists i \widetilde{R}_\varepsilon \cdots \ni (i, \bar{\tau}_i)].$$

Then, because there are at most $\ell(\widetilde{n})$ bit-locations i, and because (due to Proposition 3.10 and our definition of c) $\ell(\widetilde{n}) < (\widetilde{n})^c < 2^{\widetilde{k}/64}$, we have

(L3.2) $P_{\bar{\tau}} \leq \sum_{i=1}^{\ell(\widetilde{n})} P_{\bar{i}} < 4 \cdot 2^{-9\widetilde{k}/32} \cdot \ell(\widetilde{n}) < 4 \cdot 2^{-9\widetilde{k}/32} \cdot 2^{\widetilde{k}/64} = 4 \cdot 2^{-17\widetilde{k}/64}$.

Thus

(L3.3) $PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots \wedge \text{ all queries answered by } \tau]$

$\geq PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots] - P_{\bar{\tau}} \geq$ (by inequalities L1.1 and L3.2) $2^{-1} \cdot 2^{-3\widetilde{k}/16} - 4 \cdot 2^{-17\widetilde{k}/64} >$ (because $2^{\widetilde{k}} > n^{64c}$, $n \geq 2$ and $c \geq 1$ imply $\widetilde{k} > 64$) $2^{-\widetilde{k}}$. ∎

Notice now that Lemma 3.16 contradicts the fact that the underlying (SP, SV) is a sampling proof system according to Definition 3.7. In fact, because $\widetilde{\mathcal{P}}$ verifies all its nonempty outputs, we have

$2^{-\widetilde{k}} <$ (because of inequality L3.3) $PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}}[\widetilde{\mathcal{P}}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}) = \widetilde{R}_\varepsilon \cdots \wedge \text{ all queries answered by } \tau] = PROB_{\sigma \in \widetilde{\Sigma^{k \cdot \lambda(n)}}}[\mathcal{V}^{f_1, S_m = \sigma}(\widetilde{q}, 1^{\widetilde{k}}, \widetilde{R}_\varepsilon \cdots) = YES \wedge \text{ all queries answered by } \tau]$.

But, by our construction of $\mathcal{V}$, the last inequality implies that, by running sampling verifier SV on inputs $\widetilde{q}$ ($\notin \mathcal{L}$), $|\tau|$ (i.e., $\widetilde{n}$), and $\widetilde{k}$, having a random tape of length $\widetilde{k} \cdot \lambda(|\tau|)$, and having random access to τ, SV accepts with probability $> 2^{-\widetilde{k}}$.

Because $\widetilde{q} \notin \mathcal{L}$, the existence of τ contradicts property 2 of Theorem 3.6. The contradiction establishes Theorem 3.9, and thus that our $(\mathcal{P}, \mathcal{V})$ enjoys computational soundness, completing our proof of Theorem 3.9. ∎

3.6. The significance of CS proofs with a random oracle. Random oracles may be quite theoretical, and, as discussed later on, one might consider implementing CS proofs cryptographically (with or without interaction). But the latter implementations would be meaningless if it turned out that $\mathcal{P} = \mathcal{NP}$. This would not be too

bad, one might say, because, if $\mathcal{P} = \mathcal{NP}$, the very notion of an "efficient" proof system would be meaningless, at least in any broad sense.[22]

Personally, we disagree: fundamental intuitions such as proofs being a notion that is both meaningful and separate (from that of accepting[23]) could not be shaken by a formal result such as $\mathcal{P} = \mathcal{NP}$. Indeed, the author's inclination to believe that $\mathcal{P} \neq \mathcal{NP}$ is only based on (1) his a priori *certainty* that proofs are a meaningful and separate notion, and (2) his *inclination* to believe that $\mathcal{NP}$ is a reasonable approximation of the notion of a proof. But if it turned out that $\mathcal{P} = \mathcal{NP}$, to the author this would only mean that $\mathcal{NP}$ did not provide such a reasonable approximation after all.

It is thus important to establish meaningful models for which we can *show* that *proofs do exist as an independent notion.* CS proofs with a random oracle provide us with such a model: indeed, they guarantee that

> *even if $\mathcal{NP} = \mathcal{P}$, given a sufficient amount of randomness in the proper form, fundamental intuitions like verification being polylogarithmically easier than decision are indeed true.*

4. Other types of CS proofs. Many variants of the basic notion of a CS proof exist. Below, we confine ourselves to briefly presenting just two additional ones: that of an interactive CS proof (because it can be implemented based on standard cryptographic assumptions) and that of a noninteractive CS proof (because it implies the existence of CS checkers for $\mathcal{NP}$-complete problems).

4.1. Interactive CS proofs.

The notion of an interactive CS proof system. Let us first quickly recall interactive Turing machines (ITMs) as defined by Goldwasser, Micali, and Rackoff [21]. Informally, an ITM is a probabilistic Turing machine capable of "sending and receiving messages." An ITM is meant to be run a number of times, each time starting with the internal configuration reached at the end of the previous run. Each run of an ITM starts by reading one incoming message—i.e., reading a string on a special tape—and ends by sending one outgoing message—i.e., writing a string on another special tape. (In an initial run we assume that the incoming message is the input, and that the internal configuration consists of a blank work tape and a distinguished start state.) An ITM halts when, in a given run, it enters a special halting state, from which it takes no further action. ITMs are meant to be executed in pairs. If A and B are ITMs, an execution of (A, B) on input x is obtained by having x be both A's and B's input and by running alternatively A and B, so that each outgoing message of A is B's incoming message in the next run of B, and vice versa. The number of rounds in an execution of (A, B) is the number of times in which either of the two ITMs sends a message. By convention, an execution of (A, B) starts and ends with a run of B. In its last run, B may *accept* by outputting the special symbol YES, or *reject* by outputting the special symbol NO. Except for their exchanged messages, in an execution of (A, B) neither ITM has access to the internal computation (in particular the coin tosses) of the other. The probability that, after a random execution of (A, B) on a given input x, B accepts is taken over all coin tosses of A and B.

[22] Though concurring with us that properly capturing efficient verification may require more than $\mathcal{P} \neq \mathcal{NP}$, one might also believe that if $\mathcal{P} = \mathcal{NP}$, there would be little or no notion of efficient verification to be captured.

[23] Again, he who is concerned about truth but not about time does not need proofs and provers: he may be equally happy to run a decision algorithm whenever he wishes to establish whether a given statement holds. Proofs cannot properly exist as a separate notion unless they succeed in making verification of truth much easier than accepting truth.

Slightly more generally, we shall also consider ITM pairs (A, B), where A actually is an *interactive circuit*. This allows us to use the size of A to bound more effectively the number of "steps" A may make in an execution with B.[24] Recall that a *circuit of size* $\leq s$ is a finite function computable by at most s Boolean gates, where each gate is either a NOT-gate (with one binary input and one binary output) or an AND-gate (with two binary inputs and one binary output).

Notice that an interactive circuit A may be taken to be deterministic, because it might have wired in any finite lucky sequence of coin tosses. (In this case the probability that B is convinced in a random execution with A on input x solely depends on B's coin tosses.)

DEFINITION 4.1. *Let (P, V) be a pair of ITMs, the second of which running in polynomial time, and let $\mathcal{L}$ be the CS language. We say that (P, V) is an* interactive CS proof system *if there are four positive constants a, b, c, and d such that the following two properties are satisfied.*

$1''$. Feasible completeness. *For all $q = (M, x, y, t) \in \mathcal{L}$, and for all integers k, in every execution of (P, V) on inputs q and 1^k,*
($1''$.i) *P halts within $(|q|kt)^a$ computational steps, and*
($1''$.ii) *V outputs YES.*

$2''$. Computational soundness. *For all $\widetilde{q} \notin \mathcal{L}$, for all k such that $2^k > |q|^b$, and for all (cheating) interactive circuit $\widetilde{P}$ of size $\leq 2^{ck}$, in a random execution of $\widetilde{P}$ with V on inputs $\widetilde{q}$ and 1^k,*

$$Prob[V \text{ outputs } YES] < 2^{-dk}.$$

The constructability of interactive CS proof systems. Let us make the mentioned notion of a collision-free hash function a bit more formal.

DEFINITION 4.2. *Let KG (for key generator) be a probabilistic polynomial-time algorithm, $KG : 1^* \to \Sigma^*$, and let E (for evaluator) be a polynomial-time algorithm, $E : \Sigma^* \times 1^* \to \Sigma^*$ (more precisely, $E : \Sigma^* \times 1^k \to \Sigma^k$ for all positive integers k). We say that the pair (KG, E) is a* collision-free hash function *if there exist positive constants r, s, and t such that for all $k > r$ and for all (collision-finding) circuits CF of size $< 2^{sk}$, letting h be a random output of KG on input k,*

$$Prob_h[CF(h, 1^k) = (x, y) \text{ such that } x \neq y \ \wedge \ E(h, x) = E(h, y)] < 2^{-tk}.$$

From the proof of Theorem 3.9 (indeed, even from the informal arguments of subsection 3.4) the reader can easily derive a proof of the following corollary.

COROLLARY 4.3. *Interactive CS proof systems exist if collision-free hash functions exist.*

(Notice that such CS proof systems would actually be four-round ones: in essence, the verifier runs KG to generate a random h and sends it to the prover as the first message; the prover uses h to construct the Merkle tree and sends its root to the

[24] An ITM making, say, at most s steps, may de facto make "many more steps" if it has a large description—e.g., by encoding a large finite function in its state control. For instance, factoring a randomly chosen k-bit integer appears to be computationally intractable when k is large, but not for an algorithm whose description is about 2^k-bit long! Indeed, the finite state control of such a Turing machine could easily encode the factorization of all k-bit integers. For this reason, in a cryptographic CS proof system, a cheating prover is envisaged to be an algorithm whose running time and description, in some standard encoding, are both bounded. Having a cheating prover be a Boolean combinatorial circuit with a bounded number of gates is actually just a specific but simple way to accomplish this.

verifier as the second message; the verifier runs the sampling algorithm SV to compute a sequence of bit-positions and sends them to the prover as the third message; and the prover returns to the verifier the bit-values of those positions—together with their authenticating paths—as the fourth message.)

Two-round CS proof systems. Informally, a *two-round CS proof system* is an interactive CS proof system (P, V) in every execution of which only two messages are sent: the first by V and the second by P. Such proof systems appear significantly more difficult to implement than general interactive ones. Nonetheless they could be constructed based on Cachin, Micali, and Staedler's new computationally private information-retrieval system [14]. Their construction relies on the difficulty of deciding, given a prime p and an integer n (whose factorization is unknown), whether p divides $\phi(n)$.

4.2. CS proof-systems sharing a random string.

The notion of a CS proof system sharing a random string. In a CS proof system sharing a random string, prover and verifier are ordinary (as opposed to oracle-calling) algorithms, sharing a short random string r. That is, whenever the security parameter is k, they share a string r that both believe to have been randomly selected among those having length k^c, where c is a positive constant. If string r is universally known, it can be shared by all provers and verifiers. (CS proof systems sharing a random string are a special case of one-round CS proof systems because r could be the message sent by the verifier to the prover.)

DEFINITION 4.4. *Let (P, V) be a pair of Turing machines, the second of which runs in polynomial time. We say that (P, V) is a* CS proof system sharing a random string *if there exists a sequence of five positive constants, $c_2, \ldots, c_6$ (referred to as the* fundamental constants *of the system*[25]*), such that the following two properties are satisfied.*

$1'''$. *Feasible completeness.* For all $q = (M, x, y, t) \in \mathcal{L}$ and for all binary string r,
($1'''.i$) on inputs q and r, P halts within $(|q| \cdot |r| \cdot t)^{c_2}$ computational steps outputing a binary string $\mathcal{C}$, whose length is $\leq (|q| \cdot |r|)^{c_3}$, such that
($1'''.ii$) $V(q, r, \mathcal{C}) = YES$.

$2'''$. *Computational soundness.* For all $\widetilde{q} \notin \mathcal{L}$, for all k such that $2^k > |q|^{c_4}$, and for all (cheating) circuits $\widetilde{P}$ whose size is $\leq 2^{c_5 k}$, for a random k^{c_1}-bit string r

$$Prob_r[\widetilde{P}(\widetilde{q}, r) = \widetilde{\mathcal{C}} \wedge V(\widetilde{q}, r, \widetilde{\mathcal{C}}) = YES] \leq 2^{-c_6 k}.$$

We refer to the above strings r and $\mathcal{C}$ as, respectively, a *reference string* and a *CS certificate (of $q \in \mathcal{L}$, relative to r and (P, V)).*

The constructability of CS proof systems sharing a random string. We conjecture that CS proof systems with a random string exist. In particular, their existence is guaranteed by an ad hoc (and stronger) assumption: informally the "replaceability," in Theorem 3.6's $(\mathcal{P}, \mathcal{V})$, of random oracles with (hopefully) adequate functions (e.g., collision-free hashing ones). Such replacements have been advocated,

[25] The "numbering" of these constants has been chosen to facilitate comparison with CS proof systems with a random oracle.

in more general contexts, by Bellare and Rogaway [7].[26]

5. Computationally sound checkers. CS proofs have important implications for validating one-sided heuristics for $\mathcal{NP}$. Generalizing a prior notion of Blum's, we put forward the notion of a *CS checker* and show that CS proofs with a random string imply CS checkers for $\mathcal{NP}$.

To begin with, we state the general problem of heuristic validation we want to solve, explain why prior notions of checkers may be inadequate for solving it, discuss the novel properties we want from a checker, and convey in a simple but *wishful* manner what our checkers are. Then we shall approximate this wishful version of a CS checker by a more formal definition and construction.

Warning. Approaching meaningfully the problem of validating efficient heuristics for $\mathcal{NP}$-complete problems is nontrivial (as we shall see, it entails reconciling "two opposites"), and our particular approach to it may prove quite subjective (if not outright controversial). Nonetheless, the problem at hand is so crucial that we might be excused for putting forward some quite preliminary contributions and ideas.

5.1. The problem of validating one-sided heuristics for $\mathcal{NP}$.

A general problem. $\mathcal{NP}$-complete languages contain very important and useful problems that we would love to solve. Unfortunately, it is extensively believed that $\mathcal{P} \neq \mathcal{NP}$ and $\mathcal{NP} \neq \mathcal{C}o\text{-}\mathcal{NP}$, and thus that our ability of successfully handling $\mathcal{NP}$-complete problems is severely limited. Indeed, if $\mathcal{P} \neq \mathcal{NP}$, then no efficient (i.e., polynomial-time) algorithm may decide membership in an $\mathcal{NP}$-complete language without making any errors. Moreover, if $\mathcal{NP} \neq \mathcal{C}o\text{-}\mathcal{NP}$, then no efficient algorithm may, in general, prove nonmembership in an $\mathcal{NP}$-complete language by means of "short and easy-to-verify" strings.

In light of the above belief, the "best natural alternative" to deciding $\mathcal{NP}$-complete languages efficiently and conveying efficiently to others the results of our determinations consists of tackling $\mathcal{NP}$-complete languages by means of efficient *heuristics* that are *one-sided*. Here by "heuristic" we mean a program (emphasizing that no claim is made about its correctness) and by "one-sided" we mean that such a program, on input a string x, outputs either (1) a proper $\mathcal{NP}$-witness, thereby *proving* that x is in the language, or (2) the symbol NO, thereby *claiming* (without proof) that x is not in the language.

But for an efficient one-sided heuristic to be really useful for tackling $\mathcal{NP}$-complete problems we should *know when it is right*. Of course, when such an heuristic outputs an $\mathcal{NP}$-witness, we can be confident of its correctness on the given input. However, when it outputs NO, skepticism is mandatory: even if the heuristic came with an a priori guarantee of returning the correct answer on most inputs, we might not know whether the input at hand is among those. Thus, in light of the importance of $\mathcal{NP}$-

[26] A word of caution is now due about such replacements. For certain tasks, it is now known how to replace random oracles successfully with ordinary algorithms based on traditional assumptions. For instance, a random oracle provably is "collision-free," but collision-free hash functions can be built, say, under the assumption that the integer factorization or the discrete logarithm problems are computationally intractable. On the other hand, "random-oracle replacement" does not always work: Canetti, Goldreich, and Halevi [15] show that it is possible to construct special algorithms that behave very differently when given access to a random oracle than they do when given access to *any* pseudorandom function. (In light of their result, it should be possible to construct, somewhat artificially, some CS proof systems sharing a random oracle that can never be transformed into CS proof systems sharing a random string by replacing their oracle with a pseudorandom function. But this does not imply that the same holds for every CS proof system with a random oracle, in particular for the $(\mathcal{P}, \mathcal{V})$ of Theorem 3.9).

complete languages and in light of the many efficient one-sided heuristics suggested for these languages, a fundamental problem naturally arises.

> *Given an efficient one-sided heuristic H for an $\mathcal{NP}$-complete language, is there a meaningful and efficient way of using H so as to validate some of its NO outputs?*

Interpreting the problem. The solvability of the above general problem critically depends on its specific interpretation.

One such interpretation was proposed by Manuel Blum when, a few years ago, he introduced the beautiful notion of a checker [11][27], and asked whether $\mathcal{NP}$-complete languages are checkable. Under this interpretation, the general problem still is totally open. Moreover, as we shall argue below, unless this interpretation is suitably broadened, even a positive solution might have a limited usefulness.

In this paper we thus propose a new interpretation of the general problem and, assuming the existence of CS proofs with a random string, provide its first (and positive) solution.

5.2. Blum checkers and their limitations.

The notion of a Blum checker. Intuitively, a Blum checker for a given function f is an algorithm that either (a) determines with arbitrarily high probability that a given program, run on a given input, correctly returns the value of f at that input, or (b) determines that the program does not compute f correctly (possibly, at some other input). Let us quickly recall Blum's definition.

DEFINITION 5.1. *Let f be a function and C a probabilistic oracle-calling algorithm running in expected polynomial time. Then, we say that C is a Blum checker for f if, on input an element x in f's domain and oracle access to any program P (allegedly computing f), the following two properties hold.*

1. *If $P(y) = f(y)$ for all y (i.e., if P correctly computes f for every input), then $C^P(x)$ outputs YES with probability* 1.
2. *If $P(x) \neq f(x)$ (i.e., if P does not compute f correctly on the given input x), then $C^P(x)$ outputs YES with probability $\leq 1/2$.*

The probabilities above are taken solely over the coin tosses of C whenever P is deterministic, and over the coin tosses of both algorithms otherwise.

The above notion of a Blum checker slightly differs from the original one.[28] In particular, according to our reformulation any correct program for computing f immediately yields a checker for f, though not necessarily a useful one (because such a checker may be too slow, or because its correctness may be too hard to establish).[29]

Despite their stringent requirements, Blum checkers have been constructed for a variety of specific functions (see, in particular, the works of Blum, Luby, and Rubinfeld [12] and Lipton [25]).

Note that the notion of a checker is immediately extended to languages: an algorithm C is a Blum checker for a language L if it is a Blum checker for L's characteristic function. Indeed, the interactive proof systems of [26] and [33] yield Blum checkers

[27] We shall call his notion a *Blum checker* to highlight its difference from ours.

[28] Disregarding minor issues, Blum's original formulation imposes an additional condition, roughly, that C run asymptotically faster than the fastest known algorithm for computing f—or asymptotically faster than P when checking P. This additional constraint aims at rebuffing a natural objection: who checks the checker? The condition is in fact an attempt to guarantee, in practical terms, that C is sufficiently different from (and thus "independent" of) P, so that the probability that both C and P make an error in a given execution is smaller than the probability that just P makes an error.

[29] Thus, running a checker C (as defined by us) with a program P may be useful only if C is much faster than P, or if C's correctness is much easier to prove—or believe—than that of P.

for, respectively, any $\#\mathcal{P}$- or $\mathcal{PSPACE}$-complete language.[30]

Blum checkers vs. efficient heuristics for $\mathcal{NP}$-complete problems. We believe that the question of whether Blum checkers for $\mathcal{NP}$-complete languages exist should be interpreted more broadly than originally intended. We in fact argue that, even if they existed, Blum checkers for $\mathcal{NP}$-complete languages might be less relevant than desirable.

DEFINITION 5.2 (informal). *We say that a Blum checker A for a function f is* irrelevant *if, for all efficient heuristics H for f, and for all x in f's domain, with high probability $A^H(x) = NO$ without ever calling H on input x.*

Note that, if $\mathcal{P} \neq \mathcal{NP}$, then no efficient heuristic for an $\mathcal{NP}$-complete language is correct on all inputs. Thus it is quite legitimate for a Blum checker for an $\mathcal{NP}$-complete language to output NO whenever its oracle is an efficient heuristic, without ever calling it on the specific input at hand: a NO-output simply indicates that the efficient heuristic is incorrect on some inputs (possibly different from the one at hand). However, constructing an irrelevant Blum checker for the characteristic function of SAT (the language consisting of all Boolean formulae in conjunctive normal form) under the assumption that $\mathcal{P} \neq \mathcal{NP}$ is not trivial. The difficulty lies in the fact that a checker does not know whether it is accessing a polynomial-time program (in which case, if $\mathcal{P} \neq \mathcal{NP}$, it could always output NO), or an exponential-time program that is correct on all inputs (in which case it should always output YES). We can, however, construct such an irrelevant Blum checker under the assumption that one-way functions exist. This assumption appears to be stronger than $\mathcal{P} \neq \mathcal{NP}$ but is widely believed and provides the basis of all modern cryptography.

DEFINITION 5.3 (informal). *We say that a function f mapping binary strings to binary strings is* one-way *if it is length-preserving, polynomial-time computable, but not polynomial-time invertible in the following sense: for any polynomial-time algorithm A, if one generates at random a sufficiently long input z and computes $y = f(z)$, then the probability that $A(y)$ is a counter-image of f is miniscule.*

THEOREM 5.4 (informal). *If one-way functions and Blum checkers for $\mathcal{NP}$-complete languages exist, then there exist irrelevant Blum checkers for $\mathcal{NP}$-complete languages.*

Proof (informal). Let SAT be the $\mathcal{NP}$-complete language of all satisfiable formulae in conjunctive normal form, let P be a program allegedly deciding SAT, let C be a Blum checker for SAT, let f be a one-way function, and let $\mathcal{C}$ be the following oracle-calling algorithm.

On input an n-variable formula F in conjunctive normal form, and oracle access to P, $\mathcal{C}$ works in two stages. In the first stage, $\mathcal{C}$ randomly selects a (sufficiently long) string z and computes (in polynomial time) $y = f(z)$. After that, $\mathcal{C}$ utilizes the completeness of SAT to construct, and feed to P, n formulae in conjunctive normal form, $F_1, \ldots, F_n$, whose satisfiability "encodes a counter-image of y under f."

(For instance, F_1 is constructed so as to be satisfiable if and only if there exists a counter-image of y whose first bit is 0. The checker feeds such an F_1 to P. If P outputs "F_1 is satisfiable," then $\mathcal{C}$ constructs F_2 to be a formula that is satisfiable if and only if there exists a counter-image of y whose 2-bit prefix is 00. If, instead, P responds

[30] In fact, the definition of a Blum checker for a language L is analogous to a restricted kind of interactive proof for L: one whose prover is a probabilistic polynomial-time algorithm with access to an oracle for membership in L. Indeed, whenever a language L possesses such a kind of interactive proof system, a checker C for L is constructed as follows. On inputs P (a program allegedly deciding membership in L) and x, the checker C simply runs both prover and verifier on input x, giving the prover oracle access to program P. C outputs YES if the verifier accepts, and rejects otherwise.

"F_1 is not satisfiable," then $\mathcal{C}$ constructs F_2 to be a formula that is satisfiable if and only if there exists a counter-image of y whose 2-bit prefix is 10. And so on, until all formulae $F_1, \ldots, F_n$ are constructed and all outputs $P(F_1), \ldots, P(F_n)$ are obtained.)

Because string y is, by construction, guaranteed to be in the range of f, at the end of this process one either finds (a) a counter-image of y under f, or (b) a proof that P is wrong (because if no f-inverse of y has been found, then P must have provided a wrong answer for at least one of the formulae F_i). If event (b) occurs, $\mathcal{C}$ halts outputting NO. Otherwise, in a second phase, $\mathcal{C}$ runs Blum checker C on input the original formula F and with oracle access to P. When C halts so does $\mathcal{C}$, outputting the same YES/NO value that C does.

Let us now argue that $\mathcal{C}$ is a Blum checker for SAT. First, it is quite clear that $\mathcal{C}$ runs in probabilistic polynomial time. Then there are two cases to consider.

1. *P correctly computes SAT's characteristic function.* In this case, a counter-image of y is found, and thus $\mathcal{C}^P$ does not halt in the first phase. Moreover, in the second phase, $\mathcal{C}$ runs Blum checker C with the same correct program P. Therefore, by property 1 of a Blum checker, C^P will output YES no matter what the original input formula F might be, and, by construction, so will $\mathcal{C}^P$. This shows that $\mathcal{C}$ enjoys property 1 of a Blum checker for SAT.
2. *$P(F)$ provides the wrong answer about the satisfiability of F.* In this case, either $\mathcal{C}^P$ halts in phase 1 outputting NO, or it executes phase 2 by running $C^P(F)$, that is, the original Blum checker for SAT, C, on the same input F and the same oracle P. Therefore, by property 2 of a Blum checker, the probability that $C^P(F)$ will halt outputting YES is no greater than 1/2. By construction, the same holds for $\mathcal{C}^P(F)$. This shows that $\mathcal{C}$ enjoys property 2 of a Blum checker for SAT.

Finally, let us argue that, for any input F and any efficient P (no matter how well it may approximate SAT's characteristic function), almost always $\mathcal{C}^P(F) = NO$, without even calling P on F. In fact, because $\mathcal{C}$ runs in polynomial time, whenever P is polynomial-time, so is algorithm $\mathcal{C}^P$. Therefore, $\mathcal{C}^P$ has essentially no chance of inverting a one-way function evaluated on a random input. Therefore, $\mathcal{C}$ will output NO in phase 1, where it does not call P on F. □

In sum, differently from many other contexts, the notion of a Blum checker may not be too useful for handling efficient heuristics for $\mathcal{NP}$-complete languages, either because no such checkers exist[31] or because they may exist but not be too useful.

Blum checkers are not complexity-preserving. The lesson we derive from the above sketched proof of Theorem 5.4 is that Blum's notion of a checker lacks a new property that we name *complexity preservation*. Intuitively, a Blum checker for satisfiability, when given a "not-so-difficult" formula F, may ignore it altogether and instead call the to-be-tested efficient heuristic on *very special* and *possibly much harder* inputs, thus forcing the heuristic to make a mistake and justifying its own outputting NO (i.e., "the heuristic is wrong").

The possibility of calling a given heuristic H on inputs that are harder than the given one essentially erodes the chances of meaningfully validating H's answer whenever it happens to be correct.[32] Such a possibility may not matter much if "the

[31] Notice that this possibility does not contradict the fact that $\mathcal{NP}$ is contained in both $\#\mathcal{P}$ and $\mathcal{PSPACE}$ and that $\#\mathcal{P}$- and $\mathcal{PSPACE}$-complete languages are Blum checkable!

[32] Note that such possibility not only is present in the *definition* of a Blum checker but also in all known *examples* of a Blum checker. Typically, in fact, a Blum checker works by calling its given heuristic on random inputs, and these may be more difficult than the specific, original one.

difference in computational complexity between any two inputs of similar length" is somewhat bounded. But it may matter a lot if such a difference is enormous—which may be the case for $\mathcal{NP}$-complete languages, as they encode membership in both easy and hard languages. We thus wish to develop a notion of a "complexity-preserving" checker.

5.3. New checkers for new goals.

The old goal. Blum checkers are very useful to catch the occasional mistake of programs believed to be correct on all inputs. That is, they are ideally suited to check *fast programs for easy functions* (or slow program for hard functions). In fact, if f is an efficiently computable function, then we know a priori that there are efficient and correct programs for f. Therefore, if a reputable software company produces a program P for f, it might be reasonable to expect that P is correct. In this framework, by running a Blum checker for f, with oracle P, on a given input x we have nothing to lose[33] and something to gain. Indeed, if the checker answers YES, we have "verified our expectations" about the correctness of P at least on input x (a small knowledge gain), and if the checker answers NO, we have proved our expectations about P to be wrong (a big knowledge gain).

The new goal. We instead want to develop checkers for a related but different goal: validating efficient heuristics that are known to be incorrect on some inputs. That is, we wish to develop checkers suitable for handling *fast programs for hard functions.* Now, if f is a function hard to compute, then we know a priori that no efficient program correctly computes it. Therefore, obtaining from a checker a proof that such an efficient program does not compute f correctly would be quite redundant. We instead want checkers that, at least occasionally, if an efficient heuristic for f happens to be correct on some input x, are capable of convincing us that this is the case.

Interpreting the new goal. Several possible valid interpretations of this general constraint are possible. In this paper we focus on a single one: namely, we want checkers that are *complexity-preserving.* Let f be a function that is hard to compute (at least in the worst case). Then, intuitively, a complexity-preserving checker for f will, on input x, call a candidate program for f only on inputs for which evaluating f is essentially as difficult as for x.

Our point is that, while a given heuristic for satisfiability, $\mathcal{H}$, may make mistakes on some formulae, it may return remarkably accurate answers on some class of formulae (e.g., those decidable in $O(2^{cn})$ time, for some constant $c < 1$, by a given deciding algorithm D). Intuitively, therefore, checkers should be defined (and built!) so that, if the input formula belongs to that class and $\mathcal{H}$ happens to be correct on the input formula, they call $\mathcal{H}$ only on additional formulae in that class.

5.4. The wishful version of a CS checker. The spirit of a CS checker is best conveyed wishfully assuming (for a second) that $\mathcal{NP}$ equaled $Co\text{-}\mathcal{NP}$. In that case, our CS checkers would take the following simple and appealing form.

Wishful checkers. Define a *wishful checker* to be a polynomial-time algorithm C that, on input a Boolean formula F, outputs a Boolean formula $\mathcal{F}$ satisfying the following two properties.

1. *Membership reversion.* $\mathcal{F} \in$ SAT if and only if $F \notin$ SAT.

[33] Blum checkers are often so fast (e.g., running in time sublinear in that of the algorithm they check) that not even this is much of a concern.

2. *Complexity preservation.* "The satisfiability of $\mathcal{F}$ is as hard to decide as that of F."

How to use a wishful checker. We interpret the above algorithm C as a kind of checker because it immediately yields the following algorithm C' (more closely matching our intuition of a checker).

C' : Given an efficient one-sided heuristic for SAT, H, and an input formula, F, compute $\mathcal{F} = C(F)$. Then, call H so as to obtain the two values $H(F)$ and $H(\mathcal{F})$. If either value is different than NO, then (H being one-sided) a satisfying assignment has been computed either proving that $F \in$ SAT or that $F \notin$ SAT. Otherwise, $H(F) = H(\mathcal{F}) = NO$ proves that H is incorrect.

Note that, by the very definition of a wishful checker, the above proof that H is incorrect has been obtained without querying H on formulae harder than the original input F.

5.5. The notion of a CS checker. Let us now informally explain how, without assuming $\mathcal{NP} = Co\text{-}\mathcal{NP}$, CS checkers may approximate wishful ones to a sufficiently close extent. Renouncing to achieving greater generality, we limit our discussion to CS checkers for SAT.

CS checkers. Informally speaking, a *CS checker* is a polynomial-time algorithm $\mathcal{C}$ that, on input a formula F, outputs a Boolean formula $\mathcal{F}$, called the *coinput*, satisfying the following properties.

1. *Membership semireversion.*
   1.1. At least one of F and $\mathcal{F}$ is satisfiable.
   1.2. If F is satisfiable, then no efficient algorithm has a nonnegligible chance of finding a satisfying assignment for $\mathcal{F}$.
2. *Complexity semipreservation.* If $F \notin$ SAT, then the satisfiability of $\mathcal{F}$ is as hard to decide as that of F.
   (Note: We explain why complexity preservation is restricted to the "$F \notin$ SAT" case a few lines below.)

How to use CS checkers. We interpret the above $\mathcal{C}$ as a checker because it immediately yields the following algorithm $\mathcal{C}'$ (that better matches what we may intuitively expect from a checker).

$\mathcal{C}'$: Given an efficient one-sided heuristic for SAT, H, and an input formula, F, do the following.
- Compute the coinput $\mathcal{F} = \mathcal{C}(F)$.
- Call H so as to obtain $H(F)$.
- If $H(F) \neq NO$, HALT.
- If $H(F) = NO$, call H so as to obtain $H(\mathcal{F})$ and HALT.

The usefulness of CS checkers. The usefulness of the above $\mathcal{C}'$ stems from the following two properties.

(a) $\mathcal{C}'$ is informative about the satisfiability of F or the correctness of H.

(b) If H is correct on F, then $\mathcal{C}'$ never calls H on a formula harder than F.

Indeed, a computation of $\mathcal{C}'$ results in (1) showing a satisfying assignment of F, (2) showing a satisfying assignment of $\mathcal{F}$, or (3) showing that $H(F) = H(\mathcal{F}) = NO$.

A type-1 result clearly proves that $F \in$ SAT.

A type-2 result is interpretable as saying that F is unsatisfiable. This is so because if F belonged to SAT, then either a satisfying assignment of coinput $\mathcal{F}$ does not exist, or (by the very definition of a CS checker) the probability that it can be obtained in polynomial time is negligible. (Notice, in fact, that $\mathcal{C}'$ is efficient because both $\mathcal{C}$ and H are.)

A type-3 result proves that H is wrong. In fact, if $H(F) = NO$ is correct, then (by property 1.1 of a CS checker) $\mathcal{F} \notin$ SAT, and thus $H(\mathcal{F}) = NO$ is incorrect. Let us now argue that *if H is correct on F*, our proof of H's incorrectness has been obtained in a complexity-preserving manner. We distinguish two cases.

1. If H is correct on F and $H(F) \neq NO$, then $H(F)$ is an (easy-to-verify) satisfying assignment of F, and thus $\mathcal{C}'$ does not call H on any coinput. Therefore, $\mathcal{C}$ vacuously does not call H on any $\mathcal{F}$ harder than F.
2. If H is correct on F and $H(F) = NO$, then $F \notin$ SAT, in which case (by property 2 of a CS checker) $\mathcal{F}$ is guaranteed to have the same complexity as F.

If instead H is *not* correct about our original input F, then $H(F) = H(\mathcal{F}) = NO$ still is a proof of H's incorrectness, but not necessarily one obtained in a complexity-preserving manner. Notice, however, that lacking complexity preservation in this case is of no concern: *if H happens wrong about our own original input, we are happy to prove that H is wrong in any manner.* Recall that in checking we care about our own original input x more than about H. Thus if $H(x)$ is correct, we aim at "proving" this fact, and we do not want to throw H away by calling it on much harder inputs. But if $H(x)$ is wrong, we do not mind dismissing H in any way. Least of all, we want to be convinced that $H(x)$ is right!

The complexity preservation of a CS checker. To complete our informal discussion of CS checkers we must explain in what sense, whenever $F \notin$ SAT, the complexity of F is close to that of $\mathcal{F}$. That is, we must explain (1) how we measure the complexity of the original input, and (2) how the coinput preserves this complexity.

1. *Complexity meters.* The complexity of the original input F is defined to be the number of steps made by a chosen deciding algorithm for SAT, D, on input F. That is, when a CS checker for SAT is given an input formula F, it is also given as an additional input the description of this chosen D. We refer to D as *the complexity meter*. In fact, by specifying D, we (implicitly) pin down the complexity of the original formula F. By insisting that D be a decider for SAT (i.e., that D be correct) we insist that the complexity of the original input be a "genuine" one.[34]

   By properly choosing the complexity meter, one may be able to force the complexity of the original input to be small (and thus force the checker to query its given heuristic on a coinput of similarly small complexity). Choosing D to be the algorithm that tries all possible satisfying assignments for F is certainly legitimate but not too meaningful. (Because any formula would have "maximum complexity" relative to such a complexity meter, the checker would essentially be free to call its given heuristic on any possible coinput.) Quite differently, if the original input F is known to belong to a class of formulae for which a given SAT algorithm performs very well (e.g., runs in subexponential time), by specifying that algorithm as our complexity meter, we force the checker to call its given heuristic only on a coinput of similarly low complexity.

   Let us stress that we do not require that the checker, or someone choosing a complexity meter D, know how many steps D takes on the original input F. Nor do we require that one distinguish (somehow) for which inputs, if any, algorithm D (slow in the worst case) may be reasonably fast. Rather, we require that, if F happens to belong to those inputs on which D is fast, then

[34] In particular, if D were allowed to make errors, all formulae F could have constant complexity.

it is this lower (and possibly unknown) complexity that should be preserved by a CS checker.

By specifying D one specifies implicitly *the complexity of F, whatever it happens to be.*

For technical reasons, however, we require that D's running time be upper-bounded by 2^{2n}, a bound that essentially poses no real restrictions. Within these bounds, any algorithm for satisfiability could always be "timed-out" and then converted to an exhaustive search for a satisfying assignment).

2. *Complexity cometers.* The complexity of a coinput $\mathcal{F}$ is defined to be the number of steps taken on input $\mathcal{F}$ by a decider for SAT, $\mathcal{D}$, specified before hand. We refer to such a $\mathcal{D}$ as *the complexity cometer.*

   Thus a complexity cometer is independent of the chosen complexity meter: the first is fixed once and for all (in fact, it could be made part of the very definition of a CS checker), while the second is chosen afresh each time a CS checker is run. Under these circumstances, at first glance, it may appear surprising that a CS checker may succeed in keeping the complexity of the coinput close to that of the original input. But the fixed cometer $\mathcal{D}$ includes the code of the universal algorithm, so that, in a sense, the complexity of a coinput is measured relative to a "decider for SAT that is easily constructed on input D."

   Notice that one could conceive stating complexity preservation by simply saying that the number of steps taken by a chosen D on the original input is polynomially close to the number of steps taken by the same D on the coinput. This would be a simpler way of having the cometer easily depend on the meter. However, we needed to endow CS checkers with a bit more room to maneuver than that. In any case, we believe it preferable to have the meter that is a fixed component of the CS checker to be a *universal meter.*

5.6. The actual definition of a CS checker.

Preliminaries.

- We let CNF denote the language of all formulae in conjunctive normal form, and SAT the set of all satisfiable formulae in CNF. If $F \in \mathrm{SAT}$, then we denote by $\mathrm{SAT}(F)$ the set of all satisfiable assignments of F. For any positive integer n, CNF_n and SAT_n will denote, respectively, all formulae in CNF and SAT whose binary length is n.
- By an SAT *decider* we mean an algorithm that (correctly) decides the language SAT. (Deciders need not output a satisfying assignment in case the input formula is satisfiable.) We say that an SAT decider D is *reasonable* if, for all $F \in \mathrm{CNF}$, $\#D(F) \leq 2^{|F|}$.
- If A is a probabilistic algorithm and E an event (involving executions of A on specified inputs), by $Prob_A[E]$ we denote the probability of E, taken over all possible coin tosses of A.

DEFINITION 5.5. *Let Φ be a probabilistic polynomial-time algorithm, $\mathcal{D}$ an SAT decider, and $\mathcal{Q}(.,.,.,.)$ a positive polynomial. We say that $(\Phi, \mathcal{D}, \mathcal{Q})$ is a* CS checker *if, for all positive constant c, for all CNF formulae F, for all reasonable SAT deciders D, and for all sufficiently large k, on input $(F, D, 1^k)$ Φ outputs a formula $\mathcal{F}$ such that:*

1. $F \vee \mathcal{F} \in \mathrm{SAT}$;
2. $F \in \mathrm{SAT} \Rightarrow$ *for all k^c-size circuits A, $Prob_\Phi[A(\mathcal{F}) \in \mathrm{SAT}(\mathcal{F})] < 2^{-k}$; and*
3. $F \notin \mathrm{SAT} \Rightarrow \#\mathcal{D}(\mathcal{F}) < \mathcal{Q}(|F|, |D|, 1^k, \#D(F))$.

If $\mathcal{C} = (\Phi, \mathcal{D}, \mathcal{Q})$ is a CS checker, we refer to Φ as the *reducer*, to $\mathcal{D}$ as the *complexity cometer*, and to $\mathcal{Q}$ as the *complexity slackness*.

Remark. Note that even the existence of triplets $(\Phi, \mathcal{D}, \mathcal{Q})$ satisfying just properties 1 and 2 alone constitutes a surprising statement about SAT. Informally, they say that any formula F can be efficiently transformed to a formula $\mathcal{F}$ such that (1) at least one of them is satisfiable, while (2) every efficient algorithm can find a satisfying assignments for at most one them. That is,

properties 1 *and* 2 *"almost" say that* $\mathcal{NP} = Co\text{-}\mathcal{NP}$,

in the sense that, to convince a verifier V that a formula F is not satisfiable, a prover P may first run Φ on input $(F, D, 1^k)$ so as to compute $\mathcal{F}$, and then (because if $F \notin$ SAT, then, by property 1, $\mathcal{F}$ is satisfiable) produce a satisfying assignment for $\mathcal{F}$. The verifier is convinced because, if also F were satisfiable, then a satisfying assignment for F could be found in less than 2^k steps, which would violate property 2 whenever P is poly(k) size, and k is large enough!

5.7. Implementing CS checkers for SAT. Let us recall some known properties of Cook's [16] and Levin's [24] $\mathcal{NP}$-completeness constructions.

Key properties of Cook's and Levin's constructions. Given a polynomial-time predicate $A(\cdot, \cdot)$ and a positive constant b, these constructions consist of a polynomial-time algorithm that, on input a binary string x, outputs a CNF formula ϕ that is satisfiable if and only if there is a binary string σ such that $|\sigma| \leq |x|^b$ and $A(x, \sigma) = YES$. We refer to such a string σ as a *witness (for x)*. The construction further enjoys the following extra properties (which are actually required by Levin's definition of $\mathcal{NP}$-completeness):

(i) x is polynomial-time retrievable from ϕ;
(ii) a proper witness for x is polynomial-time computable from any satisfying assignment for ϕ (if one exists); and
(iii) a satisfying assignment for ϕ is polynomial-time computable from any proper witness for x (if one exists).

THEOREM 5.6. *If CS proof systems sharing a random string exist, then CS checkers for SAT exist.*

Proof. Let (P, V) be a CS proof system sharing a random string with fundamental constants $c_2, \ldots, c_6$, and consider the following algorithm.

ALGORITHM Φ

Inputs: F, a CNF formula, D, a reasonable SAT solver, and 1^k, a security parameter.
Subroutines: P and V.
Output: a CNF formula $\mathcal{F}$.

Code: Randomly select a k-bit (reference) string r for (P, V), and use Cook's (or Levin's) construction to compute a CNF formula $\mathcal{F}$ that is satisfiable if and only if there exist two binary strings t and σ such that, setting $q = (F, D, NO, t)$, the following three properties hold: (1) $|t| \leq 2|F|$,[35] (2) $|\sigma| \leq (|q| \cdot k)^{c_3}$, and (3) $V(q, r, \sigma) = YES$.
{*Comment:* If it exists, σ is a CS certificate of $(D, F, NO, t) \in \mathcal{L}$, relative to (P, V) and reference string r. The existence of such a σ, however, does not guarantee that $D(F) = NO$.[36]}

[35] i.e., because D is reasonable, considering t as an integer, $t = \#D(F) \leq 2^{|F|}$.

[36] In fact, we "expect" that σ exists (and thus that $\mathcal{F}$ is satisfiable) with "overwhelming probability," even when F is satisfiable. But σ is hard to find.

Let us now show that there exist an SAT decider $\mathcal{D}$ and a positive polynomial $\mathcal{Q}$ such that $\mathcal{C} = (\Phi, \mathcal{D}, \mathcal{Q})$ is a CS checker.

To begin with, notice that, because of the polynomiality of V and of Cook's construction, Φ is polynomial-time.[37]

Further, because properties 1 and 2 of a CS checker (as per Definition 5.5) only depend on its reducer, let us show that they hold for our Φ prior to defining $\mathcal{D}$ and $\mathcal{Q}$.

Property 1 holds trivially if $F \in$ SAT. Assume therefore that $F \notin$ SAT. Then, because of the correctness and running time of the complexity meter D, we have $D(F) = NO$ within $t \leq 2^{2n}$ steps. Thus, by the (feasible) completeness of (P, V) for any possible reference string r there is a CS certificate σ of $q = (D, F, NO, t) \in \mathcal{L}$. Thus, $\mathcal{F} \in$ SAT, proving that property 1 holds in all cases.

Property 2 is established by contradiction. Assume that there exists an input formula $F \in$ SAT and a poly(k)-size circuit A that, with nonnegligible probability, computes a satisfying assignment of a so-constructed coinput $\mathcal{F}$. Then, by property (ii) of Cook's construction, from such a satisfying assignment (if it exists and is found) one computes in polynomial time both t and a CS certificate σ of $q = (D, F, NO, t) \in \mathcal{L}$. But if $F \in$ SAT, then for no t is $q = (D, F, NO, t) \in \mathcal{L}$. Therefore, this contradicts the computational soundness of (P, V).

Let us finally show that there exist an SAT decider $\mathcal{D}$ and a positive polynomial $\mathcal{Q}$ such that, for all formulae $F \notin CNF$, for all complexity meters D, and for all security parameters k, if D, on input F, takes t ($\leq 2^{2|F|}$) steps to decide that no satisfying assignment for F exists, then, given any coinput $\mathcal{F}$ of F, $\mathcal{D}$ finds a satisfying assignment for $\mathcal{F}$ in at most $Q(|F|, |D|, k, t)$ steps.

Algorithm $\mathcal{D}$ works in four phases as follows.

$\mathcal{D}1$. Computes F, D, and r from $\mathcal{F}$.
(Due to property (i) of Cook's construction, $\mathcal{D}$ can execute this phase in time polynomial in $|\mathcal{F}|$. Thus, because $\mathcal{F}$ has been computed by $\mathcal{C}$ in time polynomial in $|F|$, $|D|$, and k, this phase is executable in time polynomial in $|F|$, $|D|$, and k.)

$\mathcal{D}2$. Runs D on input F to find the exact number of steps, t, taken by D to output NO on input F.
(Because D can be simulated with a slow-down polynomial in $|D|$, this phase takes time polynomial in $|D|$ and t.)

$\mathcal{D}3$. Run prover P on input $q = (D, F, NO, t)$ and reference string r to produce a CS certificate, σ, of $q \in \mathcal{L}$.
(Due to the feasible completeness of (P, V), this phase is executable in time polynomial in $|q|$, k, and t; and thus in time polynomial in $|F|$, $|D|$, k, and t.)

$\mathcal{D}4$. Use σ to compute a satisfying assignment for $\mathcal{F}$.
(Due to property (iii) of Cook's construction, this phase also can be implemented in time polynomial in $|F|$, $|D|$, and k.)

Because each phase is implementable in time polynomial in $|F|$, $|D|$, k, and t, there exists a polynomial $\mathcal{Q}$ such that $\mathcal{D}(\mathcal{F})$ outputs a satisfying assignment of $\mathcal{F}$ in $Q(|F|, |D|, k, t)$ steps.

---

[37] Indeed, define $A(\cdot, \cdot)$ as follows: $A((F, D, r), (t, \sigma)) \stackrel{\text{def}}{=} V((D, F, NO, t), r, \sigma)$. Notice now that A is polynomial-time: in fact, V is the verifier of a CS proof system with a random string. Notice also that $|\sigma|$ is polynomially bounded in $|F|$, $|D|$, and $|r|$: in fact $q = (D, F, NO, t)$, $|t| \leq 2|F|$, and $|\sigma| \leq (|q|k)^{c_3}$.

Finally, notice that the above four-phase procedure can be converted to an SAT decider by interleaving two different computations. In the first, an exhaustive search is conducted for deciding whether $\mathcal{F}$ is satisfiable. In the second, $\mathcal{F}$ is interpreted as a coinput of F, and the above four-phase procedure is run. The so-modified $\mathcal{D}$ halts when either computation halts and outputs what the halting computation does. □

5.8. Remarks.

An alternative formulation. As we said, any CS checker $\mathcal{C}$ (as per Definition 5.5) immediately yields an oracle-calling algorithm that, on input a formula F (a complexity meter D, and a security parameter k) and access to a one-sided efficient heuristic H, computes a coinput $\mathcal{F}$ and obtains $H(F)$ and $H(\mathcal{F})$.

With this in mind, we can rephrase Theorem 5.6 as follows (and obtain—implicitly—a definition of a CS checker that is more closely tailored to our implemetation).

COROLLARY 5.7. *If CS proof systems sharing a random string exist, then there exist* (1) *a polynomial-time oracle-calling algorithm* $\mathcal{C}^{(\cdot)}(\cdot,\cdot,\cdot)$ *that, whenever its first input is a CNF formula* F*, queries its oracle at most twice: once about* F*, and possibly a second time about a second CNF formula* $\mathcal{F}$*;* (2) *an SAT decider* $\mathcal{D}$ *and a polynomial* $Q(.,.,.,.)$ *such that*

for all one-sided heuristics H *for SAT, for all* $F \in CNF$*, for all reasonable SAT deciders* D *solving* F *in* $\leq 2^{2|F|}$ *steps, for all sufficiently long random binary strings* r*, the following two properties hold.*

rm 1. Individual-complexity preservation. *If* H *is correct on* F *and* $\mathcal{C}^H(F,D,r)$ *queries* H *about a second CNF formula* $\mathcal{F}$*, then*

$$\#\mathcal{D}(\mathcal{F}) \leq Q(|F|,|D|,|r|,\#D(F)).$$

2. Computational meaningfulness. $\mathcal{C}^H(F,D,r)$ *produces one of the following three outputs:*
   (a) *a satisfying assignment for* F *(i.e., a proof that* F *is satisfiable),*
   (b) *a CS proof, relative to* (P,V) *and reference string* r*, of* $D(F) = NO$ *(i.e., evidence that* F *is not satisfiable),*
   (c) *a formula* $\mathcal{F}$ *such that, by construction, either* F *or* $\mathcal{F}$ *is satisfiable, and yet* $H(F) = H(\mathcal{F}) = NO$ *(i.e., a proof that* H *is not correct).*

Unlike Blum checkers, the above oracle-calling algorithm $\mathcal{C}$ does not provide answers that are correct with arbitrarily high probability (computed over its possible coin tosses). The type-(a) and type-(c) outputs of $\mathcal{C}$ are errorless, at least in the sense that any error here can be efficiently detected. But a type-(b) output of $\mathcal{C}$, interpreted as a (computationally) meaningful explanation that F is nonsatisfiable, may be wrong in a noneasily detectable manner: if F is satisfiable, $\mathcal{C}$ could output a "false" CS proof of $D(F) = NO$ with positive probability. However, this probability is reasonably high only if an enormous amount of computation is performed; whereas, in our application, all computation is performed by $\mathcal{C}$ which is polynomial-time and by oracle H which is also polynomial-time. Therefore, the probability of a false type-(b) output is absolutely negligible.

Another advantage of one-sided heuristics. Our CS checkers only deal with one-sided heuristics for SAT. As already discussed, given the one-sided nature of $\mathcal{NP}$, this is a natural choice. On the other hand, could we have dealt with heuristics just outputting YES (i.e., "satisfiable, but with no proof") or NO?

So far, because of the self-reducibility property of $\mathcal{NP}$-complete problems, choos-

ing between either type of heuristics has often been a matter of individual taste. Indeed, it is well known that a decision oracle for SAT can, in polynomial time, be converted to a search oracle for SAT. As we explain below, however, this "equivalence" between decision and search relative to $\mathcal{NP}$-complete languages may cease to hold when one demands, as we do, that our reductions preserve the complexity of individual inputs, rather than just that of complexity classes.

When dealing with one-sided efficient heuristics H for SAT, assuming that H is correct on F, we need only to take care of complexity preservation when $H(F) \neq NO$. In fact, if $H(F) \in SAT(F)$, then there is no need to call H on any coinput $\mathcal{F}$, and thus there is no complexity to be preserved. Presumably, however, if H outputs just YES and NO, we would care about preserving F's complexity also when $H(F)$ is correct and $H(F) = YES$. Now, to convince ourselves that $H(F) = YES$ is correct, we could run the self-reducibility algorithm, calling H on a sequence of formulae $F_1, \ldots, F_n$ (obtained by "fixing" a new variable each time), so as to find a satisfying assignment of F, or prove that H is wrong (on either F or some F_i). The problem is, however, that this self-reducibility process may not be complexity-preserving: It may be the case that our F is relatively easy, while some of the F_i's are very hard. Indeed, it is conceivable that it is the "degree of freedom" of the variables of the original formula F that make it easy to decide (without finding any $\mathcal{NP}$-proof of it) that F is satisfiable. However, after sufficiently many variables of F have been fixed, the difficulty of deciding satisfiability may grow dramatically high (though later on, when sufficiently many variables have been fixed, it will dramatically drop).

Extra complexity preservation. Notice that, in the implementation of the proof of Theorem 5.6, the CS checker preserves the complexity of the original input F in a much closer manner than demanded by our definition. Indeed, the coinput $\mathcal{F}$ consists of the very encoding of the computation of the complexity meter D on input F (and we wonder whether this may yield a preferable formulation of complexity preservation).

Additional applications. We believe that complexity preservation, in different formulations, will be useful to other contexts as well. In particular, it will enhance the meaningfulness of many reductions in a complexity setting. For instance, using complexity preservation, [22] presents a more refined notion of a proof of knowledge [21, 36, 18, 6].

An open problem. Is it possible to (define and) construct CS checkers that, when given an heuristic H and an input x, also receive a concise algorithmic representation of a ("nontrivial") set S and call H only on x and elements of S? Such checkers could still be allowed to output a proof that the given heuristic H is wrong. But, if H happens to be correct on S, and the given input happens to belong to S, they should output a "validation" for $H(x)$ (rather than a proof that H is wrong).

6. Certified computation. In this section we reinterpret the results of sections 3 and 4 in terms of *computation* rather than proofs. More precisely, we aim at obtaining certificates ensuring that no error has occurred in a *given* execution of a *given* algorithm on a *given* input.

That is, certified computation does not deal with semantic questions such as "is algorithm A correct?" Rather it addresses the following syntactic question.

Is string y what algorithm A should output on input x (no matter what A is supposed to do)?

This question is quite crucial whenever we are confident in the design of a given algorithm A but less so in the physical computer that runs it. For instance, the computer

hardware may be defective. Alternatively, the hardware may function properly, but the operating system may be flawed. Alternatively yet, the hardware and software may be fine, but some α rays may succeed in flipping a bit together with its controls, so that the original bit value is not restored.

Moreover, even assuming that the physical computer is correct, after running A on input x so as to obtain a result y, can we convince someone else that y is indeed equal to $A(x)$ without having him redo the computation himself?

Certified computation provides a special way to answer these basic questions for any algorithm and for any input.

Defining certified computation. In view of our work so far, formalizing and exemplifying (at least given a random oracle) the notion of certified computation is rather straightforward but tedious indeed. We thus think that is best to proceed at a very intuitive level.

DEFINITION 6.1 (informal). *A* certified-computation system *is a compiler-verifier pair of efficient algorithms, $(\mathcal{C}, \mathcal{V})$. Given any algorithm A as input, $\mathcal{C}$ outputs an equivalent algorithm A' enjoying the following properties.*

1. *A' runs in essentially the same time as A does.*
2. *A' receives the same inputs applicable to A and produces the same outputs.*
3. *For each input x, algorithm A' produces the same output y as A, but also a short and easily inspectable string C vouching that indeed $y = A(x)$ in the following sense.*

   If $A(x)$ outputs y in t steps, then $\mathcal{V}(A, x, y, t, C) = YES$. Otherwise, it is very hard to find a string σ such that $\mathcal{V}(A, x, y, t, \sigma) = YES$.

Of course, one may ask who verifies the correctness of the verifier (i.e., either of algorithm $\mathcal{V}$ itself or of its executions). Note, however, that such a $\mathcal{V}$ is a *unique* program, capable of verifying certificates for the correct execution of *all* other programs. It is thus meaningful to invest sufficient time in proving the correctness of this particular algorithm (e.g., by verification methods). Also, being that $\mathcal{V}$ is quite efficient (and runs on short inputs) we may afford to execute it on very "conservative" hardware (i.e., with particular redundancy, resiliency, and so on), or even on a multiplicity of hardwares.

Constructing certified-computation systems. One possible way of constructing program certification systems essentially consists of giving a CS proof of the statement "$y = A(x)$ in t steps." Let us explain. On input x, algorithm $A' = \mathcal{C}(A)$ first runs A on x so that, after some number t of steps, an output y is obtained. Then A' outputs y and a CS proof of $(A, x, y, t) \in \mathcal{L}$. Thus the length of such a CS proof will be polynomial in $\log t$ (and, of course, $|A|$, $|x|$, $|y|$, and some suitable security parameter k). The additional time required to produce this CS proof is comparable to the running time of A. (This holds if one uses the construction of Theorem 3.8 rather than a generic CS proof system.) Indeed, while the definition of CS proofs allows a polynomial relation between these running times, the sketched construction actually yields a linear-time relation!

An alternative notion of certified computation. Another type of certified computation was previously proposed in [3] as an application of $\mathcal{PCP}$. In essence, in their notion, $A'(x)$ outputs both $A(x)$ and a string τ that vouches the correctness of $A(x)$. Though τ may be extraordinarily long (τ's length exceeds the number of steps taken by A on input x), it could be inspected in $\mathcal{PCP}$-style, by reading and verifying only selectively few of its bits. As we have argued in section 2, however, ensuring that one is really working with precisely these few bits of τ requires an overall verification

time that is linear in τ's length (and thus greater than the time necessary to run A on input x).

Pros and cons of certified computation. One may view some older algorithms as specialized forms of certified computation, (i.e., applicable to certain functions only). For instance, the classical extended GCD algorithm not only outputs the greatest common divisor c of two input integers a and b, but also two integers x and y such that $ax + by = c$. Blum views such extended output as a relatively simple way of checking the validity of c. That is, by checking that indeed (1) $ax + by = c$ and that (2) c divides both a and b, one verifies that there is no divisor of a and b greater than c, and so that c actually is the greatest common divisor. Thus x and y are a sort of certificate proving the correctness of c. This certificate, however, is not sufficiently short (with respect to a, b, and c) and not sufficiently easy to verify (with respect to ordinary GCD computation).

A certified computation system can, instead, be considered as an "extended" *universal* algorithm, in the sense that it shows that certified computability is not a property enjoyed only by some special functions (such as the GCD function) but is an *intrinsic property of computation.* It should be noticed that, in addition to such "universality," a certified computation system produces relatively shorter and more easily inspectable certificates (than, say, the special ones produced by the extended GCD algorithm), but has a weaker guarantee of correctness (i.e., false certificates exist but are hard to find).

Finally, it should be appreciated that our suggested certified computation system may be impractical (as it refers to the execution history of Turing machines). One way to improve its efficiency may consist of finding a more convenient, and yet sufficient for our purposes, version or representation of the execution history of an algorithm.

Assumptions and implementations. The constructability of certified computation systems is implied by CS proof systems with a random oracle or CS proof systems with a random string. As explained below, if the latter proof systems exist, then their use in the current context does not require trusting the randomness of the extra string or the computational limitation of the prover.

Indeed, in this application, the random string need not be agreed upon by both prover and verifier (by means of some possibly difficult negotiation) nor chosen by means of a trusted third party or process. When seeking reassurance that indeed $y = A(x)$, the user U of a certified computation system $(\mathcal{P}, \mathcal{V})$ controls both $\mathcal{P}$ and $\mathcal{V}$, and thus can choose their common string in a way that he believes to be genuinely random, without "asking for their consent."

In addition, the physical device D implementing algorithm $\mathcal{P}$ "sits on top of user U's desk" and produces its output within a "reasonable time" monitored by U. Therefore, user D knows for a fact that D does not comprise more than 2^k gates and that it takes less than 2^k steps of computation for a security parameter k set by U.

In sum, if CS proofs with a random string exist, they yield certified computation systems that de facto offer the same guarantees of a probabilistic algorithm.

Conceivable applications of certified computation. Certified computation can in principle be quite useful when "contracting out" computer time. Indeed, consider an algorithm A that we believe to be correct but is very time-consuming. Then, we can hire a supercomputing company for executing A on a given input x on their computers and agree that we will pay for their efforts if they give us back the value $y = A(x)$ together with a certificate of correct execution, that is, a CS certificate of "$A(x) = y$ in t steps." Note that, because such a certificate also vouches for the

number of steps taken by A's execution, it is easy for the supercomputing company to charge according to the amount of computation actually invested.

Certified computation may also facilitate the verification of certain mathematical theorems proven with the help of a computer search (as in the case of the four-color theorem). For instance, the proof may depend on a lemma stating that there is no sparse graph with less than fifty nodes possessing a given property, and the lemma could be proved by means of an exhaustive search taking a few years of computing. Often, algorithms performing an exhaustive search are sufficiently simple so that we can be confident of the correctness of their design. Thus rather than (1) asking the reader of trusting that such a search has been done and has returned a negative result, or (2) asking the reader to perform such an extensive computation himself, one might publish together with the rest of the proof a compact and easily verifiable certificate of correct execution relative to a random oracle. If the security parameter were chosen to be, say, 1,000, then even the most skeptic reader might believe that no one has invested $2^{1,000}$ steps of computation in order to find a false certificate nor that he has succeeded in finding one by relying on a probability of success less than $2^{-1,000}$.

7. Concluding remarks.

Are CS proofs really proofs? In our minds this question really goes together with an older one: do probabilistic algorithms [34, 30] really compute? There is a sense in which both answers should be NO. These negative answers, in our opinion, may stem from two different reasons: (1) a specific interpretation of the words "proof" and "computation," and (2) our mathematical tradition. The first reason is certainly true but also "harmless." The second is more "dangerous" and less acceptable: not because it is false that these notions break with a long past, but because the unchallenged length of a tradition should not be taken as implying that specific formalizations of fundamental intuitions are "final." Indeed, we believe that even fundamental intuitions cannot be divorced from the large historical contexts in which they have arisen, and we expect that they will change with the changing of these contexts. And we believe and hope that, with time, CS proofs will be regarded to be as natural as probabilistic computations.

Truths versus proofs. According to our highest-level goal, CS proofs propose a new relationship between proving and deciding. In the thirties, Turing suggested that establishing (to yourself) the truthfulness of a mathematical statement consists of running a proper (accepting) algorithm. Today, we suggest that proving (to others) a mathematical statement consists of *feasibly speeding up* the verification of the result of any given accepting algorithm. That is, (1) proofs should make verifying the result of any accepting computation exponentially faster than the same accepting computation, but (2) proofs should not be more time-consuming to find than the accepting computations whose result verification they wish to facilitate. This, in our opinion, is an appealing relationship between proving and accepting, and one that guarantees that proving is both a useful and a distinct notion.

Living with error. In order to guarantee this "feasible speed up," CS proofs replace the traditional notion of a proof with a *computational* one. While CS proofs of true statements always exist, are suitably short, and are feasibly found, proofs of false statements either do not exist or are extraordinarily hard to find. Indeed, a CS proof is a short string that can be thought of as a "compressed version" of a long accepting computation. But the same conciseness that gives CS proofs their distinctive advantage also causes them to "lose quality" with respect to the accepting computations they compress: it makes them vulnerable to the possibility of error

(though in a controllable way).[38]

To be sure, the possibility of inconsistency should not be taken lightly. But after realizing that the coherence of a sufficiently rich mathematical system cannot be decided within it, perhaps we should switch to *managing error* rather than trying desperately to ban it!

Acknowledgments. This paper is dedicated to Oded Goldreich, friend, scientist, and vigilant guardian angel. Most sincere thanks also go to Allan Borodin, Steve Cook, Shafi Goldwasser, Leonid Levin, and Michael Rabin for encouraging and criticizing this research in different ways and at various stages of its development. Thanks also to Shai Halevi, Leo Reyzin, Janos Makowski, and Ray Sidney for their useful comments.

REFERENCES

[1] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, in Proceedings of the 33rd IEEE Conference on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 14–23.

[2] S. Arora and M. Safra, *Probabilistic checking of proofs*, in Proceedings of the 33rd IEEE Conference on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 2–13.

[3] L. Babai, L. Fortnow, L. Levin, and M. Szegedy, *Checking computations in polylogarithmic time*, in Proceedings of the 23rd ACM Symposium on Theory of Computing, 1991, pp. 21–31.

[4] L. Babai and S. Moran, *Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes*, J. Comput. System Sci., 36 (1988), pp. 254–276.

[5] M. Bellare and O. Goldreich, *On defining proofs of knowledge*, in Advances in Cryptology—CRYPTO 92, Lecture Notes in Comput. Sci. 740, Springer-Verlag, Berlin, 1993, pp. 390–420.

[6] M. Bellare and S. Goldwasser, *The complexity of decision versus search*, SIAM J. Comput., 23 (1994), pp. 97–119.

[7] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, in Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, VA, 1993, pp. 62–73.

[8] M. Ben-or, S. Goldwasser, J. Kilian, and A. Wigderson, *Multi prover interactive proofs: How to remove intractability*, in Proceedings of the 20th ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 113–131.

[9] M. Blum, A. De Santis, S. Micali, and G. Persiano, *Noninteractive zero-knowledge*, SIAM J. Comput., 20 (1991), pp. 1084–1118.

[10] M. Blum, P. Feldman, and S. Micali, *Noninteractive zero-knowledge proof systems and applications*, in Proceedings of the 20th ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 103–112.

[11] M. Blum and S. Kannan, *Designing programs that check their work*, in Proceedings of the 21st ACM Symposium on Theory of Computing, Seattle, WA, 1989, pp. 86–97.

[12] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 549–595.

[13] G. Brassard, D. Chaum, and C. Crepeau, *Minimum disclosure proofs of knowledge*, J. Comput. System Sci., 37 (1988), pp. 156–189.

[14] C. Cachin, S. Micali, and M. Stadler, *Computational Private-Information Retrieval With Polylogarithmic Communication*, in preparation, 1998.

[38] Let me emphasize that, though very natural and useful, the computational boundedness of a cheating prover is not, per se, a goal of our notion of a proof. Rather, it is the Trojan horse that lets us sneak in and achieve our desiderata. (It is actually very important to establish whether some types of CS proofs exist when a cheating prover can compute for an unbounded amount of time.)

In sum, the notion of a true theorem has not changed. What we advocate changing is the notion of what it means to *prove that a theorem is true*.

[15] R. Canetti, O. Goldreich, and S. Halevi, *The random oracle methodology, revisited*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, ACM, New York, 1998, pp. 209–218.
[16] S. Cook, *The complexity of theorem proving procedures*, in Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1971, pp. 151–158.
[17] U. Feige, S. Goldwasser, L. Lovasz, S. Safra, and M. Szegedy, *Approximating clique is almost NP-complete*, in Proceedings of the 32nd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 2–12.
[18] A. Fiat and A. Shamir, *How to prove yourselves: Practical solutions of identification and signature problems*, in Advances in Cryptology—CRYPTO 86, Lecture Notes in Comput. Sci. 263, Springer-Verlag, Berlin, 1987, pp.186–194.
[19] L. Fortnow, J. Rompel, and M. Sipser, *On the power of multi-prover interactive protocols*, Theoret. Comput. Sci., 134 (1994), pp. 545–557.
[20] O. Goldreich, S. Micali, and A. Wigderson, *Proofs that yield nothing but their validity or all languages in $\mathcal{NP}$ have zero-knowledge proof systems*, J. ACM, 38 (1991), pp. 691–729. (A preliminary version of this paper, under the title *Proofs that yield nothing but their validity and a methodology for cryptographic protocol design*, appeared in Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science, New York, NY, 1986, pp. 174–187.)
[21] S. Goldwasser, S. Micali, and C. Rackoff, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208. (An earlier version of this result informally introducing the notion of a proof of knowledge appeared in Proceedings of the 17th Annual IEEE Symposium on Theory of Computing, IEEE Computer Society, Los Alamitos, CA, 1985, pp. 291–304. Earlier yet versions include *Knowledge Complexity*, submitted to the 25th IEEE Annual Symposium on the Foundations of Computer Science, 1984.)
[22] S. Halevi and S. Micali, *A Stronger Notion of Proofs of Knowledge*, manuscript, 1997.
[23] J. Kilian, *A note on efficient zero-knowledge proofs and arguments*, in Proceedings of the 24th Annual ACM Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 723–732.
[24] L. Levin, *Universal sequential search problems*, Problems Inform. Transmission, 9 (1973), pp. 265–266.
[25] R. Lipton, *New directions in testing*, in Distributed Computing and Cryptography, J. Feigenbaum and M. Merritt, eds., DIMACS Ser. Discrete Math. Theory Comput. Sci. 2, AMS, Providence, RI, 1991, pp. 191–202.
[26] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.
[27] R. Merkle, *A certified digital signature*, in Advances in Cryptology—CRYPTO 89, Lecture Notes in Comput. Sci., 435, Springer-Verlag, New York, 1990, pp. 218–238.
[28] S. Micali, *CS proofs*, in Proceedings ofthe 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 436–453. (An earlier version of this paper appeared as Technical Memo MIT/LCS/TM-510. Earlier yet versions were submitted to the 25th Annual ACM Symposium on Theory of Computing, 1993, and the 34th Annual IEEE Symposium on Foundations of Computer Science, 1993.)
[29] A. Polishchuk and D. Spielman, *Nearly-linear size holographic proofs*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montreal, Canada, 1994, pp. 194–203.
[30] M. Rabin, *Probabilistic algorithms for testing primality*, J. Number Theory, 12 (1980), pp. 128–138.
[31] R. Rivest, *The MD5 Message-Digest Algorithm*, Internet Activities Board, Request for Comments 1321, 1992.
[32] *Secure Hash Standard*, Federal Information Processing Standards, Publication 180 (1993).
[33] A. Shamir, *IP = PSPACE*, J. ACM, 39 (1992), pp. 869–877.
[34] R. Solovay and V. Strassen, *A fast Monte-Carlo test for primality*, SIAM J. Comput., 6 (1977), pp. 84–85.
[35] M. Sudan, *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, Ph.D. Thesis, University of California at Berkeley, Berkeley, CA, 1992.
[36] M. Tompa and H. Woll, *Random self-reducibility and zero-knowledge interactive proofs of possession of information*, in Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 472–482.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1299–1320

THE COMPLEXITY OF MODULAR GRAPH AUTOMORPHISM*

V. ARVIND†, R. BEIGEL‡, AND A. LOZANO§

Abstract. Motivated by the question of the relative complexities of the graph isomorphism and the graph automorphism problems, we define and study the modular graph automorphism problems. These are the decision problems mod_k-GA which consist, for each $k > 1$, of deciding whether the number of automorphisms of a graph is divisible by k. The mod_k-GA problems all turn out to be intermediate in difficulty between graph automorphism and graph isomorphism.

We define an appropriate search problem corresponding to mod_k-GA and design an algorithm that polynomial-time reduces the mod_k-GA search problem to the decision problem. Combining this algorithm with an IP protocol, we obtain a randomized polynomial-time checker for mod_k-GA $\forall k > 1$.

Key words. graph isomorphism, graph automorphism, decision problems, search problems, IP protocols, program checking

AMS subject classifications. 68Q15, 68Q22, 68Q25, 20B05

PII. S0097539799358227

1. Introduction.

The graph isomorphism (GI) problem consists of determining whether two graphs are isomorphic. It is well known that GI is in NP, but despite decades of study by mathematicians and computer scientists, it is not known whether GI is in P or is NP-complete. Many researchers conjecture that GI's complexity lies somewhere between P and NP-complete. There are several other decision problems (some graph-theoretic and others group-theoretic in nature) that also are not known to be in P or NP-complete. One such problem which is closely related to GI is graph automorphism (GA) which consists of deciding whether a graph has a nontrivial automorphism. Regarding the relative complexities of GA and GI, it is known that GA is polynomial-time many-one reducible to GI. On the other hand, GI is not known to be even polynomial-time Turing reducible to GA (see [15] for these and related results). However, in [17] it is shown that GI is polynomial-time reducible to the problem of computing the number of automorphisms of a graph.

The notion of program checking was introduced by Blum and Kannan [7] as an algorithmic alternative to program verification. Since then, the design of efficient checkers for various computational problems has rapidly grown into a discipline of algorithm design [7, 8]. One of the first program checkers in [7] was a randomized polynomial-time checker for GI. It is an outstanding open question in the area if NP-complete problems have efficient program checkers. This can be construed as ad-

*Received by the editors July 7, 1999; accepted for publication (in revised form) June 4, 2000; published electronically October 31, 2000. An early version of the results was presented at the 15th Symposium on Theoretical Aspects of Computer Science, Paris, 1998.

http://www.siam.org/journals/sicomp/30-4/35822.html

†Institute of Mathematical Sciences, C. I. T. Campus, Chennai 600113, India (arvind@imsc.ernet.in). Part of the work was done while this author was visiting U.P.C., Barcelona, and was supported by grant ABM/acs/PIV98-36 from the Direcció General de Recerca.

‡NEC Research Institute, 4 Independence Way, Princeton, NJ 08540-6634, and Department of Computer and Information Sciences, College of Science and Technology, Temple University, Wachman Hall (038-24), Room 303, 1805 N. Broad St., Philadelphia, PA 19122 (beigel@joda.cis.temple.edu).

§Universitat Politècnica de Catalunya, Dept. LSI, UPC, Jordi Girona 1-3, Mòdul C6, 08034 Barcelona, Catalonia, Spain (antoni@lsi.upc.es). The work of this author was supported in part by the E.U. through ESPRIT project (LTR 20244) and by DGICYT (Koala project, with PB95-0787) and CICIT (TIC97-1475-CE).

ditional evidence that GI is not NP-complete. Later, in [16] it was shown that GA has a *nonadaptive* checker. In other words, the checker can make all its queries to the program in parallel, hence enabling it to be fast in parallel (in NC, to be precise). It is an open question whether GI too has a nonadaptive checker. The apparent bottleneck here is that the search problem for GI is not known to be polynomial-time truth-table reducible to the decision problem for GI; i.e., it is not known if the search problem is reducible to the decision problem via parallel (nonadaptive) queries. A related question is that of computing automorphisms of a graph in polynomial time with oracle access to either GI or GA. Indeed, it is shown in [16] that with nonadaptive queries to GA, the lexicographically first automorphism of a graph can be computed in polynomial time, whereas computing the lexicographically last automorphism is as hard as GI. Also see [1], in which the complexity of computing k nontrivial automorphisms of a graph is studied.

With these issues in mind, a natural next step in investigating the relationship between GI and GA is to consider exactly how much we need to know about the number of automorphisms of a graph in order to solve the GI problem. This motivates us to define and study modular GA problems. Let $Aut(G)$ denote the automorphism group of the graph G.

DEFINITION 1. *For any k, let* $\text{mod}_k\text{-GA} = \{G : |Aut(G)| \equiv 0 \pmod{k}\}$.

We show in Theorems 4 and 5 that for any $k > 1$, GA $\leq_m^p$ mod_k-GA $\leq_m^p$ GI; thus the mod_k-GA problems are intermediate in difficulty between GA and GI. It is an open question whether any of the mod_k-GA problems is polynomial-time equivalent to GA or GI. We conjecture that mod_k-GA is *not* polynomial-time equivalent to GA or GI for any $k > 1$. An evidence that some of the mod_k-GA problems could be actually harder than GA is the observation that tournament isomorphism (GI for tournament graphs) is many-one reducible to mod_2-GA. This follows from the fact that the automorphism group of any tournament is of odd size [15], which in turn implies that two tournaments are isomorphic if and only if the automorphism group of their disjoint union contains an order-2 permutation (which must switch the two graphs).

The proof technique we use is a combination of elementary properties of finite graphs (see, e.g., [12]) and the theory of finite groups of prime-power order (see, e.g., a group theory text like [18]).

The layout of the paper is as follows. Section 2 contains the preliminaries. In section 3, we prove that the mod_k-GA problems are located between GA and GI. In section 4, we show that search is polynomial-time Turing equivalent to decision for $\underline{\text{mod}_k\text{-GA}}$, and in section 5 we use this result in combination with an IP protocol for $\overline{\text{mod}_p\text{-GA}}$ to obtain an efficient program checker for mod_k-GA. Notice that although both GA and GI have program checkers ([16] and [7], resp.) and mod_k-GA is intermediate in complexity, it does not necessarily imply that mod_k-GA has a program checker [7].

2. Preliminaries. In this paper, by a graph we mean a finite directed graph[1] (see, for example, [12] or any other standard text on graph theory for basic definitions). For a graph G, let $V(G)$ denote its vertex set and $E(G)$ denote its edge set. A permutation π on $V(G)$ is an automorphism of the graph G if $(u, v) \in E(G) \iff (\pi(u), \pi(v)) \in E(G)$. The set of automorphisms $Aut(G)$, of a graph G, is a subgroup

[1] We consider the problems GI, GA, and mod_k-GA on directed graphs. However, our results hold for these problems on undirected graphs as well.

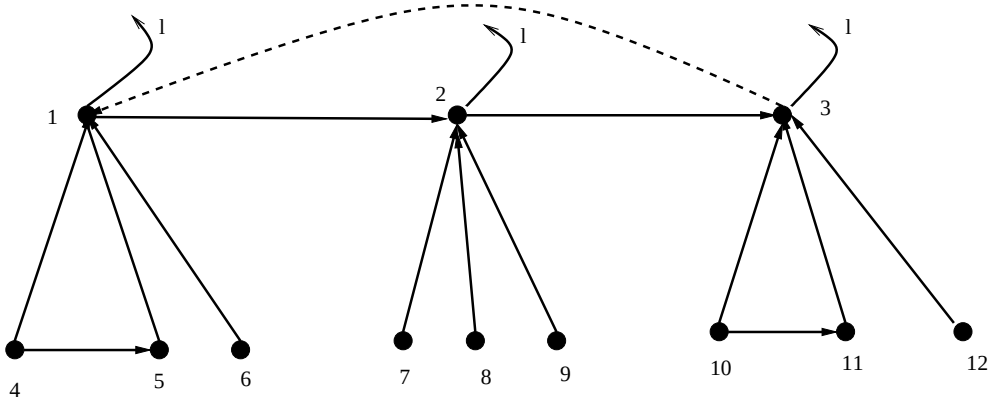


FIG. 1. *Examples of Cycle and Path: The above graph, including the dashed edge, is Cycle*(G_1, G_2, G_3)*, where* G_1*,* G_2*, and* G_3 *are the graphs induced by* $\{4, 5, 6\}$*,* $\{7, 8, 9\}$*, and* $\{10, 11, 12\}$*, respectively. Excluding the dashed edge in the above graph gives Path*(G_1, G_2, G_3)*. The underlying cycle* C_3 *(resp., path* P_3*) is induced by vertices* 1*,* 2*, and* 3*, which are labeled* l*.*

of the permutation group on $V(G)$. The identity automorphism of any graph will be denoted by id.

Let X be a list of vertices in $V(G)$ for a graph G. By $G_{[X]}$ we mean the graph G with distinct labels attached to the vertices in X. We can define these labels as given in [15, p. 8]: If $|V(G)| = n$, label j, $1 \leq j \leq n$, is defined as a graph L_j consisting of $2n+j+3$ vertices which has a directed path P of length $2n+3$ and a directed path of length j emanating from the $(n+2)$nd vertex of path P. Notice that L_j is *rigid* (i.e., it has no nontrivial automorphisms). The size of $G_{[X]}$ is $O(n^2)$. This definition of labels guarantees that every automorphism of $G_{[X]}$ must pointwise fix the vertices of X. Thus $Aut(G_{[X]})$ is isomorphic to the subgroup of $Aut(G)$ which pointwise fixes the vertices in X. Furthermore, given an automorphism of $Aut(G_{[X]})$, the corresponding automorphism of $Aut(G)$ can be efficiently (i.e., in polynomial time) constructed. Given two lists of vertices $X, Y \subseteq V(G)$, we usually assume that the graphs $G_{[X]}$ and $G_{[Y]}$ have the same labels in vertices occupying the same relative positions in X and Y.

DEFINITION 2. *Let* $G_1, \ldots, G_n$ *be* n *graphs.*

- *Let* P_n *be a directed simple path of* n *new vertices* $v_1, v_2, \ldots, v_n$*, where each vertex* v_i *is labeled with a single label* l*. The graph Path*$(G_1, \ldots, G_n)$ *is obtained by taking one copy of each of the graphs* $G_1, \ldots, G_n$ *and, for* $1 \leq i \leq n$*, attaching all the vertices of* G_i *to* v_i*. See Figure* 1 *for an example.*
- *Let* C_n *be the directed simple cycle on* n *new vertices* $v_1, v_2, \ldots, v_n$*, with each vertex* v_i*,* $1 \leq i \leq n$*, labeled with a single label* l*. The graph Cycle*$(G_1, \ldots, G_n)$ *is obtained by taking one copy of each of the graphs* $G_1, \ldots, G_n$ *and, for* $1 \leq i \leq n$*, attaching all the vertices of* G_i *to* v_i*. See Figure* 1 *for an example.*

In both $Path(G_1, \ldots, G_n)$ and $Cycle(G_1, \ldots, G_n)$, since vertices $v_1, v_2, \ldots, v_n$ are labeled with l, any automorphism of these graphs must map $\{v_1, v_2, \ldots, v_n\}$ onto itself. Thus, any automorphism of $Path(G_1, \ldots, G_n)$ ($Cycle(G_1, \ldots, G_n)$) when restricted to $\{v_1, v_2, \ldots, v_n\}$ is an automorphism of P_n (respectively, C_n). It follows that an automorphism of $Path(G_1, \ldots, G_n)$ cannot permute the copies of $G_1, \ldots, G_n$, while an automorphism of $Cycle(G_1, \ldots, G_n)$ can permute them but only along the cycle C_n.

The reducibilities discussed in this paper are the standard polynomial-time Turing

and many-one reducibilities. Formal definitions of these and other standard notions in complexity theory can be found in [3, 2].

We now recall some useful complexity-theoretic concepts. A set $A \subseteq \Sigma^*$ is a *d-cylinder* if there is a polynomial-time computable function OR that takes a list of strings $x_1, x_2, \ldots, x_m$ as argument and produces a string y such that

$$\mathrm{OR}(x_1, x_2, \ldots, x_m) = y \in A \iff \exists i \ : \ 1 \leq i \leq m \ : \ x_i \in A.$$

Similarly, a set $A \subseteq \Sigma^*$ is a *c-cylinder* if there is a polynomial-time computable function AND that takes a list of strings $x_1, x_2, \ldots, x_m$ as argument and produces a string y such that

$$\mathrm{AND}(x_1, x_2, \ldots, x_m) = y \in A \iff (\forall i \ : \ 1 \leq i \leq m)[x_i \in A].$$

PROPOSITION 3 (see [10, 16]). GI *is a d-cylinder and a c-cylinder.*

It is interesting to note here that GA is known to be a d-cylinder [15], but it is open whether it is also a c-cylinder. The relative complexity of decision and search for NP problems is well studied [9, 4, 19, 6]. For instance, it is known that search and decision are polynomial-time Turing equivalent for all NP-complete problems. In particular, we recall that for GI search is polynomial-time Turing reducible to decision [19], whereas for GA a stronger result holds: search is *nonadaptively* polynomial-time reducible to decision [16].

3. Locating the mod$_k$-GA problems. We show in this section that mod$_k$-GA is located between GA and GI $\forall k > 1$.

THEOREM 4. *For all* $k > 1$, GA $\leq_m^p$ mod$_k$-GA.

Proof. Given a graph G, we define for every i, j with $1 \leq i < j \leq n$ the graph $H_{i,j} = \mathit{Cycle}(G_{[\{i\}]}, G_{[\{j\}]}, \ldots, G_{[\{j\}]})$ which contains one copy of $G_{[\{i\}]}$ and $k-1$ copies of $G_{[\{j\}]}$. Further, let H be obtained by applying the *Path* operator to all the graphs $H_{i,j}$ with $1 \leq i < j \leq n$. We claim that G has a nontrivial automorphism if and only if H is in mod$_k$-GA.

Suppose that G has a nontrivial automorphism φ. There exist two vertices i and j such that $\varphi(i) = j$. Notice that $H_{i,j}$ has a nontrivial automorphism α that cyclically permutes the k graphs in $\mathit{Cycle}(G_{[\{i\}]}, G_{[\{j\}]}, \ldots, G_{[\{j\}]})$ as follows. The automorphism α maps the first graph $G_{[\{i\}]}$ to $G_{[\{j\}]}$ by φ. It maps each of the first $k-2$ copies of $G_{[\{j\}]}$ to the next copy of $G_{[\{j\}]}$ by the identity automorphism. Finally, α maps the last copy of $G_{[\{j\}]}$ back to $G_{[\{i\}]}$ by the automorphism φ^{-1}.

The order of α is k since the vertices in $H_{i,j}$ are moved in a cyclic way through the k subgraphs. In fact, α is a product of k-cycles. Thus, $H_{i,j} \in$ mod$_k$-GA. Since $|\mathit{Aut}(H)| = \prod_{1 \leq i < j \leq n} |\mathit{Aut}(H_{i,j})|$, it follows that $H \in$ mod$_k$-GA.

For the converse, assume that $H \in$ mod$_k$-GA. Then H has a nontrivial automorphism, say, α. Notice that α must induce a nontrivial automorphism β in one of its subgraphs $H_{i,j}$. Since $H_{i,j} = \mathit{Cycle}(G_{[\{i\}]}, G_{[\{j\}]}, \ldots, G_{[\{j\}]})$, there are two possibilities: either β induces a nontrivial automorphism of $G_{[\{i\}]}$ or $G_{[\{j\}]}$, or β maps $G_{[\{i\}]}$ to some copy of $G_{[\{j\}]}$. In either case we get a nontrivial automorphism of G. □

Mathon [17] has shown that $|\mathit{Aut}(G)|$ is polynomial-time computable with GI as oracle. From this it easily follows that mod$_k$-GA $\leq_T^p$ GI. In the next theorem, we strengthen this to a $\leq_m^p$-reduction using some permutation group theory.

THEOREM 5. *For all* $k > 1$, mod$_k$-GA $\leq_m^p$ GI.

We need a couple of definitions and group-theoretic lemmas before we prove Theorem 5. Let A be a subgroup of S_n and let $[n]$ denote the set $\{1, 2, \ldots, n\}$. A subset

$X \subseteq [n]$ is *A-invariant* if $g(X) = X\ \forall g \in A$. If $X \subseteq [n]$ is A-invariant, then consider the action of A *restricted* to X. This gives rise to a subgroup of the symmetric group S_X, which we denote by A^X. A useful obvious property is that $|A^X| \leq |A|$ $\forall A$-invariant sets X.

LEMMA 6. *Let A be a subgroup of S_n such that $|A| = m$. Then there exists an A-invariant subset $X \subseteq [n]$ with $|X| \leq m \log m$, such that A is isomorphic to A^X.*

Proof. Consider the following procedure for constructing the set X:

$X := \emptyset$;
while $\exists i \notin X : |A^X| < |A^{X \cup A(i)}|$ **do**
    ($A(i)$ denotes the orbit of i under A *)*
    Pick such an i;
    $X := X \cup A(i)$
endwhile

First we claim that, as a loop invariant, X is always an A-invariant subset of $[n]$. To see this, notice that it holds at the beginning when X is empty, and if X is A-invariant, then so is $X \cup A(i)$ since we are including an entire A-orbit in the set.

Next, suppose X is A-invariant and $i \notin X$ is some index. Consider the mapping φ from $A^{X \cup A(i)}$ to A^X which maps an element of $A^{X \cup A(i)}$ to its restriction to X. Since X is A-invariant, it is easy to verify that φ is a surjective homomorphism from $A^{X \cup A(i)}$ to A^X. It follows that $|A^X|$ divides $|A^{X \cup A(i)}|$. Suppose now, at some stage of the while loop, i is an index such that $|A^X| < |A^{X \cup A(i)}|$. Then it must hold that $2|A^X| \leq |A^{X \cup A(i)}|$. Thus we have argued that every time X increases by including an orbit $A(i)$ in it, the size of the group A^X increases by at least a factor of 2. Thus the assignment $X := X \cup A(i)$ is executed at most $\log m$ times, implying also that the procedure must terminate. Since the size of any orbit $A(i)$ is bounded by $|A|$, it follows that the procedure terminates with an A-invariant set X such that $|X| \leq m \log m$. Let X be the set computed when the while-loop is exited. To complete the proof we must show that A^X is isomorphic to A. Consider the canonical surjective homomorphism ψ from A to A^X, which maps a given element of A to its corresponding restriction to X. To show that this homomorphism is an isomorphism we must argue that $Ker(\psi)$ is (id). Suppose $g \in Ker(\psi)$ is a nontrivial element. Then there is $i \notin X$ such that $g(i) \neq i$. This in turn implies that the surjective homomorphism φ from $A^{X \cup A(i)}$ to A^X, which maps an element of $A^{X \cup A(i)}$ to its restriction to X, has a nontrivial kernel with $g \in Ker(\varphi)$. Consequently, $|A^X| < |A^{X \cup A(i)}|$. Thus, both X and i satisfy the while-loop condition, contradicting the fact that the while-loop has terminated. This completes the proof of this lemma. □

LEMMA 7. *Let A be a finite group. Let $X = \{a_1, a_2, \ldots, a_t\}$ and $Y = \{b_1, b_2, \ldots, b_t\}$ be two subsets of A such that $\langle X \rangle \cap \langle Y \rangle = \{\text{id}\}$ and $a_i b_j = b_j a_i$ for $1 \leq i, j \leq t$. Then $|\langle X \rangle|$ divides the order of the group $\langle \{a_i b_i : 1 \leq i \leq t\} \rangle$.*

Proof. Let H denote the subgroup of A generated by $\{a_i b_i \mid 1 \leq i \leq t\}$, K denote the subgroup of A generated by $\{a_i \mid 1 \leq i \leq t\}$, and L denote the subgroup of A generated by $\{b_i \mid 1 \leq i \leq t\}$. Notice that since $a_i b_j = b_j a_i$ for $1 \leq i, j \leq t$, we have $KL = LK$, and therefore the set KL is actually a subgroup of A. Next, notice that, by definition of H, any $x \in H$ can be written as a product of elements from the generator set $\{a_i b_i \mid 1 \leq i \leq t\}$. Using $a_i b_j = b_j a_i$ for $1 \leq i, j \leq t$ as a rewrite rule, this product of generators expressing x can be rewritten as ay, where $a \in K$ and $y \in L$. It follows that $H \subseteq KL$. Consider the following map ψ from the group H to the group K, defined as follows:

$$\forall x \in H\ :\ \psi(x) = a \text{ where } x = ay, \text{ with } a \in K \text{ and } y \in L.$$

We claim that ψ is a well-defined surjective homomorphism from H to K. We first show that ψ is well defined. Suppose there are two distinct elements $a, a' \in K$ such that $x = ay = a'y'$ for elements $y, y' \in L$. This implies, by cancellation laws, that $a^{-1}a' = yy'^{-1}$, which belongs to both K and L. Since $K \cap L = \{\mathrm{id}\}$, we have $a = a'$. Thus ψ is well defined. To see that ψ is a homomorphism is routine: we can easily check that $\psi(xx') = \psi(x)\psi(x')$ and that $\psi(x^{-1}) = (\psi(x))^{-1}$ hold using the rewrite rules $a_ib_j = b_ja_i$ for $1 \le i, j \le t$. To see that ψ is surjective, let $a \in K$ be any element. We can express a as a product $\Pi_{1\le r\le m}a_{i_r}$ for indices $i_r \in [t]$. Consider the element $x = \Pi_{1\le r\le m}a_{i_r}b_{i_r} \in H$. It is easy to see that $\psi(x) = a$.

Thus, by the fundamental theorem of homomorphisms, it follows that $H/Ker(\psi)$ is isomorphic to K. Therefore, $|H/Ker(\psi)| = |K|$. It follows that $|K|$ divides $|H|$, which proves the lemma. □

Proof of Theorem 5. First, we argue that it suffices to show that mod_{p^l}-GA $\le^p_m$ GI for all prime p and $l > 0$. To see this, let $\prod_{1\le j\le r} p_j^{l_j}$ be the prime factorization of k. Clearly, a graph $G \in \mathrm{mod}_k$-GA if and only if $G \in \bigcap_{j=1}^{r} \mathrm{mod}_{p_j^{l_j}}$-GA. Thus, if $\mathrm{mod}_{p_j^{l_j}}$-GA $\le^p_m$ GI for $1 \le j \le r$, it follows that mod_k-GA $\le^p_m$ GI, since GI is a c-cylinder.

We next prove a useful group-theoretic claim. Let G be a graph on n vertices and let f be a partial permutation on $[n]$ (i.e., f is defined on a subset of the domain $[n]$ and can be extended to a permutation in S_n). Then we call f a *partial automorphism* of G if f can be extended to an automorphism of G.

Claim. *Let p be a fixed prime and $l > 0$. A graph G on n vertices is in* mod_{p^l}-GA *if and only if there exist a set $X \subseteq [n]$ with $|X| \le p^l(\log p^l)$ and a subgroup $K = \{a_1, a_2, \ldots, a_{p^l}\}$ of S_X such that each $a_i \in K$ is a partial automorphism of G.*

Proof. Let $G \in \mathrm{mod}_{p^l}$-GA be an n vertex graph. Since p^l divides $|Aut(G)|$, by Sylow's theorem $Aut(G)$ has a subgroup A of size p^l. By Lemma 6, there is an A-invariant set $X \subseteq [n]$ with $|X| \le p^l(\log p^l)$, such that A^X is isomorphic to A. Let $A^X = \{a_1, a_2, \ldots, a_{p^l}\}$. Furthermore, it also follows that A^X is a subgroup of S_X where each $a_i \in A^X$ is a partial automorphism of G. Conversely, suppose there is $X \subseteq [n]$ with $|X| \le p^l(\log p^l)$ and a subgroup $K = \{a_1, a_2, \ldots, a_{p^l}\}$ of S_X where each $a_i \in K$ is a partial automorphism of G. Then for each i with $1 \le i \le p^l$, there is a $b_i \in S_{[n]-X}$ such that $a_ib_i \in Aut(G)$. We can now apply Lemma 7 to the elements $\{a_i\}_{1\le i\le p^l}$ and $\{b_i\}_{1\le i\le p^l}$, since therequired conditions are fulfilled. Consequently, $|\langle\{a_ib_i : 1 \le i \le p^l\}\rangle|$ is divisible by p^l. Since $\langle\{a_ib_i : 1 \le i \le p^l\}\rangle$ is a subgroup of $Aut(G)$, it follows that p^l divides $|Aut(G)|$.

Now, note that the language $B = \{(G, f) : f$ is a partial automorphism of the graph $G\}$ is $\le^p_m$-equivalent to GI (for details, see [15, Theorem 1.18]). We will give a truth-table reduction from mod_{p^l}-GA to B, where the truth-table is a disjunction of conjunctions. Since the language B is $\le^p_m$-reducible to GI and since GI is both a c-cylinder and a d-cylinder, it follows that mod_{p^l}-GA is $\le^p_m$-reducible to GI. We describe below the said reduction of mod_{p^l}-GA to B as a logical expression, which is easily seen to describe a disjunction-of-conjunctions truth-table reduction:

$$G \in \mathrm{mod}_{p^l}\text{-GA} \iff (\exists\, X \subseteq [n] : |X| \le p^l \log p^l)$$
$$(\exists \text{ subgroup } K < S_X : |K| = p^l)(\forall a \in K)[(G, a) \in B].$$

This completes the proof of Theorem 5. □

Since GA is not known to be a c-cylinder and, moreover, the set B defined above is $\le^p_m$-equivalent to GI, it is unlikely that the approach taken in the above proof can

be used to show that mod_k-GA is $\leq_m^p$-reducible to GA.

We next show a useful result relating the complexity of mod_k-GA to mod_l-GA when l is a factor of k.

LEMMA 8. *For $k, l > 1$ such that l divides k, mod_l-GA $\leq_m^p$ mod_k-GA.*

Proof. Let G be an instance of mod_l-GA, where $k = lm$ for some $m > 1$. Let C_m denote the directed cycle of length m. Consider the graph $Path(G, C_m)$. By construction, any automorphism of $Path(G, C_m)$ must map G to itself and C_m to itself. Thus, $|Aut(Path(G, C_m))| = |Aut(G)||Aut(C_m)| = |Aut(G)|m$. Thus k divides $|Aut(Path(G, C_m))|$ if and only if l divides $|Aut(G)|$. It follows that $G \mapsto Path(G, C_m)$ is a $\leq_m^p$-reduction from mod_l-GA to mod_k-GA. ∎

4. Computing solutions for mod_k-GA instances. Let $\prod_{1 \leq j \leq m} p_j^{e_j}$ be the prime factorization of k. We define an NP *witness* of the membership of a graph G in mod_k-GA as m subgroups $\{A_1, A_2, \ldots, A_m\}$ of $Aut(G)$, where A_i is listed as a set of permutations and $|A_i| = p_i^{e_i}$ for each i. We are interested in the *search* problem of computing an NP witness for $G \in \mathrm{mod}_k$-GA. Clearly, given a collection $\{A_1, A_2, \ldots, A_m\}$ of permutation sets, it can be verified to be a witness for G in time polynomial in $|V(G)|$. Notice that, for an instance G in mod_k-GA, it is straightforward to design a polynomial-time algorithm with GI as oracle for computing a witness. This can be done, for instance, by adapting ideas in the proof of Theorem 5 of the previous section where we showed how mod_k-GA is many-one reducible to GI. The result of this section is stronger since we will use the weaker oracle mod_k-GA for the same task. Hence, for the above definition of search problem, it follows that search is polynomial-time Turing reducible to decision for mod_k-GA.

Intuitive description of the algorithmic task. Our aim is to design a polynomial-time algorithm that, given G in mod_k-GA as input, computes the NP witness described above with oracle access to mod_k-GA. Clearly, a graph $G \in \mathrm{mod}_k$-GA if and only if $G \in \bigcap_{j=1}^{m} \mathrm{mod}_{p_j^{e_j}}$-GA. Also, by Lemma 8, $\mathrm{mod}_{p_j^{e_j}}$-GA is $\leq_m^p$-reducible to mod_k-GA $\forall j$. Thus, it suffices to design a polynomial-time algorithm for the case when $k = p^l$ for some prime p. In this case, given $G \in \mathrm{mod}_{p^l}$-GA as input, the witness that the algorithm must compute with oracle mod_{p^l}-GA is a p^l-subgroup A of $Aut(G)$. (We actually do better; the oracle our algorithm will access is mod_p-GA, which is $\leq_m^p$-reducible to mod_{p^l}-GA by Lemma 8.) By Sylow's theorem, such a subgroup A exists. By standard p-group theory, if A is a p^l-subgroup of $Aut(G)$, we have a subgroup chain: $A_0 = (\mathrm{id}) < A_1 < \cdots < A_l = A$, where $|A_i| = p^i$ and $\forall i$, A_i is a normal subgroup of A_{i+1}. Our algorithm will compute such a subgroup A inductively by computing each subgroup in such a sequence $A_1, \ldots, A_i$ in increasing order of i until finally $A_l = A$ is computed. Thus it suffices to perform the following algorithmic task: Given a p^i-subgroup A_i of $Aut(G)$ for a graph G and $0 \leq i \leq p-1$, compute a p^{i+1}-subgroup of $Aut(G)$ if $G \in \mathrm{mod}_{p^{i+1}}$-GA.

The following results about finite groups of prime-power order indicate how this can be done. First, we recall a stronger form of the third Sylow theorem [14, Theorem 11.1.1, p. 66]: if p^{i+1} divides $|Aut(G)|$, then for each subgroup of order p^i there is some subgroup of order p^{i+1} containing it.

THEOREM 9. (Sylow's theorem [14]). *If p^{i+1} divides the order of a finite group H, then for each subgroup K of H of order p^i there is some subgroup of H of order p^{i+1} containing K.*

We derive from this the following easy fact, which is useful for our particular application.

FACT 10. *Let A_i be a p^i-subgroup of $Aut(G)$. A graph G is in* $\mathrm{mod}_{p^{i+1}}$-GA *if*

and only if the following three conditions hold for some $g \in Aut(G)$.

(a) $g \notin A_i$.

(b) *g has order p^s for some $s \leq i+1$.*

(c) $gA_ig^{-1} = A_i$.

Moreover, if $t+1$ is the smallest positive integer such that $g^{p^{t+1}} \in A_i$ for $g \in Aut(G)$ satisfying (a), (b), *and* (c), *the set $A_{i+1} = \bigcup_{j=0}^{p-1} A_i g^{jp^t}$ is a p^{i+1}-subgroup of $Aut(G)$ containing A_i as a normal subgroup.*

Proof. Suppose $G \in \mathrm{mod}_{p^{i+1}}$-GA and A_i is a p^i-subgroup of $Aut(G)$. By Theorem 9, there is a p^{i+1}-subgroup A_{i+1} of $Aut(G)$ containing A_i. Furthermore, A_i is normal in A_{i+1} since every p^i-subgroup of a p^{i+1}-subgroup is normal (see, e.g., [18, Theorem 4.6, p. 75]). Thus, the quotient group A_{i+1}/A_i exists and is a cyclic group of order p. Let A_ig be a generator for A_{i+1}/A_i for some $g \in A_{i+1} - A_i$. Such an element g clearly fulfills the conditions (a), (b), and (c).

To prove the reverse implication suppose there exists $g \in Aut(G)$ satisfying the above three conditions. Let A be the group generated by $A_i \cup \{g\}$. Since $gA_ig^{-1} = A_i$, A_i is a normal subgroup of A. Furthermore, $o(A_ig) = p^{t+1}$ for some $0 \leq t < s$. It follows that $A_ig^{p^t}$ is an order-p element of the quotient group A/A_i. Let $A_{i+1} = \bigcup_{j=0}^{p-1} A_i g^{jp^t}$. Clearly, $A_i \subset A_{i+1} \subset A$. It is easy to check that A_{i+1} is a p^{i+1}-subgroup of $Aut(G)$. Consequently, A_i is a normal subgroup of A_{i+1}. ∎

By Fact 10 our algorithmic task is reduced to the following: Given as input $G \in \mathrm{mod}_{p^{i+1}}$-GA and A_i as a list of permutations, where A_i is a p^i-subgroup of $Aut(G)$, compute $g \in Aut(G)$ satisfying properties (a), (b), and (c) of Fact 10. We first design an algorithm for the case $i = 0$. We will use this as a subroutine in our algorithm for the general case.

We first describe a property of the graph gadget *Cycle* that we use extensively in this section.

LEMMA 11. *Let H and K be two graphs on n vertices, and p be a fixed prime. Consider $Cycle(H, K, \ldots, K)$ with $p-1$ copies of K. $Aut(Cycle(H, K, \ldots, K))$ is in bijective correspondence with*

$$Aut(H) \times (Aut(K))^{p-1} \cup Z_p^* \times (Aut(K))^{p-2} \times Iso(H, K) \times Iso(K, H),$$

where $\times$ denotes the Cartesian product, Z_p^* is $\{1, \ldots, p-1\}$, and $Iso(H, K)$ denotes the set of isomorphisms from H to K. Additionally, if a nontrivial $\pi \in Aut(Cycle(H, K, \ldots, K))$ is represented in $Aut(H) \times (Aut(K))^{p-1}$, then a nontrivial automorphism of either H or K can be computed from π in polynomial time. Similarly, if π is represented in $Z_p^* \times (Aut(K))^{p-2} \times Iso(H, K) \times Iso(K, H)$, then a nontrivial element of $Iso(H, K)$ can be computed from π in polynomial time. Furthermore,

$$(1)\quad |Aut(Cycle(H, K, \ldots, K))| = |Aut(H)| \cdot |Aut(K)|^{p-1} \cdot (1 + (p-1)\chi[H \cong K]),$$

where $\chi[H \cong K]$ is defined as 1 if H is isomorphic to K, and 0 otherwise.

Proof. Any automorphism of $Cycle(H, K, \ldots, K)$ induces an automorphism in the underlying directed p-cycle to which the graphs H and K are attached. The automorphism group of the directed p-cycle is isomorphic to $Z_p = \{0, \ldots, p-1\}$ under addition modulo p, where the element $k \in Z_p$ represents a forward rotation of the directed p-cycle by k positions.

Consider $\pi \in Aut(Cycle(H, K, \ldots, K))$. If the underlying p-cycle is pointwise fixed (represented by 0), then π maps H to itself, and also each copy of K to itself.

Thus, π can be represented as an element of the product $Aut(H) \times (Aut(K))^{p-1}$. Suppose the automorphism induced on the p-cycle is a rotation by some $k > 0$. Then π can be represented by an element of $Z_p^* \times (Aut(K))^{p-2} \times Iso(H,K) \times Iso(K,H)$, where the first component represents the k by which the underlying p-cycle is rotated by π, the second component represents the $p-2$ permutations used by π to successively map one copy of K into the next (corresponding to the rotate by k), $Iso(H,K)$ represents the mapping by π of H into the kth copy of K, and finally $Iso(K,H)$ represents the mapping of the $(p-k)$th copy of K back to H. Clearly, in this representation of $Aut(Cycle(H,K,\ldots,K))$, the part $Z_p^* \times (Aut(K))^{p-2} \times Iso(H,K) \times Iso(K,H)$ is nonempty if and only if H and K are isomorphic. It is clear from the above description that given a nontrivial $\pi \in Aut(Cycle(H,K,\ldots,K))$, if π can be represented as an element of the product $Aut(H) \times (Aut(K))^{p-1}$, then it induces a nontrivial automorphism, either on H or on some copy of K which can be computed in polynomial time from π. Similarly, if π is represented in $Z_p^* \times (Aut(K))^{p-2} \times Iso(H,K) \times Iso(K,H)$, then π induces an isomorphism from H to K which can be computed in polynomial time from π.

Since $Aut(H) \times (Aut(K))^{p-1} \cup Z_p^* \times (Aut(K))^{p-2} \times Iso(H,K) \times Iso(K,H)$ is a disjoint union and $|Iso(H,K)| = |Iso(K,H)|$ we get

$$|Aut(Cycle(H,K,\ldots,K))| = |Aut(H)| \cdot |Aut(K)|^{p-1} + (p-1) \cdot |Aut(K)|^{p-2} \cdot |Iso(H,K)|^2.$$

The expression for $|Aut(Cycle(H,K,\ldots,K))|$ in the lemma statement follows by observing additionally that if $H \cong K$, then $|Iso(H,K)| = Aut(H) = Aut(K)$, and if $H \ncong K$, then $|Iso(H,K)| = 0$. □

LEMMA 12. *For any prime p, there is a polynomial-time algorithm $\mathcal{A}$ with* mod$_p$-GA *as oracle such that given a graph G as input, $\mathcal{A}$ rejects G if $G \notin$* mod$_p$-GA*, and $\mathcal{A}$ outputs an element of order p contained in $Aut(G)$ if $G \in$* mod$_p$-GA*.*

Proof. We first give an intuitive description of $\mathcal{A}$. Let $G \in$ mod$_p$-GA. Any order-p automorphism of G is a product of disjoint p-cycles. Given G as input, algorithm $\mathcal{A}$ will compute an order-p automorphism of G by successively determining all its p-cycles. In order to find the p-cycles of an order-p automorphism we will use the *Cycle* gadget of Lemma 11 as follows. Let X be a *maximal* fixed point set for G such that $G_{[X]} \in$ mod$_p$-GA, and let $H = G_{[X]}$. For a vertex list $C = \{i_1, \ldots, i_p\}$ in a graph G, let its rotate shift $r(C)$ be the vertex list $\{i_2, \ldots, i_{p-1}, i_p, i_1\}$. By the maximality of X in the definition of H, identity (1) of Lemma 11 implies that $Cycle(H_{[C]}, H_{[r(C)]}, \ldots, H_{[r(C)]}) \in$ mod$_p$-GA, with $p-1$ copies of $H_{[r(C)]}$, if and only if $H_{[C]} \cong H_{[r(C)]}$. In turn, $H_{[C]} \cong H_{[r(C)]}$ if and only if G has an automorphism with C as a p-cycle. This property of *Cycle* is repeatedly exploited in algorithm $\mathcal{A}$ to build the desired order-p automorphism. The algorithm is formally described in Figure 2.

Correctness. If $G \notin$ mod$_p$-GA, then G is clearly rejected by $\mathcal{A}$. In order to prove correctness we must establish the following claims for any input $G \in$ mod$_p$-GA.

CLAIM 1. *The while-loop invariant: At the beginning of each iteration of the while-loop, there is an order-p automorphism of G that fixes X pointwise and contains each $C \in \mathcal{C}$ as a cycle.*

CLAIM 2. *Termination: Eventually each vertex in $S = V(G) - X$ will be covered by some cycle in $\mathcal{C}$ and the while-loop is exited.*

Clearly, if Claims 1 and 2 are true, then the algorithm will output an order-p automorphism ψ.

The first for-loop computes $G_{[X]} \in$ mod$_p$-GA where X is a maximal set (of fixed points) preserving membership of $G_{[X]}$ in mod$_p$-GA.

input G;
if $G \notin \mathrm{mod}_p$-GA **then reject** and **stop**;
$X := \emptyset$;
for $i := 1$ **to** $|V(G)|$ **do**
  if $G_{[X\cup\{i\}]} \in \mathrm{mod}_p$-GA **then** $X := X \cup \{i\}$
endfor;
$S := V(G) - X$;
$\mathcal{C} := \emptyset$;
$G' := G_{[X]}$; $G'' := G_{[X]}$;
while $S \neq \bigcup_{D\in\mathcal{C}} D$ **do**
  find a p-cycle $C \subseteq S - \bigcup_{D\in\mathcal{C}} D$ such that $Cycle(G'_{[C]}, G''_{[r(C)]}, \ldots, G''_{[r(C)]}) \in \mathrm{mod}_p$-GA;
  (There are $p-1$ copies of $G''_{[r(C)]}$ in the above Cycle definition *)*
    $G' := G'_{[C]}$; $G'' := G''_{[r(C)]}$;
    $\mathcal{C} := \mathcal{C} \cup \{C\}$;
endwhile;
output ψ and **stop**.

FIG. 2. *Algorithm* $\mathcal{A}$.

Proof of Claim 1. Claim 1 clearly holds at the beginning of the first iteration of the while-loop when $\mathcal{C}$ is empty. Now, suppose it holds at the beginning of some iteration of the while-loop for a set of p-cycles $\mathcal{C}$, and a new p-cycle C gets added in that iteration. Since the algorithm adds C to $\mathcal{C}$, we have $Cycle(G'_{[C]}, G''_{[r(C)]}, \ldots, G''_{[r(C)]}) \in \mathrm{mod}_p$-GA.

Observe that by construction of G' and G'' in $\mathcal{A}$ we have at any stage of the algorithm

$$Aut(G'_{[C]}) = Aut(G''_{[r(C)]}) = \left\{ \pi \in Aut(G) \mid \forall x \in \bigcup_{D\in\mathcal{C}} D \cup C \ : \ \pi(x) = x \right\}.$$

Hence, by identity (1) of Lemma 11 we have

$$|Aut(Cycle(G'_{[C]}, G''_{[r(C)]}, \ldots, G''_{[r(C)]}))| = |Aut(G'_{[C]})|^p \cdot (1 + (p-1)\chi[G'_{[C]} \cong G''_{[r(C)]}]).$$

From the above identity, $Aut(Cycle(G'_{[C]}, G''_{[r(C)]}, \ldots, G''_{[r(C)]})) \in \mathrm{mod}_p$-GA if and only if either $G'_{[C]} \in \mathrm{mod}_p$-GA or $G'_{[C]} \cong G''_{[r(C)]}$. Notice that $G'_{[C]} \notin \mathrm{mod}_p$-GA, since maximality of X implies that no order-p element of $Aut(G)$ can pointwise fix $X \cup C$. Thus, $G'_{[C]} \cong G''_{[r(C)]}$. Let $\psi \in Iso(G'_{[C]}, G''_{[r(C)]})$. Then ψ induces a $\phi \in Aut(G)$ which pointwise fixes X, and $\mathcal{C} \cup \{C\}$ is contained in the p-cycle set of ϕ. Let $o(\phi) = p^s m$ for positive integers m and s with $(p, m) = 1$. Notice that $s = 1$, for if $s > 1$, then $\phi^{p^{s-1}m}$ is an order-p automorphism of G that pointwise fixes not only X but each point in the cycles in $\mathcal{C}\cup\{C\}$, contradicting maximality of X. As $(p, m) = 1$, there are integers α, β such that $p\alpha + m\beta = 1$. Notice that $\phi^{m\beta}$ has order p, because $o(\phi) = pm$ and p does not divide $m\beta$. We claim that $\phi^{m\beta}$ is an order-p automorphism of G that contains $\mathcal{C} \cup \{C\}$ in its p-cycle set. To see this, let ρ_1 be the product of the p-cycles in $\mathcal{C} \cup \{C\}$. Then $\phi = \rho_1\rho_2$ for some permutation ρ_2 whose nontrivial cycles are disjoint from $\mathcal{C} \cup \{C\}$. We have $\phi^{m\beta} = \rho_1^{m\beta}\rho_2^{m\beta}$ as ρ_1 and ρ_2 commute. Since $o(\rho_1) = p$ we have $\rho_1^{m\beta} = \rho_1^{1-p\alpha} = \rho_1$. Thus, $\phi^{m\beta} = \rho_1\rho_2^{m\beta}$, implying that $\phi^{m\beta}$ is an order-p automorphism of G which contains $\mathcal{C} \cup \{C\}$ as p-cycles and also pointwise fixes X. This finishes the proof of Claim 1.

Proof of Claim 2. To prove Claim 2 it suffices to show that after every iteration of the while-loop a new p-cycle gets added to $\mathcal{C}$. Since X is a maximal fixed point set for the graph G preserving membership in mod$_p$-GA, notice that for $\psi \in Aut(G_{[X]})$ of order p, $\psi(x) \neq x\ \forall x \in V(G) - X$. That is, each vertex in $V(G) - X$ is in some p-cycle of ψ. Consider the beginning of some iteration of the while-loop such that $S \neq \bigcup_{D\in\mathcal{C}} D$. By Claim 1 there is an order-p automorphism ψ of $G_{[X]}$ containing each $D \in \mathcal{C}$ as a cycle. Let $C \subseteq S - \bigcup_{D\in\mathcal{C}} D$ be a p-cycle of ψ. Clearly, $\psi \in Iso(G'_{[C]}, G''_{[r(C)]})$, and hence $Cycle(G'_{[C]}, G''_{[r(C)]}, \ldots, G''_{[r(C)]}) \in$ mod$_p$-GA. Thus, there is at least one p-cycle C for which the "find" will succeed. So, a new p-cycle will get included in $\mathcal{C}$ at the end of this iteration. It follows that the while-loop iteration will terminate in $|V(G) - X|/p$ iterations. This completes the proof of Claim 2.

Finally, observe that $\mathcal{A}$ has polynomially bounded running time, since the while-loop iterates at most n/p times, and each "find" operation takes time $O(n^p)$ (implemented as an exhaustive search of all p-cycles contained in $S - \bigcup_{D\in\mathcal{C}} D$). □

For the general case we need two additional graph gadgets. The first of these, *Paste*, is built out of t n-vertex graphs, $G_1, \ldots, G_t$. The gadget $Paste(G_1, G_2, \ldots, G_t)$ has the property that any $\psi \in Aut(Paste(G_1, G_2, \ldots, G_t))$ induces on each G_j, $1 \leq j \leq t$ the same automorphism $\pi \in \bigcap_{j=1}^{t} Aut(G_j)$. In the next lemma we formally describe *Paste* and state relevant properties.

LEMMA 13. *Given t graphs $G_1, G_2, \ldots, G_t$, each with n nodes, we can construct in polynomial time a new graph $Paste(G_1, G_2, \ldots, G_t)$ with vertex set a disjoint union of $V(G_i)$, $1 \leq i \leq t$, such that the following properties hold.*

1. *ψ is an isomorphism from $Paste(G_1, G_2, \ldots, G_t)$ to $Paste(H_1, H_2, \ldots, H_t)$ if and only if there is a permutation $\pi \in \bigcap_{i=1}^{t} Iso(G_i, H_i)$ such that ψ restricted to G_i is π for $1 \leq i \leq t$. In fact, $\psi \leftrightarrow \pi$ is a bijective correspondence between $Iso(Paste(G_1, G_2, \ldots, G_t), Paste(H_1, H_2, \ldots, H_t))$ and $\bigcap_{i=1}^{t} Iso(G_i, H_i)$.*
2. *Given $\psi \in Iso(Paste(G_1, G_2, \ldots, G_t), Paste(H_1, H_2, \ldots, H_t))$, we can construct in polynomial time the corresponding $\pi \in \bigcap_{i=1}^{t} Iso(G_i, H_i)$.*
3. *Let p be a prime. $Paste(G_1, G_2, \ldots, G_t) \in$ mod$_p$-GA if and only if there is an order-p permutation $\pi \in \bigcap_{i=1}^{t} Aut(G_i)$.*

Proof. We define $Paste(G_1, G_2, \ldots, G_t)$ for t graphs $G_1, G_2, \ldots, G_t$, with n vertices each. It has one copy of each of the graphs $G_1, G_2, \ldots, G_t$. Furthermore, we label all the nodes of G_i using a distinct label c_i which depends only on the index i (see Figure 3 for an example). In particular, for two graphs $Paste(G_1, G_2, \ldots, G_t)$ and $Paste(H_1, H_2, \ldots, H_t)$, the nodes of both G_i and H_i get the same label c_i by construction for each i. This forces each isomorphism from $Paste(G_1, G_2, \ldots, G_t)$ to $Paste(H_1, H_2, \ldots, H_t)$ to map the copy of G_i to H_i for every i. Next, for each $i < t$ and $1 \leq j \leq n$, we put a directed edge from the jth node of G_i to the jth node of G_{i+1}. This ensures that for each isomorphism ψ from $Paste(G_1, G_2, \ldots, G_t)$ to $Paste(H_1, H_2, \ldots, H_t)$, if node j_1 in G_i is mapped to node j_2 in H_i, then for each $i' \neq i$, the corresponding node j_1 of G'_i gets mapped to the node j_2 of H'_i. It follows that there is a $\pi \in S_n$ such that for each i, ψ maps G_i to H_i via π. Thus, the construction guarantees the property claimed in part 1.

Notice that parts 2 and 3 of the lemma are both direct consequences of part 1 and the above gadget construction. This proves the lemma. □

We give some intuitive explanation before proceeding further. Recall that in the general case the input are $G \in$ mod$_{p^{i+1}}$-GA and A_i as a list of permutations, where A_i is a p^i-subgroup of $Aut(G)$. By Fact 10 we need to compute $g \in Aut(G) - A_i$ such that $o(g)$ divides p^{i+1} and $gA_ig^{-1} = A_i$. For this purpose, we will design the last

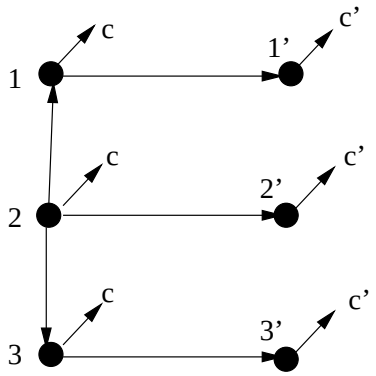


FIG. 3. *An example of Paste: The entire graph is Paste(G, G'), where G is induced by* 1, 2, *and* 3, *and G' is induced by* $1'$, $2'$, *and* $3'$. *Nodes of G and G' are labeled by c and c', respectively.*

of our graph gadgets *Combine* which is built from G and A_i using *Paste* and *Cycle*. Essentially, *Combine* has the property that allows us to easily extract the desired element g from an order-p automorphism of *Combine*. We explain the underlying ideas before giving the formal details. First, we transform the algebraic requirement $gA_ig^{-1} = A_i$ into the setting of GAs. For this we need a crucial definition and a lemma of Hoffman from [13].

DEFINITION 14 (see [13]). *Let $\pi \in S_n$ be a permutation. The* cycle graph *of π is the directed graph $G = ([n], E)$, where $(i, j) \in E$ if and only if $\pi(i) = j$ for $i \neq j$.*

LEMMA 15 (see [13]). *Let G and H be the cycle graphs of π and ψ in S_n, respectively. Then, Aut(G) is precisely the set of all permutations in S_n that commute with π. Furthermore, Iso(G, H) is precisely the set of permutations g such that $g\pi g^{-1} = \psi$.*

Let $A_i = \{g_1, g_2, \ldots, g_t\}$. Let G_i be the cycle graph of g_i for $1 \leq i \leq t$. Clearly, $gA_ig^{-1} = A_i$ if and only if $g \in \bigcap_{i=1}^{t} Iso(G_i, G_{\tau(i)})$ for some permutation τ of $[t]$. Let $H = Paste(G, G_1, G_2, \ldots, G_t)$ and $K = Paste(G, G_{\tau(1)}, G_{\tau(2)}, \ldots, G_{\tau(t)})$. The element g we are seeking is in $Iso(H, K)$ for some τ, with the additional properties that $g \notin A_i$ and $o(g)$ divides p^{i+1}. Since we must compute g with access to mod$_p$-GA as oracle (not GI), we will define *Combine* to be essentially $Cycle(H, K, \ldots, K)$ with $p - 1$ copies of K. Actually, we will incorporate in *Combine* more labels attached to H and the copies of K that allow us to identify cycles of g and also to ensure that the resulting element g is in $Aut(G) - A_i$. We now formally define *Combine* and establish some properties.

LEMMA 16. *Let G be a graph on n nodes and $S = \{g_1, g_2, \ldots, g_t\} \subseteq S_n$ be a set of permutations. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_s\} \subseteq S_n$ be a set of pairwise disjoint cycles, p be a fixed prime, and τ be a permutation on $[t]$. Then we can compute in time polynomial in n, t, and s a graph Combine$(\tau, G, S, \mathcal{C}, p)$ such that Aut(Combine$(\tau, G, S, \mathcal{C}, p)$) is in bijective correspondence with*

$$A^p \cup Z_p^* \times A^{p-2} \times B^2,$$

where

1. *A is the set of automorphisms π of G such that $\pi(x) = x$ $\forall x \in \bigcup_{i=1}^{s} C_i$ and such that $\pi g_i \pi^{-1} = g_i$ for $1 \leq i \leq t$;*
2. *B is the set of all automorphisms π of G such that $C_1, C_2, \ldots, C_s$ are cycles of π and such that $\pi g_i \pi^{-1} = g_{\tau(i)}$ for $1 \leq i \leq t$.*

Furthermore, given a nontrivial $\psi \in Aut(Combine(\tau, G, S, \mathcal{C}, p))$, we can decide in polynomial time whether it has its representation in A^p, or in $Z_p^* \times A^{p-2} \times B^2$, and also compute its representation as an element of $A^p \cup Z_p^* \times A^{p-2} \times B^2$. If ψ has its representation in A^p, then we can compute in polynomial time a nontrivial element $\psi_G \in A$, and if ψ has its representation in $Z_p^* \times A^{p-2} \times B^2$, then we can compute in polynomial time a nontrivial element $\psi_G \in B$.

Proof. Let the composition $C_1 C_2 \cdots C_s$ of the cycles in $\mathcal{C}$ be $\psi \in S_n$. Let G' be the graph obtained from G by labeling each $x \in \bigcup_{i=1}^{s} C_i$ with a distinct label n_x. Similarly, let G'' be the graph obtained from G by labeling $\psi(x) \in \bigcup_{i=1}^{s} C_i$ with label n_x for each $x \in \bigcup_{i=1}^{s} C_i$.

Let $H = Paste(G', G_1, G_2, \ldots, G_t)$, where G_i is the cycle graph of g_i for $1 \le i \le t$. Similarly, let $K = Paste(G'', G_{\tau(1)}, G_{\tau(2)}, \ldots, G_{\tau(t)})$. Finally, we put one copy of H and $p-1$ copies of K together to build the graph $Cycle(H, K, \ldots, K)$. We define $Combine(\tau, G, S, \mathcal{C}, p)$ as $Cycle(H, K, \ldots, K)$. Clearly, $Combine(\tau, G, S, \mathcal{C}, p)$ can be computed in time polynomial in n, t, and s. See Figure 4 for an example graph.

Notice that by construction we have

$$Aut(G') = Aut(G'') = \left\{ \pi \in Aut(G) \mid \pi(x) = x \quad \forall x \in \bigcup_{i=1}^{s} C_i \right\}. \tag{2}$$

By Lemma 15 we have

$$Aut(G_i) = \{\pi \in S_n \mid \pi g_i \pi^{-1} = g_i\} \text{ for } 1 \le i \le t. \tag{3}$$

By construction we have

$$Iso(G', G'') = \{\pi \in Aut(G) \mid C_1, C_2, \ldots, C_s \text{ are cycles of } \pi\}. \tag{4}$$

Again, by Lemma 15 we have

$$Iso(G_i, G_{\tau(i)}) = \{\pi \in S_n \mid \pi g_i \pi^{-1} = g_{\tau(i)}\}. \tag{5}$$

By identities 2 and 3 and the definition of A we have

$$A = Aut(G') \cap \bigcap_{i=1}^{t} Aut(G_i) = Aut(G'') \cap \bigcap_{i=1}^{t} Aut(G_{\tau(i)}). \tag{6}$$

By identities 4 and 5 and the definition of B we have

$$B = Iso(G', G'') \cap \bigcap_{i=1}^{t} Iso(G_i, G_{\tau(i)}). \tag{7}$$

We now show that $Aut(Combine(\tau, G, S, \mathcal{C}, p))$ is in bijective correspondence with $A^p \cup Z_p^* \times A^{p-2} \times B^2$. By Lemma 11, $Aut(Combine(\tau, G, S, \mathcal{C}, p))$ can be represented as

$$Aut(H) \times (Aut(K))^{p-1} \cup Z_p^* \times (Aut(K))^{p-2} \times Iso(H, K) \times Iso(K, H).$$

First, consider $Aut(H) \times (Aut(K))^{p-1}$. By Lemma 13 and identity (6), $Aut(H)$ is in bijective correspondence with $Aut(G') \cap \bigcap_{i=1}^{t} Aut(G_i) = A$ and $Aut(K)$ is in

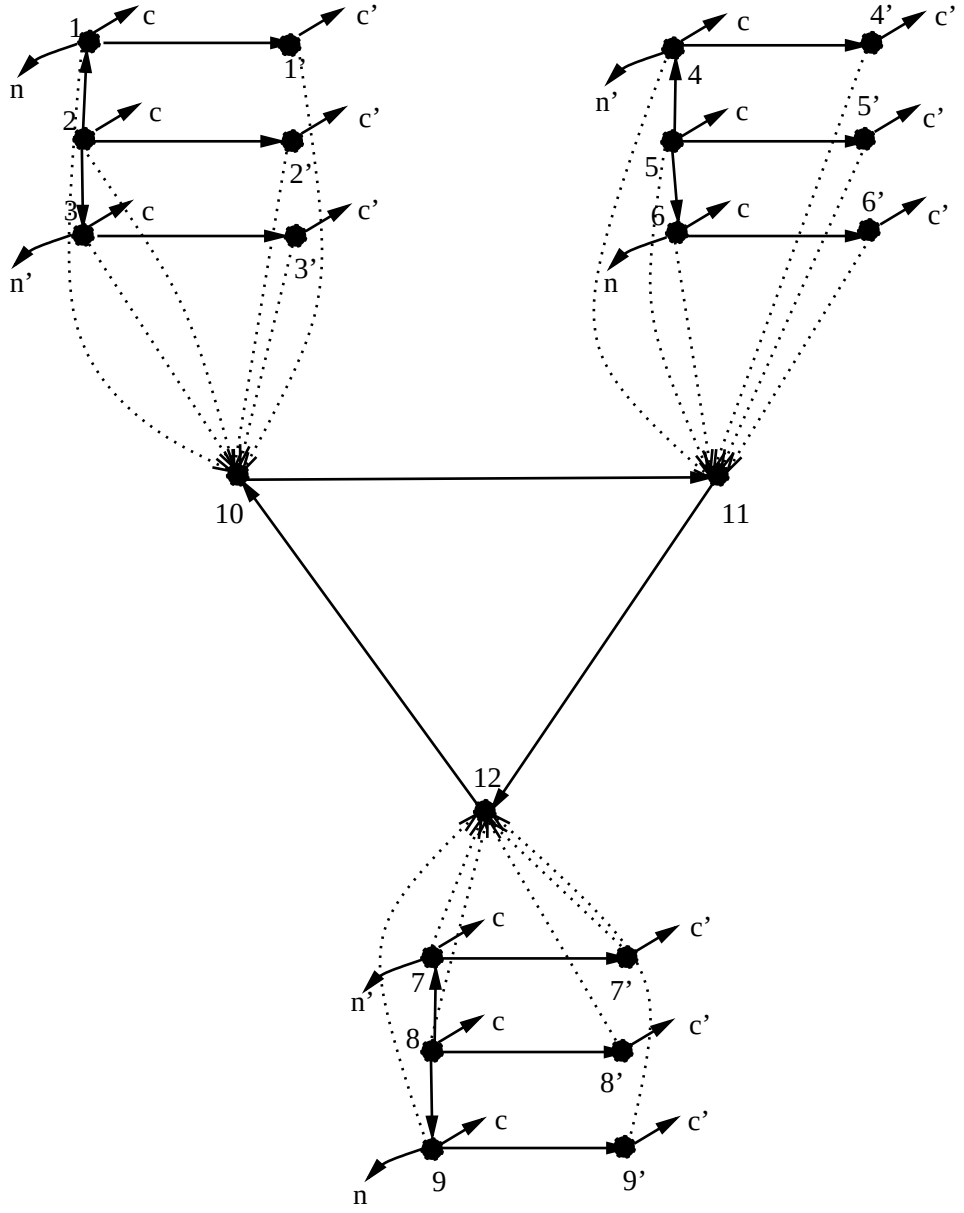


FIG. 4. *An example of Combine: The entire graph is Combine*$(\tau, G, S, \mathcal{C}, p)$*, where* $\tau = \mathrm{id}$*,* G *is the graph induced by nodes* 1*,* 2*, and* 3*,* $S = \{\mathrm{id}\}$*,* $\mathcal{C} = \{(1\ 3)\}$*, and* $p = 3$*. Notice that* G' *is the graph* G *with labels* n *and* n' *on nodes* 1 *and* 3*, respectively.* G'' *has labels* n' *and* n *on nodes* 1 *and* 3*, respectively. The graph induced by* $1'$*,* $2'$*, and* $3'$ *is the cycle graph* G_{id} *of* id*. The additional labels* c *and* c' *are for building* $H = Paste(G', G_{\mathrm{id}})$ *and* $K = Paste(G'', G_{\mathrm{id}})$*. Finally, the nodes* 10*,* 11*, and* 12 *induce the cycle underlying* $Cycle(H, K, K)$*. For clarity, some edges are shown with dashed lines.*

bijective correspondence with $Aut(G'') \cap \bigcap_{i=1}^{t} Aut(G_{\tau(i)}) = A$. Hence, $Aut(H) \times (Aut(K))^{p-1}$ is in bijective correspondence with A^p.

Next, consider $Z_p^* \times (Aut(K))^{p-2} \times Iso(H, K) \times Iso(K, H)$. Since $Aut(K)^{p-2}$ is in bijective correspondence with A^{p-2}, it suffices to show that $Iso(H, K) \times Iso(K, H)$

is in bijective correspondence with B^2. By Lemma 13, $Iso(H, K)$ is in bijective correspondence with $Iso(G', G'') \cap \bigcap_{i=1}^{t} Iso(G_i, G_{\tau(i)})$, which is B by identity (7). Since $Iso(K, H)$ is clearly in bijective correspondence with $Iso(H, K)$, it follows that $(Aut(K))^{p-2} \times Iso(H, K) \times Iso(K, H)$ is in bijective correspondence with $Z_p^* \times A^{p-2} \times B^2$.

Finally, by the properties of *Cycle* and *Paste* described in Lemmas 11 and 13, notice that given a nontrivial automorphism $\psi \in Aut(Combine(\tau, G, S, \mathcal{C}, p))$ we can decide in polynomial time whether its representation is in A^p or in $Z_p^* \times A^{p-2} \times B^2$. If the representation of ψ is in A^p, then, by Lemma 13, ψ induces a nontrivial automorphism on H or some copy of K. To be precise, we take the automorphism induced on H or the first such copy of K on which a nontrivial automorphism is induced. This automorphism of H or K yields a nontrivial automorphism $\psi_G \in A$ by identity (6) and the bijective correspondence of Lemma 13. Similarly, if ψ has its representation in $Z_p^* \times A^{p-2} \times B^2$, then, by Lemmas 11 and 13, ψ induces an isomorphism $\psi_G \in Iso(H, K) = B$. In either case, by Lemma 11, ψ_G is easily computable from ψ. □

COROLLARY 17. *$Combine(\tau, G, S, \mathcal{C}, p) \in \mathrm{mod}_p$-GA if and only if either A has an element of order p or B is not empty. Furthermore, given any order-p element $\psi \in Aut(Combine(\tau, G, S, \mathcal{C}, p))$, either $\psi_G \in A$ and ψ_G is of order p, or $\psi_G \in B$ (where A, B, and ψ_G are as defined in Lemma 16).*

Proof. Identity (1) in Lemma 11 and the bijective correspondence of Lemma 16 immediately yields

$$|Aut(Combine(\tau, G, S, \mathcal{C}, p))| = |A|^p(1 + (p-1)\chi[B \neq \emptyset]), \tag{8}$$

where $\chi[B \neq \emptyset]$ is 1 if $B \neq \emptyset$ and is 0 otherwise.

For the first part, by identity (8), notice that $Combine(\tau, G, S, \mathcal{C}, p) \in \mathrm{mod}_p$-GA if and only if p divides $|A|$ or B is not empty. Since A is a group, it has an element of order p if and only if p divides $|A|$ (by Sylow's theorem (Theorem 9)).

Next, by Lemma 16 $\psi_G \in A \cup B$. Notice that $\psi_G \in A$ precisely when ψ has its representation in A^p. In that case, ψ induces an order-p automorphism on H or some copy of K. Consequently, by identity (6) and the bijective correspondence of Lemma 13, ψ_G is an order-p element in A. □

We are ready to prove the main result of this section.

THEOREM 18. *For any prime p, there is a polynomial-time algorithm $\mathcal{B}$ with mod_p-GA as oracle such that, given a graph G and a p^i-subgroup A_i of $Aut(G)$ as input, if $G \notin \mathrm{mod}_{p^{i+1}}$-GA, then $\mathcal{B}$ rejects, and if $G \in \mathrm{mod}_{p^{i+1}}$-GA, then $\mathcal{B}$ lists out the elements of an order-p^{i+1} subgroup of $Aut(G)$.*

Proof. The case $i = 0$ is proven in Lemma 12. We first give a brief intuitive description of algorithm $\mathcal{B}$. Given G and A_i as input, the algorithm exhaustively—but in time polynomial in $|V(G)|$—searches for a permutation τ of A_i and a cycle collection $\mathcal{C}$ of constant size (specified in the algorithm) such that $Combine(\tau, G, A_i, \mathcal{C}, p) \in \mathrm{mod}_p$-GA. Now, using algorithm $\mathcal{A}$ of Lemma 12 it computes an order-p element $\nu \in Aut(Combine(\tau, G, A_i, \mathcal{C}, p))$. Next, applying Lemma 16, $\nu_G \in Aut(G)$ is extracted from ν. The element g (defined in Fact 10) is computed from ν_G, and A_{i+1} is computed from g and A_i using Fact 10. In order to ensure that g is the desired element defined in Fact 10, the cycle collection $\mathcal{C}$ is picked by the algorithm in such a way that it fulfills two properties: First, the points of $\bigcup_{D \in \mathcal{C}} D$ witness that $\nu_G \notin A_i$, as $\mathcal{C}$ is picked such that ν_G differs from each element of A_i at some point in $\bigcup_{D \in \mathcal{C}} D$; this ensures that $g \notin A_i$. Second, only cycles whose size is a power of p are picked, which guarantees that $o(g) = p^s$ for some $s > 0$.

input G and a p^i-subgroup A_i of $Aut(G)$;
$X := \{m \mid 1 \leq m \leq p^{i+1} \text{ and } (p,m) = 1\}$;
for each pairwise disjoint cycle collection
$\{\{C_s\}_{s=1}^t \mid t \leq 2p^{2i+1}, \prod_{s=1}^t C_s \neq \text{id}, \text{ and } (\prod_{s=1}^t C_s)^{p^{i+1}} = \text{id}\}$
and each sequence $\{l_{m,j} \in M_j\}_{1<j\leq p^i, m\in X}$
and each sequence $\{f_{m,j} \in F_j\}_{j\in J, m\in X}$
and each permutation τ of A_i
do
if the following conditions hold:
(a) $\{l_{m,j} \in M_j\}_{1<j\leq p^i} \subseteq \cup_{1\leq i\leq t} C_i$ for each $m \in X$
(b) $\{f_{m,j} \in F_j\}_{j\in J} \subseteq \cup_{1\leq i\leq t} C_i$ for each $m \in X$
(c) $(C_1C_2\cdots C_t)^m(l_{m,j}) \neq g_j(l_{m,j})$ **or** $(C_1C_2\cdots C_t)^m(f_{m,j}) \neq g_j(f_{m,j})$ $\forall j$ and for each $m \in X$
(d) $Combine(\tau, G, A_i, \mathcal{C}, p) \in \text{mod}_p\text{-GA}$, where $\mathcal{C} := \{C_1, \ldots, C_t\}$
then
Use Algorithm $\mathcal{A}$ of Lemma 12 to compute an order-p automorphism ν of $Combine(\tau, G, A_i, \mathcal{C}, p)$;
Using Lemma 16, compute from ν an automorphism ν_G of G;
Compute $o(\nu_G) = rp^s$, where $(p,r) = 1$;
Let $g := \nu_G^r$;
Compute A_{i+1} defined in Fact 10;
Output A_{i+1} and **stop**
endif
endfor
reject and **stop**.

FIG. 5. *Algorithm* $\mathcal{B}$.

We now present the details. Let $A_i = \{g_1, \ldots, g_{p^i}\}$, where without loss of generality, $g_1 = \text{id}$. For $1 \leq j \leq p^i$, let $F_j = \{x \in [n] : g_j(x) = x\}$ and $M_j = [n] - F_j$. That is, F_j is the set of fixpoints of $g_j \in A_i$ and M_j is the set of points that are not fixed by g_j. Let $J \subseteq [p^i]$ denote the set of indices j such that $F_j \neq \emptyset$. Notice that $J \neq \emptyset$ since $1 \in J$. The algorithm is described in Figure 5.

Correctness. In order to prove the correctness of algorithm $\mathcal{B}$, it suffices to establish the following claims.

CLAIM 1. *If* $G \in \text{mod}_{p^{i+1}}\text{-GA}$*, then for some choice of the "for" loop parameters, the four "if" conditions* (a), (b), (c), *and* (d) *will be true.*

CLAIM 2. *If the "if" conditions* (a), (b), (c), *and* (d) *are met by some choice of the for-loop parameters, then the set* A_{i+1} *that is output is a* p^{i+1}*-subgroup of* $Aut(G)$.

Proof of Claim 1. Suppose $G \in \text{mod}_{p^{i+1}}\text{-GA}$ and A_i is a p^i-subgroup of $Aut(G)$. By Fact 10, there is a $\pi \in Aut(G) - A_i$ such that $\pi A_i \pi^{-1} = A_i$ and $o(\pi) = p^s$. Fix such a π. If $(p,m) = 1$, then π^m and π both generate the same cyclic group. It follows that $\forall m \in X \;:\; \pi^m \notin A_i$. Thus, for each $m \in X$, we have sequences $\{l_{m,j} \in M_j\}_{1<j\leq p^i}$ and $\{f_{m,j} \in F_j\}_{j\in J}$ such that $\forall j$, the following holds:

$$\pi^m(l_{m,j}) \neq g_j(l_{m,j}) \text{ or } \pi^m(f_{m,j}) \neq g_j(f_{m,j}).$$

Writing π as a product of disjoint cycles we need at most $2p^i$ cycles of π to verify as above that $\pi^m \notin A_i$ for a given $m \in X$, since in the worst case we may need to pick a distinct cycle for each point $l_{m,j}$ or $f_{m,j}$. Furthermore, since $|X| \leq p^{i+1}$ we

need to pick at most $2p^{2i+1}$ cycles of π to verify that $(\forall m \in X)[\pi^m \notin A_i]$. For such a choice of cycles $\mathcal{C} = \{C_1, \ldots, C_t\}$ of π and sequences $\{l_{m,j}\}$ and $\{f_{m,j}\}$, conditions (a), (b), and (c) in the "if" statement are fulfilled.

Next, observe that since $\pi A_i \pi^{-1} = A_i$ there is a permutation τ of A_i such that $\pi g_j \pi^{-1} = g_{\tau(j)}$ for each $g_j \in A_i$. We claim that $\mathit{Combine}(\tau, G, A_i, \mathcal{C}, p) \in \mathrm{mod}_p\text{-GA}$. To see this, recall by Lemma 16 that $\mathit{Aut}(\mathit{Combine}(\tau, G, A_i, \mathcal{C}, p))$ has the representation $A^p \cup Z_p^* \times A^{p-2} \times B^2$. It is easy to verify that $\pi \in B$. Since $B \neq \emptyset$, by Corollary 17 we have $\mathit{Combine}(\tau, G, A_i, \mathcal{C}, p) \in \mathrm{mod}_p\text{-GA}$. Thus, condition (d) in the "if" statement is fulfilled by $\mathit{Combine}(\tau, G, A_i, \mathcal{C}, p)$. This completes proof of Claim 1.

Proof of Claim 2. Let ν be the order-p automorphism of $\mathit{Combine}(\tau, G, A_i, \mathcal{C}, p)$ computed by algorithm $\mathcal{A}$. Consider $\nu_G \in \mathit{Aut}(G)$, defined in Lemma 16. Let $o(\nu_G) = rp^s$, $(p, r) = 1$. Let $g = \nu_G^r$. Then $o(g) = p^s$.

By Lemma 16 and Corollary 17, either ν_G is an order-p element in A or $\nu_G \in B$. We show in each case that the algorithm outputs a p^{i+1}-subgroup A_{i+1} of $\mathit{Aut}(G)$.

- Suppose ν_G is an order-p element of A. In this case, $g = \nu_G$. Now, by definition of A, we have $\nu_G A_i \nu_G^{-1} = A_i$ and $\nu_G(x) = x \ \forall x \in \bigcup_{j=1}^t C_j$. Since $\{l_{1,j} \in M_j\}$ is contained in $\bigcup_{j=1}^t C_j$, we have $\nu_G(l_{1,j}) = l_{1,j} \ \forall j > 1$, which implies that $g = \nu_G \notin A_i$. Thus, by Fact 10, A_{i+1} is a p^{i+1}-subgroup of $\mathit{Aut}(G)$.
- Suppose $\nu_G \in B$. By Lemma 16, $\nu_G A_i \nu_G^{-1} = A_i$, and $C_1, \ldots, C_t$ are cycles of ν_G. The algorithm $\mathcal{B}$ picks $C_1, \ldots, C_t$ such that $\prod_{j=1}^t C_j \neq \mathrm{id}$ and $(\prod_{j=1}^t C_j)^{p^{i+1}} = \mathrm{id}$. Since $(\prod_{j=1}^t C_j)^{p^{i+1}} = \mathrm{id}$ and $(p, r) = 1$, we have $(\prod_{j=1}^t C_j)^r = (\prod_{j=1}^t C_j)^{m'}$ for some m' such that $1 \leq m' < p^{i+1}$ and $(p, m') = 1$. Therefore,

$$(9) \qquad \forall x \in \bigcup_{j=1}^t C_j \ : \ g(x) = \nu_G^r(x) = \left(\prod_{j=1}^t C_j\right)^r(x) = \left(\prod_{j=1}^t C_j\right)^{m'}(x).$$

  Since the "if" conditions (a), (b), and (c) are fulfilled, the sequences $\{l_{m',j} \in M_j\}$ and $\{f_{m',j} \in F_j\}$ are both contained in $\bigcup_{j=1}^t C_j$, and we also have

$$\left(\prod_{j=1}^t C_j\right)^{m'}(l_{m',j}) \neq g_j(l_{m',j}) \ \textbf{or} \ \left(\prod_{j=1}^t C_j\right)^{m'}(f_{m',j}) \neq g_j(f_{m',j}), \quad 1 \leq j \leq p^i.$$

  By identity (9) this is equivalent to

$$g(l_{m',j}) \neq g_j(l_{m',j}) \ \textbf{or} \ g(f_{m',j}) \neq g_j(f_{m',j}), \quad 1 \leq j \leq p^i.$$

  Hence, $g \notin A_i$. Finally, since $\nu_G A_i \nu_G^{-1} = A_i$, we have $g A_i g^{-1} = \nu_G^r A_i \nu_G^{-r} = A_i$. Putting it all together, it follows from Fact 10 that the output set A_{i+1} is a p^{i+1}-subgroup of $\mathit{Aut}(G)$.

This completes the proof of Claim 2.

Running time. Let $|V(G)| = n$, where G is the input graph. An easy analysis shows that the running time of algorithm $\mathcal{B}$ is polynomial in n (treating p and i as constants): Notice that each of the four nested for-loops has a polynomial (in n) range. The following computations involved are within for-loops: checks of the "if" conditions, calls to algorithm $\mathcal{A}$ of Lemma 12, invoking the procedure described in

Lemma 16, and computing powers of permutations. Each of these has polynomial running time. Hence, the overall algorithm has polynomial running time. □

Notice the following immediate consequence of Theorem 18 and Lemma 8. Interestingly, it is analogous to the well-known result that $\mathrm{Mod}_p\mathrm{P}$ and $\mathrm{Mod}_{p^k}\mathrm{P}$ are identical [5].

COROLLARY 19. *For any prime p and any $k > 0$, mod_p-GA and mod_{p^k}-GA are polynomial-time Turing equivalent.*

The main consequence of Theorem 18 is stated below. In the beginning of the section we argued how it follows easily from Theorem 18.

THEOREM 20. *Let $\prod_{1\le j\le m} p_j^{e_j}$ be the prime factorization of k. There is a polynomial-time oracle algorithm with oracle access to mod_k-GA that computes, for input $G \in \mathrm{mod}_k$-GA, a list of m subgroups $\{A_1, A_2, \ldots, A_m\}$ of $Aut(G)$, where each A_i is listed as a set of permutations and $|A_i| = p_i^{e_i}$.*

5. A program checker for mod_k-GA. As mentioned in the introduction, both GI and GA have polynomial-time program checkers. Since mod_k-GA sits between GA and GI with respect to many-one reducibility, it is natural to ask if mod_k-GA is also polynomial-time checkable. This does not follow from the results of GI and GA as polynomial-time checkability is known to be preserved only under polynomial-time Turing *equivalence* [7]. We show that mod_k-GA has a polynomial-time program checker for each $k > 1$.

DEFINITION 21 (see [7]). *A program checker C_A for a decision problem A is a probabilistic oracle algorithm that takes as input an instance x_0 of A and a positive integer k (the security parameter) given in unary such that, given any program P (purportedly for A) that halts on all instances, the following properties are satisfied:*

1. *If P is a correct program, that is, if $P(x) = A(x)$ for all instances x, then with probability $\ge 1 - 2^{-k}$, $C_A(x_0, P, k) =$ Correct.*
2. *If $P(x_0) \ne A(x_0)$, then with probability $\ge 1 - 2^{-k}$, $C_A(x_0, P, k) =$ Incorrect.*

The probability is computed over the internal coin-tosses of the checker C_A, and C_A is allowed to make oracle queries to the program P on some instances.

DEFINITION 22 (see [11]). *An interactive proof system consists of a prover-verifier pair $P \leftrightarrow V$. The verifier V is a probabilistic polynomial-time machine, and the prover P is, in general, a machine of unlimited computational power which shares the input tape and a communication tape with V. $P \leftrightarrow V$ is an interactive (i.e., IP) protocol for a language L if for every $x \in \Sigma^*$,*

$$x \in L \Rightarrow \mathrm{Prob}[P \text{ makes } V \text{ accept }] > 3/4,$$

$$x \notin L \Rightarrow \text{ for all provers } P' : \mathrm{Prob}[P' \text{ makes } V \text{ accept }] < 1/4.$$

The design of our checker for mod_k-GA is based on the following theorem.

THEOREM 23 (see [7]). *If a decision problem A and its complement have both interactive proof systems, in each of which the honest prover can be simulated in polynomial time with queries to A, then A has a polynomial-time program checker.*

We first consider mod_p-GA for prime p. Notice that Lemma 12 already gives an IP protocol for mod_p-GA with the prover polynomial-time Turing reducible to mod_p-GA. Thus, it suffices to design an IP protocol for $\overline{\mathrm{mod}_p\text{-GA}}$ with requisite properties.

LEMMA 24. *For any prime p, there is an IP protocol for $\overline{\mathrm{mod}_p\text{-GA}}$ in which the honest prover is polynomial-time Turing reducible to mod_p-GA.*

Proof. We will build the desired IP protocol from the above IP protocol for the

input (G, C);
$Y := [n] - \{i : i \in C\}$;
1. Verifier:
Pick $\psi \in S_Y$ and $b \in \{0, 1\}$, both uniformly at random;
if $b = 0$ **then send** $G' = \psi(G)$ to the Prover
else send $G' = \psi \circ C(G)$ to the Prover
endif
2. Prover:
if there is $\pi \in S_Y$ such that $\pi(G) = G'$ **then send** $c = 0$
else send $c = 1$
endif;
if $c = b$ **then** the Verifier **accepts**
else the Verifier **rejects**
endif

FIG. 6. 2-*round IP protocol for* L.

related language $L = \{(G, C) : |V(G)| = n,\ C \in S_n$ is a p-cycle and G has no automorphism with C as one of its cycles$\}$ (see Figure 6).

We first show that if the prover is honest, then the protocol accepts an input $(G, C) \in L$ with probability 1. Suppose $b = 0$ and the graph $\psi(G) = G'$ is sent to the prover. Then clearly the prover will find a permutation, namely, ψ, such that $\psi(G) = G'$ and send back $c = 0$, leading to the acceptance of the input. Next, suppose $b = 1$. In that case we claim that there does not exist any permutation $\pi \in S_Y$ such that $\pi(G) = G'$. Suppose there exists such a π. Then, since $\pi(G) = \psi \circ C(G)$, it follows that $(\pi)^{-1}\psi \circ C$ is in $Aut(G)$, which contradicts the assumption that $(G, C) \in L$. In this case the prover will send back $c = 1$ and the verifier will again accept.

Next, to prove that the protocol is sound, we must show that the verifier rejects an input $(G, C) \notin L$ with probability at least $3/4$ for any prover. In what follows we denote the set $\{i : i \in C\}$ by X and use Y to denote $[n] - X$.

CLAIM A. *If* $\tau \in Aut(G)$ *such that* τ *has* C *as a cycle, then the random graphs* $\psi(G)$ *and* $\psi \circ C(G)$ *are identically distributed, where* ψ *is picked uniformly at random from* S_Y.

Proof of Claim A. Let $\tau = \rho \circ C$, where $\rho \in S_Y$. Since S_Y is a group and $\rho \in S_Y$, if α is uniformly distributed in S_Y, then so is $\beta = \alpha \circ \rho$. Therefore, for any graph H

$$Prob_\alpha[\alpha(G) = H] = Prob_\alpha[\alpha\tau(G) = H]$$

and, replacing τ by $\rho \circ C$, we have

$$Prob_\alpha[\alpha\tau(G) = H] = Prob_\alpha[(\alpha \circ \rho) \circ C(G) = H] = Prob_\beta[\beta \circ C(G) = H],$$

where α and β are picked uniformly at random from S_Y. This completes the proof of Claim A.

It follows from Claim A that if $(G, C) \notin L$, the prover cannot distinguish between whether G' came from the case $b = 0$ or from $b = 1$. In fact, whether $b = 0$ or $b = 1$ the prover will find a $\pi \in S_Y$ such that $\pi(G) = G'$. Therefore, the bit c that is sent back by any (even a cheating) prover can agree with b with probability at most $1/2$. Consequently, the verifier will reject an input $(G, C) \notin L$ with probability at least $1/2$. The error probability can be made exponentially small (say, 2^{-n}) in the

input G; (* G has n nodes *)
for each p-cycle $C \in S_n$ **do**
    if the IP protocol for L rejects (G, C) **then reject** (and stop)
endfor;
accept

FIG. 7. *IP protocol for* $\overline{\text{mod}_p\text{-GA}}$.

above protocol by repeating the protocol[2] (in parallel or sequentially). In Figure 7 we describe the IP protocol for $\overline{\text{mod}_p\text{-GA}}$.

Notice that since p is constant, the total number of p-cycles in S_n is bounded by qn^p for a constant q. Thus the for-loop in the above protocol is bounded by a polynomial in n. It is easy to see that this IP protocol accepts $G \in \overline{\text{mod}_p\text{-GA}}$ with probability 1 and rejects $G \in \text{mod}_p\text{-GA}$ with probability larger than 3/4. The following claim completes the proof of the lemma.

CLAIM B. *There is an honest prover that is polynomial-time Turing reducible to* mod_p-GA *for the above IP protocol for* $\overline{\text{mod}_p\text{-GA}}$.

Proof of Claim B. It suffices to describe a polynomial-time algorithm with mod_p-GA as oracle that simulates the honest prover correctly for inputs $G \in \overline{\text{mod}_p\text{-GA}}$. Since the IP protocol for $\overline{\text{mod}_p\text{-GA}}$ invokes the protocol for L, it is enough to give a polynomial-time algorithm, with oracle mod_p-GA, that simulates the honest prover of the IP protocol for L for each input in the set $\{(G, C) : C \text{ is a } p\text{-cycle in } S_n\}$, where $G \in \overline{\text{mod}_p\text{-GA}}$. The honest prover's task in the protocol for L is to compute $\pi \in S_Y$ such that $\pi(G) = G'$, where $Y = [n] - C$. We have already argued in the correctness proof that for $G \in \overline{\text{mod}_p\text{-GA}}$ such a $\pi \in S_Y$ exists if and only if the outcome of b is 0 and $G' = \psi(G)$ for the random permutation $\psi \in S_Y$. Thus, we can assume that the honest prover is sent a graph G' by the verifier, where $G' = \psi(G)$ for some $\psi \in S_Y$ (ψ is not known to the prover). Since $Y \cap C = \emptyset$, it follows that $G_{[C]}$ and $G'_{[C]}$ are isomorphic (take the isomorphism that pointwise fixes C and is defined as ψ on Y). Thus, $|Aut(G_{[C]})| = |Aut(G'_{[C]})|$. Now, consider $G'' = Cycle(G_{[C]}, G'_{[C]}, \ldots, G'_{[C]})$ with $p-1$ copies of $G'_{[C]}$. By identity (1) in Lemma 11, we have

$$|Aut(G'')| = |Aut(G_{[C]})|^p \cdot (1 + (p-1)\chi[G_{[C]} \cong G'_{[C]}]).$$

It follows that $G'' \in \text{mod}_p\text{-GA}$ since $G_{[C]}$ and $G'_{[C]}$ are isomorphic.

The honest prover simply invokes algorithm $\mathcal{A}$ on input G'' (which is in mod_p-GA) and computes an order-p automorphism ν of G''.

Recall, by Lemma 11, that $Aut(G'')$ can be represented as

$$Aut(G_{[C]}) \times (Aut(G'_{[C]}))^{p-1} \cup Z_p^* \times (Aut(G'_{[C]}))^{p-2} \times Iso(G_{[C]}, G'_{[C]}) \times Iso(G'_{[C]}, G_{[C]}).$$

Since $G_{[C]} \notin \text{mod}_p\text{-GA}$ and $G'_{[C]} \notin \text{mod}_p\text{-GA}$, by Lemma 11 each order-p element $\nu \in Aut(G'')$ has its representation in $Z_p^* \times (Aut(G'_{[C]}))^{p-2} \times Iso(G_{[C]}, G'_{[C]}) \times Iso(G'_{[C]}, G_{[C]})$, and from ν an element $\pi' \in Iso(G_{[C]}, G'_{[C]})$ can be computed in polynomial time. Clearly, π' when projected on Y yields a $\pi \in S_Y$ such that $\pi(G) = G'$. The honest prover computes π as described and sends it back to the verifier.

As claimed, we have shown that there is an honest prover of the IP protocol for L that is polynomial-time Turing reducible to mod_p-GA. □

[2]With some modifications we can easily get constant round IP protocols.

THEOREM 25. *For any prime p,* mod_p-GA *has a polynomial-time program checker.*

Proof. Note that, from Lemma 12, we get an IP protocol for mod_p-GA with the prover polynomial-time Turing reducible to mod_p-GA and that, by Lemma 24, an IP protocol with requisite properties exists for $\overline{\mathrm{mod}_p\text{-GA}}$. Now, by Theorem 23 it follows that there is an efficient checker for mod_k-GA. □

THEOREM 26. *For each $k > 1$,* mod_k-GA *has a polynomial-time program checker.*

Proof. Let $\prod_{1 \le i \le m} p_i^{e_i}$ be the prime factorization of k. Because the class of checkable sets is closed under join and under Turing equivalence [7], by Theorem 25 it suffices to show that $\mathrm{mod}_k\text{-GA} \equiv_T^p \mathrm{mod}_{p_1}\text{-GA} \oplus \cdots \oplus \mathrm{mod}_{p_m}\text{-GA}$. Observe that a graph G belongs to mod_k-GA if and only if $(\forall i \le m)[G \in \mathrm{mod}_{p_i^{e_i}}\text{-GA}]$. Since, by Corollary 19, $\mathrm{mod}_{p_i^{e_i}}\text{-GA} \equiv_T^p \mathrm{mod}_{p_i}\text{-GA}$ for each i, we have $\mathrm{mod}_k\text{-GA} \le_T^p \mathrm{mod}_{p_1}\text{-GA} \oplus \cdots \oplus \mathrm{mod}_{p_m}\text{-GA}$. Recall from Lemma 8 that $\mathrm{mod}_{p_i}\text{-GA} \le_m^p \mathrm{mod}_k\text{-GA}$ for each i. Therefore, $\mathrm{mod}_{p_1}\text{-GA} \oplus \cdots \oplus \mathrm{mod}_{p_m}\text{-GA} \le_T^p \mathrm{mod}_k\text{-GA}$ as well. □

6. Concluding remarks. We define the modular GA problems (mod_k-GA) and locate them between GA and GI. We also design an efficient program checker for mod_k-GA based on an algorithm that reduces search to decision for mod_k-GA and an IP protocol for $\overline{\mathrm{mod}_k\text{-GA}}$. The bottleneck in making our checker nonadaptive is essentially the following: Can search be reduced to decision via parallel queries for mod_p-GA for prime p? Indeed, our initial motivation in studying the mod_k-GA problems was to understand the difference between GI and GA by introducing problems of intermediate difficulty. In this context, a challenging problem is whether search reduces to decision via parallel queries for GI (hence yielding nonadaptive checkers for GI). We believe that this question will be easier to answer for mod_p-GA.

Acknowledgments. We are grateful to the referees for their thoughtful comments. In particular, we are indebted to a referee for pointing out a flaw in an earlier proof of Theorem 18. His insightful suggestions, especially the use of identity (1), have helped us considerably to improve the readability of section 4.

REFERENCES

[1] M. AGRAWAL AND V. ARVIND, *A note on decision versus search for graph automorphism*, Inform. and Comput., 131 (1996), pp. 179–189.

[2] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* II, Springer-Verlag, Berlin, 1990.

[3] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* I, 2nd ed., Springer-Verlag, Berlin, 1995.

[4] J. L. BALCÁZAR, *Self-reducibility structures and solutions of NP problems*, Rev. Math, Univ. Complut. Madrid, 2 (1989), pp. 175–184.

[5] R. BEIGEL AND J. GILL, *Counting classes: Thresholds, parity, mods, and fewness*, Theoret. Comput. Sci., 103 (1992), pp. 3–23.

[6] R. BEIGEL, M. KUMMER, AND F. STEPHAN, *Approximable sets*, Inform. and Comput., 120 (1995), pp. 304–314.

[7] M. BLUM AND S. KANNAN, *Designing programs that check their work*, J. ACM, 43 (1995), pp. 269–291.

[8] M. BLUM, M.M. LUBY, AND R. RUBINFELD, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1993), pp. 73–83.

[9] A. BORODIN AND A. DEMERS, *Some Comments on Functional Self-Reducibility and the NP Hierarchy*, Technical Report 76-284, Dept. of Computer Science, Cornell University, Ithaca, NY, 1976.

[10] R. CHANG, *On the Structure of NP Computations under Boolean Operators*, Ph.D. thesis, Cornell University, Ithaca, NY, 1991.

[11] S. GOLDWASSER, S. MICALI, AND C. RACKOFF, *The knowledge complexity of interactive proof systems*, SIAM J. Comput., 18 (1989), pp. 186–208.

[12] F. Harary, *Graph Theory*, Addison–Wesley, Reading, MA, 1969.
[13] C. Hoffman, *Subcomplete generalizations of graph isomorphism*, J. Comput. System Sci., 25 (1982), pp. 332–359.
[14] M.I. Kargapolov and Ju.I. Merzljakov, *Fundamentals of the Theory of Groups*, Grad. Texts in Math. 62, Springer-Verlag, New York, 1979.
[15] J. Köbler, U. Schöning, and J. Torán, *The Graph Isomorphism Problem: Its Structural Complexity*, Birkhäuser, Boston, 1993.
[16] A. Lozano and J. Torán, *On the nonuniform complexity of the graph isomorphism problem*, in Proceedings of the 7th Structure in Complexity Theory Conference, IEEE Press, Boston, MA, 1992, pp. 118–129.
[17] R. Mathon, *A note on the graph isomorphism counting problem*, Inform. Process. Lett., 8 (1979), pp. 131–132.
[18] J.J. Rotman, *An Introduction to the Theory of Groups*, 4th ed., Grad. Texts in Math., Springer-Verlag, New York, 1995.
[19] C.-P. Schnorr, *On self-transformable combinatorial problems*, Math. Programming Stud., 14 (1981), pp. 225–243.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1321–1340

APPROXIMATING SHORTEST PATHS ON A NONCONVEX POLYHEDRON*

KASTURI R. VARADARAJAN† AND PANKAJ K. AGARWAL‡

Abstract. We present an approximation algorithm that, given the boundary P of a simple, nonconvex polyhedron in $\mathbb{R}^3$ and two points s and t on P, constructs a path on P between s and t whose length is at most $7(1+\varepsilon)d_P(s,t)$, where $d_P(s,t)$ is the length of the shortest path between s and t on P, and $\varepsilon > 0$ is an arbitrarily small positive constant. The algorithm runs in $O(n^{5/3}\log^{5/3} n)$ time, where n is the number of vertices in P. We also present a slightly faster algorithm that runs in $O(n^{8/5}\log^{8/5} n)$ time and returns a path whose length is at most $15(1+\varepsilon)d_P(s,t)$.

Key words. approximation algorithms, Euclidean shortest paths, computational geometry

AMS subject classification. 68E99

PII. S0097539799352759

1. Introduction. Problem statement. Let P be the boundary (surface) of a simple, possibly nonconvex polyhedron in $\mathbb{R}^3$ with n vertices, and let s and t be two points on P. (A polyhedron is *simple* if it is homeomorphic to a ball in $\mathbb{R}^3$.) Let $\pi_P(s,t)$ denote any shortest path between s and t on P, and $d_P(s,t)$ denote its length. Let us call a path between s and t on P λ-*approximate*, for $\lambda \geq 1$, if its length is at most $\lambda d_P(s,t)$. In this paper, we consider the problem of computing an *approximate shortest path* on P from s to t. See Figure 1.

Computing a shortest path on a polyhedral surface is a central problem in numerous areas, including robotics, geographic information systems, medical imaging, low-altitude flight simulation, and water flow analysis. In most of these applications, a simple, efficient algorithm for computing an approximate shortest path is preferable to an expensive algorithm that computes an exact shortest path, since the input surfaces are rather large, efficiency is critical, and the polyhedral surface is typically an approximation of the real surface.

Previous results. Motivated by these applications, many researchers have studied the problem of computing a shortest path on a polyhedral surface [2, 3, 6, 10, 11, 12, 13, 17, 22, 25, 27]. Sharir and Schorr [27] gave an $O(n^3 \log n)$ algorithm for computing a shortest path on the boundary of a convex polyhedron, exploiting the property that a shortest path "unfolds" into a straight line. Mitchell, Mount, and Papadimitriou [22] improved the running time to $O(n^2 \log n)$; their algorithm works for nonconvex polyhedra as well. Chen and Han [6] gave another algorithm with an improved running time of $O(n^2)$. Although several heuristics have been proposed (see [2, 10, 11, 13, 17, 25], for example), it is a long-standing and intriguing open problem whether a subquadratic algorithm can be developed even for computing an

*Received by the editors March 3, 1999; accepted for publication (in revised form) May 18, 2000; published electronically October 31, 2000. Work on this paper has been supported by National Science Foundation grant CCR-93–01259, Army Research Office MURI grant DAAH04-96-1-0013, a Sloan fellowship, an NYI award, matching funds from Xerox Corporation, and a grant from the U.S.–Israeli Binational Science Foundation. A preliminary version of this paper appeared in *Proceedings of the 38th IEEE Symposium on the Foundations of Computer Science*, 1997, pp. 182–191.
http://www.siam.org/journals/sicomp/30-4/35275.html

†Department of Computer Science, University of Iowa, 14 MacLean Hall, Iowa City, IA 52242-1419 (kvaradar@cs.uiowa.edu). Research was performed while this author was at Duke University.

‡Department of Computer Science, Box 90129, Duke University, Durham, NC (pankaj@cs.duke.edu).

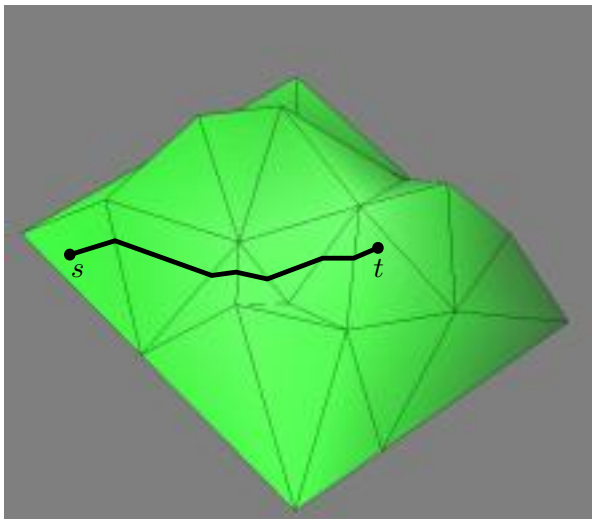


FIG. 1. *A path on a polyhedral surface.*

approximate shortest path. Hershberger and Suri [16] presented a simple algorithm that computes a 2-approximate path on a *convex* polyhedron in $O(n)$ time. Agarwal et al. [1] presented an algorithm that computes a $(1+\varepsilon)$-approximate path on a convex polyhedron in $O(n \log 1/\varepsilon + 1/\varepsilon^3)$ time for any given $\varepsilon > 0$. However, both these algorithms heavily exploit the convexity of the polyhedron and do not extend to nonconvex polyhedra.

For the three-dimensional Euclidean shortest path problem—where we are given a set of polyhedral obstacles and we want to compute the shortest collision-free path between two given points—there are some approximation algorithms due to Papadimitrou [24], Clarkson [8], and Choi, Sellen, and Yap [7]. All these algorithms compute a $(1+\varepsilon)$-approximate path, but their running times are superquadratic in n. Recently, Lanthier, Maheshwari, and Sack [19] and Mata and Mitchell [21] had considered the problem of computing the *weighted* shortest path problem on polyhedral surfaces. In this problem, each face of the given polyhedral surface has a weight associated with it, and the cost of traversing a face is the distance traveled on the face times the weight of the face. In this scenario, they give approximation algorithms for computing a minimum-cost path between two given points on the polyhedral surface. Their algorithms compute a $(1+\varepsilon)$-approximate path for the unweighted case in superquadratic time.

Our results. In this paper, we present an algorithm that runs in $O(n^{5/3} \log^{5/3} n)$ time and computes a $7(1+\varepsilon)$-approximate path on P from s to t. (Throughout the rest of this paper, we will assume that $\varepsilon > 0$ is an arbitrarily small, positive number.) We also present a second algorithm that computes in $O(n^{8/5} \log^{8/5} n)$ time a $15(1+\varepsilon)$-approximate path between s and t on P. Although our approach falls short of giving a simple, near-linear-time algorithm for this problem, it is an important step because the problem of computing an approximate shortest path in subquadratic time has been open for quite some time.

The basic idea of our algorithm is rather simple: we choose a parameter r and partition P into $O(n/r)$ "patches," each consisting of at most r faces. For each patch R_i, we carefully choose a set of points on the "boundary edges" of R_i and construct

a graph G_i that approximates a shortest path between any two points lying on the boundary edges of R_i. We merge these graphs G_i into a single graph G; s and t are guaranteed to be vertices of G. We then compute a shortest path from s to t in G using Dijkstra's algorithm and prove that the length of this path is at most $7(1+\varepsilon)d_P(s,t)$. There are two nontrivial steps in the algorithm: how to choose points on the boundary edges and how to construct the graphs G_i's. Using a scheme based on the planar separator theorem [20] to partition P into patches, we ensure that there are only $O(\sqrt{r})$ boundary edges per patch, which allows us to put only a small number of points on the boundary of each patch R_i. In order to compute the graph G_i efficiently, we need the idea of computing shortest paths on R_i from *edge sources*.

The paper is organized as follows. In section 2, we give basic definitions and properties related to shortest paths, and in section 3, we describe how to partition P into patches. Section 4 describes the overall algorithm. Sections 5 and 6 describe how to choose points on the edges of each patch and how to construct the graphs G_i's, respectively. Section 7 describes how the algorithm of Mitchell, Mount, and Papadimitriou [22] can be generalized to compute shortest paths from edge sources; the generalized algorithm is used in the construction of the G_i's. Although this is a rather technical extension of the algorithm of Mitchell, Mount, and Papadimitriou [22], we include it for completeness. We conclude in section 8 by mentioning some exciting recent developments.

2. Geometric preliminaries. We assume that the polyhedral surface P is specified by a set of faces, edges, and vertices that form a simplicial complex, with each edge occurring in exactly two faces, and two faces intersecting either at a common edge, vertex, or not at all. We consider faces to be closed polygons and edges to be closed line segments. Without loss of generality, we assume that all faces are triangles (otherwise, we can use any polygon triangulation algorithm to triangulate the faces, thus introducing at most $O(n)$ additional edges), and that s and t are vertices of P. Since P is the boundary of a polyhedron, we will sometimes refer to a path on P as a path on the polyhedron. If π is a path on P, we let $|\pi|$ denote its length. For any two points $a, b \in \pi$, let $\pi[a,b]$ denote the portion of π between a and b. For any two points u, v in P, let $\pi_P(u,v)$ denote the shortest path on P between u and v. We refer to $d_P(u,v) = |\pi_P(u,v)|$ as the *shortest path distance* on P between u and v. We denote by $d(p,q)$ the Euclidean distance between points p and q in $\mathbb{R}^3$.

We review some relevant properties of shortest paths on polyhedral surfaces. A detailed analysis can be found in [22, 27].

Unfolding. Two faces f and f' of P are said to be *edge-adjacent* if they share a common edge. A *sequence of edge-adjacent faces* is a list of one or more faces $\mathcal{F} = (f_1, f_2, \ldots, f_{k+1})$ such that the face f_i is edge-adjacent to the face f_{i+1} for $1 \le i \le k$. Let e_i be the edge shared by the faces f_i and f_{i+1}. We then refer to the (possibly empty) list of edges $\mathcal{E} = (e_1, e_2, \ldots, e_k)$ as an *edge sequence*. If no face (resp., edge) appears more than once in $\mathcal{F}$ (resp., $\mathcal{E}$), then we call the sequence *simple*.

We associate with each face a two-dimensional coordinate system. If faces f and f' are edge-adjacent sharing an edge e, we define the *planar unfolding of f' onto f* as the image of points of f' when f' is rotated about the line through e until f and f' become coplanar and lie on *opposite* sides of e; see Figure 2. Points in the planar unfolding of f' onto f are represented in the coordinate system of f. We *unfold an edge sequence* $\mathcal{E}$ as follows: Rotate f_{k+1} around e_k until its plane coincides with f_k, rotate f_{k+1} (already unfolded around e_k) and f_k around e_{k-1} until their plane

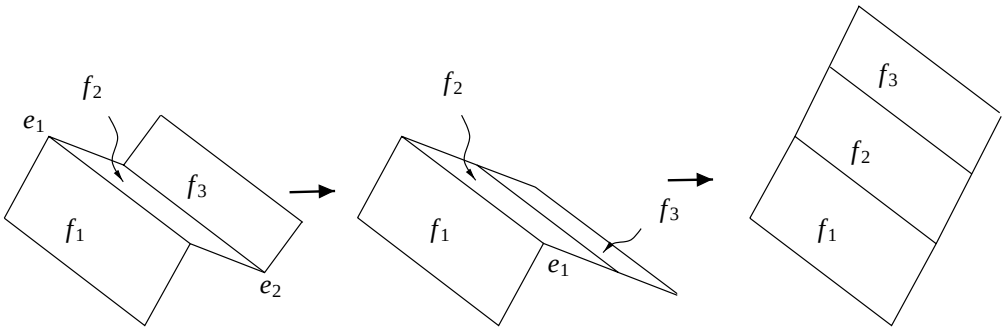


FIG. 2. *Unfolding an edge sequence.*

coincides with that of f_{k-2}, and continue in this manner until all faces $f_2, \ldots, f_{k+1}$ lie in the plane of f_1. If x is a point on face f_i, then the *unfolded image of x along $\mathcal{E}$* (written in the coordinate system of face f_1) is the image of x when we unfold the edge sequence $\mathcal{E}$.

Let v_i be a point that lies in the interior of edge e_i for $1 \leq i \leq k$. Let v (resp., v') be a point on face f_1 (resp., f_{k+1}) so that one of the following three conditions hold. (There are nine possibilities allowed here, three each for v and v'.)

(i) v (resp., v') is the vertex of f_1 (resp., f_{k+1}) that is opposite edge e_1 (resp., e_k).
(ii) v (resp., v') lies in the interior of f_1 (resp., f_{k+1}).
(iii) v (resp., v') coincides with v_1 (resp., v_k).

Then we say that the path π given by the concatenation of the segments $vv_1, v_1v_2, \ldots, v_{k-1}v_k, v_kv'$ *connects the edge sequence* $\mathcal{E}$. If π connects edge sequence $\mathcal{E}$, then the *planar unfolding of π along $\mathcal{E}$* is simply the unfolded image of π along $\mathcal{E}$.

Shortest paths. A *geodesic* is a path that is locally optimal; that is, it cannot be shortened by slight perturbations. A shortest path is obviously a geodesic. We now state a few relevant properties of geodesics and shortest paths.

LEMMA 2.1. *The intersection of a shortest path with a face is a (possibly empty) line segment. If π is a geodesic path that connects (two points along) the edge sequence $\mathcal{E}$, then the planar unfolding of π along $\mathcal{E}$ is a straight line segment.*

Unlike the case of convex polyhedra, where a geodesic cannot pass through a vertex of the polyhedron, a geodesic on the boundary of a nonconvex polyhedron can pass through a vertex. The subpath of a geodesic π between any two vertices v and v' of P that are consecutive on π connects some edge sequence $\mathcal{E}$. (If $\mathcal{E} = \emptyset$, then v and v' are the endpoints of some edge e which is contained in π.) By Lemma 2.1, the subpath from v to v' is completely determined by giving the edge sequence $\mathcal{E}$. Thus, we have the following characterization of geodesics and shortest paths.

LEMMA 2.2. *We can describe a geodesic path π from s to t by writing it as a list*

$$\Phi(\pi) = (v_1 = s, \mathcal{E}_1, v_2, \mathcal{E}_2, v_3, \ldots, v_k, \mathcal{E}_k, v_{k+1} = t),$$

where $v_1, \ldots, v_k$ are the vertices (in order) through which π passes, and each $\mathcal{E}_i$ is the (possibly empty) edge sequence which $\pi[v_i, v_{i+1}]$ connects. The planar unfolding of $\pi[v_i, v_{i+1}]$ along $\mathcal{E}_i$ is a straight line segment. If π is a shortest path, no edge of P appears in more than one edge sequence $\mathcal{E}_i$, and each edge sequence $\mathcal{E}_i$ is simple.

3. Partitioning the polyhedral surface. We can associate with the polyhedral surface P a graph G_P, the dual graph of the 1-skeleton of P, defined as follows. (We use "nodes" and "arcs" in the context of graphs and use "vertices" and "edges" when speaking of polyhedral features.) For each face f of P, there is a node n_f in G_P. There is an arc between two nodes n_f and $n_{f'}$ in G_P if the corresponding faces f and f' share an edge. Since P is a simple polyhedron, its 1-skeleton is a planar graph; therefore, G_P is also planar. Since each face of P is a triangle, the degree of each node of G_P is 3.

Our algorithm exploits a scheme that was developed by Frederickson [14] for partitioning any planar graph $G = (V, E)$. Let $(V_1, V_2, \ldots, V_k)$ be a cover of the node set V, that is, $V_i \subseteq V$ and $\bigcup_i V_i = V$. We refer to each V_i as a *region*. A node in V is *interior* to a region if all its adjacent nodes are in that region; a *shared* node is one that is present in at least two regions. For a given parameter r, an *r-partition* of G is a covering of the node set V by $\Theta(|V|/r)$ regions, so that the following two conditions hold (see Figure 3).

(i) Any node is either a shared node or an interior node of some region; and

(ii) each region contains at most r nodes and $O(\sqrt{r})$ shared nodes.

Based on the separator theorem of Lipton and Tarjan [20], Frederickson [14] showed that r-partitions exist and can be constructed in $O(|V| \log |V|)$ time.

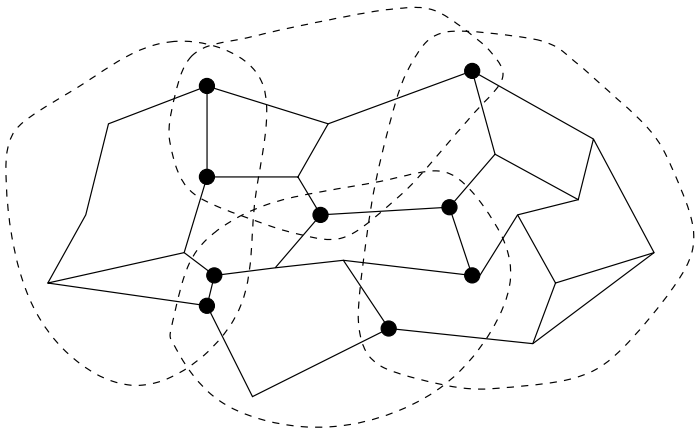


FIG. 3. *An r-partition of a planar graph. The shared nodes are highlighted.*

An r-partition of G_P induces a subdivision of the polyhedral surface P, which we describe below. A *polyhedral patch* of P (or simply, a patch of P) is just the portion of P comprised of a subset of the faces of P. Given an r-partition of G_P, we designate any face that corresponds to a shared node of G_P as a *buffer face*. For each region V_i, we associate a polyhedral patch P_i consisting of the faces corresponding to the interior nodes of V_i. (That is, each shared node defines one buffer face, and each region defines one polyhedral patch.) Thus, an r-partition of G_P induces a partition P into $O(n/\sqrt{r})$ buffer faces and $O(n/r)$ polyhedral patches, with each patch containing at most r faces. Abusing terminology slightly, we refer to this partition of P as an *r-partition* of P. An r-partition of P can be computed in $O(n \log n)$ time.

An edge of P incident to a buffer face is called a *frontier* edge. It is easily seen that an r-partition satisfies the following properties:

1. If a face f in a patch shares an edge e with another face f', then either f' belongs to the same patch as f, or f' is a buffer face. In the latter case, e is

a frontier edge.

2. Each patch has only $O(\sqrt{r})$ frontier edges.

Note that the frontier edges of a polyhedral patch constitute its boundary, so we can also refer to them as the *boundary* edges of the patch. The total number of frontier edges is at most $cn/\sqrt{r}$ for some constant $c > 0$. Let $\alpha = cn/\sqrt{r}$. The following lemma describes the main application of the above partitioning scheme, but first we need a definition.

DEFINITION 3.1. *An ε-portal set on an edge e of P is a collection $\phi(e)$ of points (or portals) on e so that the following holds: For any point x on e that lies on the shortest path $\pi_P(s,t)$, there exists $p \in \phi(e)$ whose distance from x is at most $(\varepsilon/2\alpha) \times d_P(s,t)$. For simplicity, we assume that $\phi(e)$ includes the endpoints of edge e. Given an r-partition of P and an ε-portal set on each frontier edge of the r-partition, a path is legal if it lies completely within a single patch (resp., buffer face), and each of its two endpoints is either s, t, or a portal on a frontier edge of the patch (resp., buffer face).*

LEMMA 3.2. *Given an r-partition of the polyhedron P and an ε-portal set on each frontier edge of the r-partition, there is a path π^* on P between s and t such that (1) its length is at most $(1+\varepsilon)d_P(s,t)$, and (2) it is a concatenation of legal paths.*

Proof. Let $\pi = (v_1 = s, \mathcal{E}_1, v_2, \mathcal{E}_2, v_3, \ldots, v_k, \mathcal{E}_k, v_{k+1} = t)$ represent the shortest path between s and t, where $v_1, \ldots, v_k$ are the vertices (in order) through which π passes, and each $\mathcal{E}_i$ is the (possibly empty) edge sequence which $\pi[v_i, v_{i+1}]$ connects. Let e be a frontier edge that occurs in some edge sequence $\mathcal{E}_i$ above. Let x be the (unique) point on e that lies on π, and let p be the portal on e so that $d_P(x,p) = d(x,p) \le \varepsilon/2 \times d_P(s,t)/\alpha$. We alter π slightly by introducing a loop from x to p and back. This increases the length of the path by at most $\varepsilon d_P(s,t)/\alpha$. If we perform this surgery on every frontier edge that occurs in some edge sequence, we get a path π^* whose length is at most $(1+\varepsilon)d_P(s,t)$. It is clear that π^* is a concatenation of legal paths, since s and t are assumed to be vertices of P. ∎

4. The overall algorithm. In this section, we present our algorithm for computing a path between s and t on P whose length is at most $7(1+\varepsilon)d_P(s,t)$. For the sake of clarity, we first present an algorithm that approximates the shortest path distance, i.e., it returns a quantity that is at least $d_P(s,t)$ and at most $7(1+\varepsilon)d_P(s,t)$. Subsequently, we modify this algorithm to compute an approximate shortest path.

4.1. Approximating the shortest path distance. In order to describe the algorithm, we need to define a β-approximator, which is similar to the well-studied notion of a spanner (see [5, 26]) with Steiner points.

DEFINITION 4.1. *Let R be a polyhedral patch and S a set of points on R. A β-approximator for S on R is a weighted graph $G = (S \cup S', E)$, where S' is a set of additional points on R, that has the following properties.*

(P1) *The weight $w(u,v)$ of any arc $(u,v) \in E$ is $d_R(u,v)$, the length of the shortest path between u and v on R.*

(P2) *For any $u, v \in S$, the length of the shortest path in G between u and v, denoted $d_G(u,v)$, is at most $\beta d_R(u,v)$.*

ALGORITHM. APPROXIMATE-DISTANCE.

1. We compute an r-partition of P, where $r = n^{1/3}\log^{1/3} n$. Let $R_1, \ldots, R_{k_1}$ denote the resulting buffer faces, $R_{k_1+1}, \ldots, R_{k_1+k_2}$ denote the resulting patches, and B denote the set of resulting frontier edges. Here, $k_1 = O(n/\sqrt{r})$, $k_2 = O(n/r)$, and $|B| \le \alpha = O(n/\sqrt{r})$. Recall that the boundary of each

buffer face consists of three frontier edges, and the boundary of each patch R_i consists of at most $O(\sqrt{r})$ frontier edges.

2. On each frontier edge $e \in B$, we compute an $\varepsilon/2$-portal set.
3. Let V denote the set of vertices of P, and let M denote the set of all the portals introduced in step 2. For each $i \leq k_1 + k_2$, let $V_i = V \cap R_i$ be the set of vertices in R_i, and let $M_i = M \cap R_i$ be the set of portals on the frontier edges of R_i. Set $|M_i| = m_i$ and $\beta = 7(1+\varepsilon/3)$. We compute a β-approximator $G_i = (M_i \cup M_i', E_i)$ for M_i on R_i, where M_i' is a set of additional Steiner points on R_i.
4. We compute the graph $G = (N, E)$, where $N = \bigcup_i (M_i \cup M_i')$ and $E = \bigcup_i E_i$. Using Dijkstra's algorithm [9], we compute the shortest path $\pi_G(s,t)$ between s and t in G, and return its length $d_G(s,t)$ as our estimate of $d_P(s,t)$.

A critical component of the algorithm is the procedure for computing the β-approximators in step 3, which we describe in section 6. The procedure for computing the portal sets in step 2, which we outline in section 5, is comparitively straightforward.

THEOREM 4.2. (i) $d_P(s,t) \leq d_G(s,t) \leq 7(1+\varepsilon)d_G(s,t)$. (ii) *The algorithm runs in* $O(n^{5/3}\log^{5/3} n)$ *time.*

Proof. Assuming the correctness of the various subprocedures, it follows from Lemma 3.2 that the algorithm returns an estimate $d_G(s,t)$ such that $d_P(s,t) \leq d_G(s,t) \leq 7(1+\varepsilon)d_P(s,t)$, provided ε is chosen sufficiently small.

We now analyze the running time of the algorithm. In step 1 of the algorithm, we can compute an r-partition in $O(n \log n)$ time. The number of frontier edges in the r-partition is $\alpha = O(n/\sqrt{r})$.

Using the algorithm described in the next section (see Lemma 5.2), we can compute an $\varepsilon/2$-portal set of size $O((n/\sqrt{r})\log n)$ on a frontier edge in $O((n/\sqrt{r})\log n)$ time. Summing up over all frontier edges, step 2 takes $O((n^2/r)\log n)$ time. Moreover, $M = O((n^2/r)\log n)$.

To analyze step 3, we consider the buffer faces and patches separately. For each buffer face R_i, $1 \leq i \leq k_1$, we compute a β-approximator G_i using any algorithm for computing spanners in the plane. Using Clarkson's algorithm [8], for instance, we obtain a graph $G_i = (M_i, E_i)$ (without Steiner points), with $|E_i| = O(m_i/\beta)$, in $O(m_i \log m_i)$ time. A buffer face R_i has at most three frontier edges incident on it, so $m_i = O((n/\sqrt{r})\log n)$. Hence, the running time for computing G_i using Clarkson's algorithm is $O((n/\sqrt{r})\log^2 n)$. The time taken in step 3 over all buffer faces is thus $O((n^2/r)\log^2 n)$.

For a patch R_i, $k_1+1 \leq i \leq k_1+k_2$, we use the algorithm described in section 6.1 to construct a $7(1+\varepsilon)$-approximator for M_i on R_i. The algorithm introduces a set M_i' of at most $O(m_i)$ Steiner points and constructs a graph $G_i = (M_i \cup M_i', E_i)$ with $O(r^2 + m_i)$ arcs. Furthermore, the algorithm takes $O(r^3 \log r + m_i \log r + m_i \log m_i)$ time to construct G_i. Since the boundary of R_i consists of $O(\sqrt{r})$ frontier edges and each edge has $O((n/\sqrt{r})\log n)$ portals, $m_i = O(n \log n)$. The running time for computing G_i is then $O(r^3 \log r + n \log^2 n)$, and the running time for step 3 summed over all patches is $O(nr^2 \log r + (n^2/r)\log^2 n)$.

To compute the time taken in step 4, we first estimate the size of the graph G. Each buffer face contributes $O(n/\sqrt{r})$ nodes and arcs to G, so the contribution to G from all buffer faces is $O(n^2/r)$ nodes and arcs. Each patch contributes $O(r + n\log n)$ nodes and $O(r^2 + n \log n)$ arcs, so the overall contribution to G from all patches is $O((n^2/r)\log n)$ nodes and $O(nr + (n^2/r)\log n)$ arcs. Running Dijkstra's algorithm on

G takes $O(nr + (n^2/r)\log^2 n)$ time [9].

Summing up, the overall running time of the algorithm is $O(nr^2 \log r + (n^2/r)\log^2 n)$. Substituting $r = n^{1/3}\log^{1/3} n$, the running time of the algorithm is $O(n^{5/3}\log^{5/3} n)$. □

The running time of the above algorithm is constrained by the procedure used in step 3 for computing the β-approximators. By using the slightly faster procedure that is described in section 6.3 for computing 15-approximators and choosing $r = n^{2/5}\log^{2/5} n$ in step 1, we obtain the following result.

THEOREM 4.3. *Given the boundary P of a simple, possibly nonconvex polyhedron in $\mathbb{R}^3$ with n vertices, and two points $s, t \in P$, we can compute in $O(n^{8/5}\log^{8/5} n)$ time a quantity between $d_P(s,t)$ and $15(1+\varepsilon)d_P(s,t)$.*

4.2. Computing an approximate shortest path. We now describe the modifications to the algorithm presented above that will allow us to compute an approximate shortest path from s to t. Let G_i be the β-approximator, computed in step 3, for M_i on R_i. We want to be able to answer the following *path query* for an arc in G_i: given $(p, p') \in E_i$, compute a path on R_i between p and p' whose length is at most the weight of (p, p'). If R_i is a buffer face, we can answer such a query by simply taking the line segment joining p and p'. For a patch R_i, we augment G_i with a data structure that can answer a path query in $O(r)$ time. It will be evident in section 6 that such a data structure can be computed within the time bound for computing G_i.

Let π_G be the shortest path in G that is computed by step 4 of the algorithm. We can compute a path π in P between s and t by simply doing a path query for each arc traversed by π_G and concatenating the resulting paths. But this is expensive if π_G traverses too many arcs of G. However, as described below, we really need to do path queries for only a few arcs in π_G. For a face f of P, if p_1 and p_2 are the first and last nodes in f that appear in π_G, we can bypass the arcs in π_G between p_1 and p_2 by including in π the line segment joining p_1 and p_2. If we perform this step for every face, we will be left with only $O(n)$ arcs of π_G for which we have to perform path queries; performing these takes a total of $O(nr)$ time. Thus we get the main result of this paper.

THEOREM 4.4. *Given the boundary P of a simple, possibly nonconvex polyhedron with n vertices, and points $s, t \in P$, we can compute in $O(n^{5/3}\log^{5/3} n)$ (resp., $O(n^{8/5}\log^{8/5} n)$) time a path on P between s and t whose length is at most $7(1+\varepsilon)d_P(s,t)$ (resp., $15(1+\varepsilon)d_P(s,t)$).*

5. Computing portal sets. In this section, we present an algorithm that, given an edge e of P and $\varepsilon > 0$, computes an ε-portal set $\phi(e)$ on e. We first describe a scheme for computing a rough estimate ρ^* of the shortest path distance $d_P(s,t)$ such that

$$\rho^* \le d_P(s,t) \le O(n)\rho^*.$$

The estimate ρ^* will be useful in constructing portal sets. Let $C(s,\omega)$ be the axis-parallel cube with edge-length 2ω centered at s, and let $P(\omega)$ be the portion of P lying within $C(s,\omega)$, i.e., $P(\omega) = P \cap C(s,\omega)$. Note that $P = P(\infty)$. For a face f of P, let $f(\omega) = f \cap C(s,\omega)$. Since f is a triangle, $f(\omega)$ is convex. This implies that $P(\omega)$ consists of $O(n)$ faces. Let ρ^* be the smallest value of ω for which s and t are connected by a path lying completely in $P(\omega)$.

LEMMA 5.1. (i) *$\rho^* \le d_P(s,t) \le c_1 n\rho^*$ for some constant c_1, and* (ii) *ρ^* can be computed in $O(n\log n)$ time.*

Proof. (i) We claim that any shortest path $\pi_P(s,t)$ must intersect the boundary of $C(s,\rho^*)$. Suppose, on the contrary, that the claim is false. Then there exists a $\rho' < \rho^*$ such that $\pi_P(s,t)$ lies completely within $C(s,\rho')$. Thus, $\pi_P(s,t)$ connects s and t in $P(\rho')$, contradicting the fact that ρ^* is the smallest value of ω for which s and t are path-connected within $P(\omega)$. Since $\pi_P(s,t)$ intersects the boundary of $C(s,\rho^*)$, we have $d_P(s,t) = |\pi_P(s,t)| \geq \rho^*$.

To establish the second inequality, consider the shortest path π' between s and t in $P(\rho^*)$. Since $P(\rho^*)$ consists of at most $O(n)$ faces and π' intersects each of these faces in a single (possibly empty) line segment, π' can be specified as a concatenation of at most $O(n)$ such line segments. Each such line segment lies completely within $C(s,\rho^*)$ and so has length at most $2\sqrt{3}\rho^*$. It follows that $|\pi'| \leq c_1 n\rho^*$ for some constant c_1. Since π' is also a path on P, $d_P(s,t) \leq |\pi'| \leq c_1 n\rho^*$.

(ii) For any ω, $P(\omega)$ is a disjoint union of path-connected components. For each face f of P, $f(\omega)$ is contained in at most one such component. For a path-connected component Ψ of $P(\omega)$, let F_Ψ be the collection f of faces of F such that $f(\omega) \neq \emptyset$ and $f(\omega)$ is a face of Ψ. Note that $\Psi = \bigcup_{f \in F_\Psi} f(\omega)$. Observe that the sets F_Ψ partition the set of faces of P that have a nonempty intersection with $C(s,\omega)$.

Let f^s (resp., f^t) be a face of P containing s (resp., t). Then ρ^* is the smallest value of ω for which (i) $s,t \in C(s,\omega)$, and (ii) $f^s(\omega)$ and $f^t(\omega)$ belong to the same path-connected component of $P(\omega)$. Thus, computing ρ^* reduces essentially to finding the smallest ω when (ii) occurs. We sketch below an algorithm to do this.

Our algorithm represents each connected component Ψ of $P(\omega)$ implicitly by the set F_Ψ. As ω increases from zero, our representation changes. Two sets F_{Ψ_i} and F_{Ψ_j} may get merged. A new face f may be added to one of the already existing sets F_Ψ. A set consisting of just a single new face f may be created if $f(\omega)$ forms a single path-connected component in $P(\omega)$. Clearly, such changes in our representation occur only when some vertex, edge, or face of P first intersects $C(s,\omega)$. Hence, there are only $O(n)$ critical values of ω at which our representation needs to be updated.

We first compute and sort (in increasing order) these critical values of ω. Starting from $\omega = 0$, we go through them in order, updating our representation of the path-connected components of $P(\omega)$ appropriately. We stop and report the value of ω when f^s and f^t are in the same set in our representation. If we use any data structure for disjoint sets [9] to represent the path-connected components, the entire procedure can be implemented in $O(n \log n)$ time. □

Using Lemma 5.1, we can show the following.

LEMMA 5.2. *Let B be the set of frontier edges in an r-partition of P, where $|B| \leq \alpha$. Given any edge $e \in B$, we can compute an ε-portal set $\phi(e)$ of size $O((\alpha/\varepsilon)\log n)$ in time $O((\alpha/\varepsilon)\log n)$.*

Proof. We use the estimate ρ^* to compute $\phi(e)$ as follows. For $1 \leq i \leq \lceil \log c_1 n \rceil$, where c_1 is the constant defined in Lemma 5.1, we add a set S_i of $O(\alpha/\varepsilon)$ portals so that they subdivide $e \cap C(s,\rho^* 2^i)$ into equal-sized intervals, each of length $(\varepsilon/2\alpha)\rho^* 2^i$.

Clearly, this procedure adds a total of $O((\alpha/\varepsilon)\lg n)$ portals. To see that $\phi(e)$ constructed above is an ε-portal set, let x be any point on $\pi_P(s,t) \cap e$. Since $d_P(s,t) \leq c_1 n\rho^*$, $x \in C(s,\rho^* 2^i)$ for some $1 \leq i \leq \lceil \log c_1 n \rceil$. Let i_x be the smallest such i. If $i_x = 1$, then there is a portal $p \in S_1$ so that

$$d(p,x) \leq (\varepsilon/2\alpha)\rho^* \leq (\varepsilon/2\alpha) d_P(s,t).$$

If $i_x > 1$, then x does not lie in $C(s,\rho^* 2^{i_x - 1})$, so $d(s,x) \geq \rho^* 2^{i_x - 1}$. There is a portal

$p \in S_{i_x}$ so that

$$d(p,x) \le (\varepsilon/2\alpha)2^{i_x-1}\rho^* \le (\varepsilon/2\alpha)d(s,x) \le (\varepsilon/2\alpha)d_P(s,t).$$

Hence there is always a portal p on e such that $d(p,x) \le (\varepsilon/2\alpha)d_P(s,t)$, as required. □

6. Approximating legal paths. Let R be a polyhedral patch with r (triangular) faces, V the set of its vertices, and B the set of its boundary edges. We assume that $|B| = O(\sqrt{r})$. Let M be a set of m points (portals) lying on the edges in B. In this section, we describe algorithms for computing a β-approximator (introduced in Definition 4.1) on R for M, the set of portals on R. That is, we construct a weighted graph $G = (M \cup M', E)$, where M' is a set of additional points on R, that has the following properties:

1. The weight $w(u,v)$ of any arc $(u,v) \in E$ is $d_R(u,v)$, the length of the shortest path between u and v on R.
2. For any $u, v \in V^B \cup M$, the length of the shortest path in G between u and v, denoted $d_G(u,v)$, is at most $\beta d_R(u,v)$.

The key to computing an approximator is the notion of shortest paths from edge sources, which we introduce below. In order to present the main ideas clearly, we first describe an algorithm for computing a 13-approximator for $V^B \cup M$ on R. We then show how this algorithm can be modified so that it returns a $7(1+\varepsilon)$-approximator. Finally, we present a faster algorithm for computing a 15-approximator.

Mount [23] showed that the single-source shortest path algorithm of Mitchell, Mount, and Papadimitriou [22] can be generalized to obtain the following result.

LEMMA 6.1. *Given a polyhedral patch R and a subset $U \subseteq V$ of "vertex sites," we can preprocess it in $O(r^2 \log r)$ time, where r is the number of faces in R, so that given a query point x lying on any edge e of R, we can compute the vertex $u \in U$ that is closest to x, and the distance $d_R(q,x)$ in $O(\log r)$ time. We can also compute in $O(r)$ time a shortest path between q and x.*

Let Q be the region of the patch R given by a union of a subset of vertices of R ("vertex sites") and a subset of boundary edges of R ("edge sites"). We will regard Q as a set of points (which is infinite if there is an edge site). We define the distance $d_R(Q,x)$ of a point $x \in R$ from Q to be the minimum $\min_{q \in Q} d_R(q,x)$. In section 7, we show that we can preprocess the patch R in $O(r^2 \log r)$ time so that we can quickly compute the shortest distance between Q and a query point lying on the edge of the patch. This result, stated in Lemma 7.4, is obtained by fairly straightforward modifications to the algorithms of Mitchell, Mount, and Papadimitriou [22] and Mount [23]. (Note that Lemma 6.1 is a special case of Lemma 7.4 where there are no edge sites.) As we shall see below, allowing edge sites is a useful idea in constructing β-approximators.

6.1. Computing a 13-approximator. We now present our algorithm for computing a 13-approximator $G = (M \cup M', E)$ for the portals M on the polyhedral patch R.

ALGORITHM. 13-APPROXIMATOR.

1. For each pair $u, v \in V$ of vertices in the path R, we compute the shortest path distance $d_R(u,v)$ on R between u and v and introduce an arc $(u,v) \in E$ whose weight is $d_R(u,v)$. We can easily compute these distances in $O(r^3 \log r)$ time by invoking the algorithm of Mitchell, Mount, and Papadimitriou [22] for each vertex of R.

2. We run the preprocessing algorithm of Lemma 6.1 with $U = V$. For each $p \in M$, we find the vertex $v \in V$ that is closest to p. We compute the distance $d_R(v, p)$ and introduce an arc (p, v) in G with weight $d_R(v, p)$.
3. We repeat the following for every boundary edge $e \in B$: We let Q be the set of points on R consisting of the union of all the vertices V and all the boundary edges except edge e and run the preprocessing algorithm of Lemma 7.4. For each $p \in M$ that lies in e, we find the $q \in Q$ that is closest to p. We introduce in G a new node q, and an arc (p, q) with weight $d_R(p, q)$.
4. Let M' be the set of points that are introduced as nodes in G in the above steps. For each edge $e \in B$, we sort the points $(M \cup M') \cap e$ along e and add an arc (p, p') of weight $d(p, p')$ if p and p' are consecutive in the sorted order.

LEMMA 6.2. *The algorithm constructs in $O(r^3 \log r + m \log r + m \log m)$ time a graph with $O(r + m)$ nodes and $O(r^2 + m)$ arcs.*

Proof. Step 1 of the algorithm requires $O(r^3 \log r)$ time. In this step, we introduce $O(r^2)$ arcs. In step 2, the preprocessing algorithm of Lemma 6.1 takes $O(r^2 \log r)$ time. The computation involving a single $p \in M$ takes $O(\log r)$ time, so the overall computation for every $p \in M$ takes $O(m \log r)$ time. Running the preprocessing algorithm of Lemma 7.4 once takes $O(r^2 \log r)$ time. Since we run this algorithm $O(\sqrt{r})$ times in step 3, once for each boundary edge, we spend $O(r^{5/2} \log r)$ time. We can perform the computation for each $p \in M$ in a total of $O(m \log r)$ time. We introduce at most m new nodes and arcs in this step. After these steps, we have a total of $O(r + m)$ nodes in G. So we can implement step 4 in $O(r + m \log m)$ time. We introduce only $O(r + m)$ arcs in this step.

Summing up, we spent $O(r^3 \log r + m \log m + m \log r)$ time and constructed a graph G with $O(r + m)$ nodes and $O(r^2 + m)$ arcs. □

The following lemma plays a crucial role in proving that the above algorithm computes a 13-approximator.

LEMMA 6.3. *Let e be a boundary edge of the patch R, and let Q be the set of points of R consisting of all the vertices of R and all the boundary edges of R except e. Let $e' \in B$ be a boundary edge of R distinct from e, and let $x \in e$ and $y \in e'$. Let $\pi(x, y)$ be a shortest path, of length l, between x and y. If $d_R(x, v) > 3l$ for every vertex $v \in V$, then the closest point q in Q to x lies on edge e', and $d_R(Q, x) \leq l$.*

Figure 4 illustrates the situation in Lemma 6.3. We first establish a preliminary claim before proving Lemma 6.3. We refer to any path that originates from x and that has length at most l as a *short* path. Since $d_R(x, v) > 3l$ for every $v \in V$, no short path passes through a vertex. It follows that every short, geodesic path connects some edge sequence beginning with the edge e. Let $\mathcal{E} = e\mathcal{E}_0 e'$ be the edge sequence that $\pi(x, y)$ connects (e and e' are, respectively, the first and last edges of $\mathcal{E}$).

CLAIM 6.4. *The edge sequence connected by any short, geodesic path is a prefix of $\mathcal{E}$.*

Proof. Suppose, on the contrary, that the claim is false. That is, there is a short, geodesic path π' that connects $\mathcal{E}' e_1 e_3$, whereas the edge sequence connected by $\pi(x, y)$ is $\mathcal{E} = \mathcal{E}' e_1 e_2 \mathcal{E}''$.

Since both paths unfold into straight lines, the edges e_1, e_2, and e_3 must be incident on the same (triangular) face, say, f. Let v_i be the vertex of f opposite e_i. Let $a_1 \in \pi(x, y) \cap e_1$, $a_2 \in \pi(x, y) \cap e_2$, and $a_3 \in \pi' \cap e_3$. Then

$$d(a_i, a_j) \leq d_R(a_i, x) + d_R(x, a_j) \leq 2l.$$

In triangle f, suppose that the angle at vertex v_1 is at least as large as the angles at v_2 and v_3. (The other cases are treated identically.) Using the fact that the angle

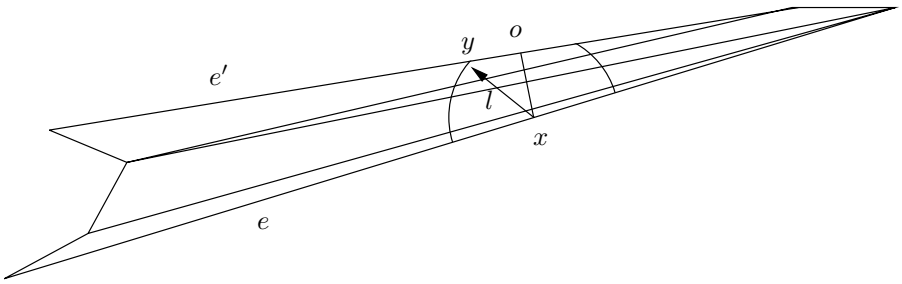


FIG. 4. *Illustration of Lemma* 6.3.

at v_1 is at least $\pi/3$,

$$\min\{d(v_1, a_2), d(v_1, a_3)\} \leq d(a_2, a_3) \leq 2l.$$

It follows from the triangle inequality that $d_R(x, v_1) \leq 3l$. This contradicts the assumption that $d_R(x, v) > 3l$ for every $v \in V$. □

We can now prove Lemma 6.3.

Proof of Lemma 6.3. Since $\pi(x, y)$ is a shortest path, the edge sequence $\mathcal{E} = e\mathcal{E}_0 e'$ is simple. Since $\pi(x, y)$ is geodesic, it unfolds into a straight line, and so there can be no boundary edge in $\mathcal{E}_0$. As a consequence of the claim, e' is the only edge site in Q that can be reached by a short, geodesic path. We have already established that no short path can reach a vertex site. Thus, the closest point to x in Q lies in edge e', and $d_R(e', x) \leq l$. □

LEMMA 6.5. *The graph G constructed by the algorithm is a* 13*-approximator for $V^B \cup M$ on the polyhedral patch R.*

Proof. Clearly, $d_G(p, p') = d_R(p, p')$ for any two vertices $p, p' \in V$ (step 1) or for any two points $p, p' \in (M \cup M')$ lying on the same boundary edge $e \in B$ (step 4).

Let π be the shortest path on R between $p, p' \in V^B \cup M$, and let $|\pi| = l$. We will show that $d_G(p, p') \leq 13l$. Let $w, w' \in V$ be the vertices in R that are closest (along R) to p and p', respectively. We consider two cases.

Case 1. $\max\{d_R(p, w), d_R(p', w')\} \leq 3l$. In step 2 of the algorithm, we introduce an arc (p, w) with weight $d_R(p, w)$ in G, so $d_G(p, w) = d_R(p, w) \leq 3l$. Similarly, $d_G(p', w') = d_R(p', w') \leq 3l$. Now,

$$d_R(w, w') \leq d_R(w, p) + d_R(p, p') + d_R(p', w') \leq 7l.$$

Since $w, w' \in V$, $d_G(w, w')$ is also at most $7l$. Finally, we have

$$d_G(p, p') \leq d_G(p, w) + d_G(w, w') + d_G(w', p') \leq 13l.$$

Case 2. $\max\{d_R(p, w), d_R(p', w')\} > 3l$. Assume, without loss of generality, that $d_R(p, w) > 3l$. In this case, neither p nor p' is a vertex. Let e (resp., e') be the boundary edge containing p (resp., p') in its interior. If e and e' are the same edge, the lemma follows because $d_G(p, p') = d_R(p, p')$.

Let us assume that e and e' are distinct. Let Q be the region consisting of all the vertices and boundary edges of the patch except edge e. Then, by Lemma 6.3, the closest point $q \in Q$ to p lies on edge e', and $d_R(q,p) \leq l$. When we processed p in step 3 of the algorithm, we had to introduce point $q \in e'$ as a node of G, and we also had to add an arc (q,p) with weight $d_R(q,p)$. Thus, $d_G(q,p) = d_R(q,p) \leq l$. By the triangle inequality, $d_R(q,p') \leq d_R(q,p) + d_R(p,p') \leq 2l$.

Since q and p' both lie on e', $d_G(q,p') \leq 2l$. Finally, we have $d_G(p,p') \leq d_G(p,q) + d_G(q,p') \leq 3l$. □

6.2. Computing a $7(1+\varepsilon)$-approximator. Using a scheme based on recent work of Har-Peled [15] for computing approximate shortest path maps on a polyhedron, we modify algorithm 13-APPROXIMATOR, as described below, so that it returns a $7(1+\varepsilon)$-approximator (for any $\varepsilon > 0$) for M on the polyhedral patch R. For a given source point v on R and an edge e, Har-Peled's algorithm computes a set $W(v,e)$ of $O((1/\varepsilon)\log 1/\varepsilon)$ points on e so that for any $q \in e$, there is a $p \in W(v,e)$ with

$$d_R(v,p) + d(p,q) \leq (1+\varepsilon) d_R(v,q).$$

The modification involves replacing steps 1 and 2 of algorithm 13-APPROXIMATOR by the following procedure, which is repeated for each $v \in V$. We compute the sets $W(v,e)$ for each edge $e \in B$; using Har-Peled's algorithm, we can do this in $O(r^2 \log r)$ time. We add each point $p \in \bigcup_{e \in B} W(v,e)$ to the node set M' of G and also include an arc (v,p) in G with weight $d_R(v,p)$. We can compute all the distances $d_R(v,p)$ in $O(r^2 \log r)$ time using the algorithm of Mitchell, Mount, and Papadimitriou [22].

It is easy to see that the running time of the new algorithm is the same as that of algorithm 13-APPROXIMATOR. To show that the graph G it computes is a $7(1+\varepsilon)$-approximator, we analyze Case 1 of Lemma 6.5 as follows. In this case, there is a vertex w of R such that $d_R(p,w) \leq 3l$. The path on R obtained by concatenating the shortest paths from p to w, w to p, and p to p' has length at most $7l$. The modifications ensure that there is a path in G between p and p' that uses w and whose length is at most $7(1+\varepsilon)l$.

6.3. Computing a 15-approximator. We now present a slightly different algorithm that computes a 15-approximator $G = (M \cup M', E)$ for M on the patch R. This algorithm, which we call 15-APPROXIMATOR, runs in $O(r^{5/2} \log r + m \log r + r \log r)$ time. Algorithm 15-APPROXIMATOR is identical to 13-APPROXIMATOR, except that we replace step 1 by the following procedure. From each $e \in B$, we let Q consist of just the edge site e and run the $O(r^2 \log r)$-time preprocessing algorithm of Lemma 7.4. For each vertex $v \in V$, let $o \in e$ be the closest point in e to v. We introduce o as a new node in G and add an arc (o,v) with weight $d_R(o,v)$. We can show the following lemma.

LEMMA 6.6. *We can compute, in time $O(r^{5/2} \log r + m \log m + m \log r)$, a 15-approximator for M on the polyhedral patch R.*

Proof. The analysis of the algorithm's running time is quite straightforward. We argue that it returns a 15-approximator. Let π be the shortest path on R between $p, p' \in V^B \cup M$ and let $|\pi| = l$. We modify the analysis of Case 1 of Lemma 6.5 as follows to show that $d_G(p,p') \leq 15l$; the analysis of Case 2 is unchanged.

In Case 1, $d_R(p,w) \leq 3l$. In step 2 of the algorithm, we introduce an arc (p,w) with weight $d_R(p,w)$ in G, so $d_G(p,w) = d_R(p,w) \leq 3l$.

Now $d_R(p',w) \leq d_R(p',p) + d_R(p,w) \leq 4l$. Let $e \in B$ be the boundary edge containing p' and let $o \in e$ be the closest point in e to w. Clearly, $d_R(w,o) \leq d_R(w,p')$. Also from the triangle inequality, $d_R(o,p') \leq d_R(o,w) + d_R(w,p') \leq 2d_R(w,p')$.

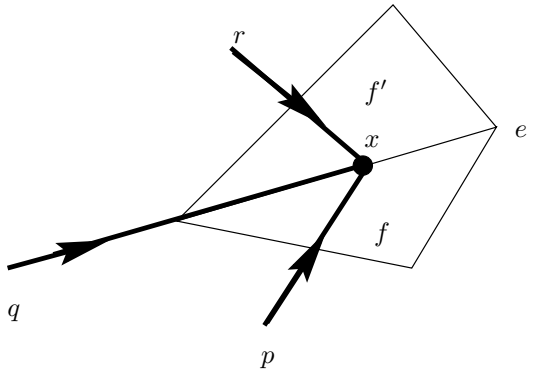


FIG. 5. *Three geodesic paths to x on edge e: p is monotone with respect to (e, f) but not with respect to (e, f'); q is monotone with respect to both (e, f) and (e, f'); r is monotone with respect to (e, f') but not with respect to (e, f).*

The new algorithm introduces in step 1 an arc (w, o) with weight $d_R(w, o)$ in G, so $d_G(w, o) = d_R(w, o)$. Since p' and o are on the same edge, $d_G(o, p') = d_R(o, p')$. Thus

$$d_G(w, p') \le d_G(w, o) + d_G(o, p') = d_R(w, o) + d_R(o, p') \le 3d_R(w, p') \le 12l.$$

Finally, $d_G(p, p') \le d_G(p, w) + d_G(w, p') \le 15l$. □

7. Shortest path computation. Let R be a polyhedral patch with r triangular faces, and let V, E, and B denote, respectively, the set of vertices, edges, and boundary edges of the patch. Let $F \subseteq E$ be a collection of boundary edges and $U \subseteq V$ be a collection of vertices. Let Q denote the portion of the patch consisting of the edges in F and the vertices in U. Since the edges in E include their endpoints, Q is a closed set. For any point x on the patch, the closest point to x in Q is any $y \in Q$ such that $d_R(y, x) = \min_{q \in Q} d_R(q, x)$. In this section, we describe a variant of the algorithms of Mount [23] and Mitchell, Mount, and Papadimitriou [22] that preprocesses R in $O(r^2 \log r)$ time into a data structure so that for any query point x lying on a boundary edge, the closest point q in Q to x and the distance $d_R(q, x)$ can be computed in $O(\log r)$ time. If desired, the shortest path between q and x can also be returned in an additional $O(r)$ time.

Given a face f, and e, one of the three edges incident to it, we refer to (e, f) as an *edge-face pair*. Every nonboundary edge is part of two edge-face pairs, while every boundary edge is part of one edge-face pair. With respect to an edge-face pair (e, f), a path π to $x \in e$ from any $q \in Q$ is *monotone* if the following hold.

1. Any subpath π' of π that does not pass through a vertex unfolds into a straight line. This means that we can represent π as a list

$$\Phi(\pi) = (v_1 = q, \mathcal{E}_1, v_2, \mathcal{E}_2, v_3, \ldots, v_k, \mathcal{E}_k, v_{k+1} = x),$$

   where $v_2, \ldots, v_k$ are the vertices (in order) through which π passes, and each $\mathcal{E}_i$ is the (possibly empty) edge sequence which $\pi[v_i, v_{i+1}]$ connects.
2. The path π reaches x through face f. This means that if x lies in the interior of edge e, the path π lies in face f just before it reaches x. (See Figure 5.)

The second condition is assumed to hold trivially if x is one of the endpoints of e. If e is a nonboundary edge incident also to face f' and x lies in the interior of e,

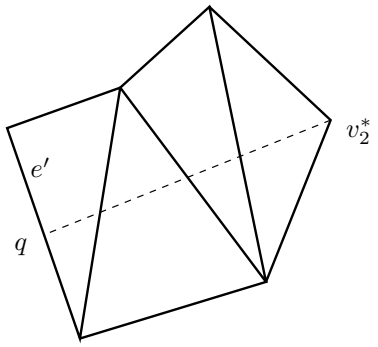


FIG. 6. *In Proposition* 7.1*, the segment* qv_2^* *is perpendicular to* e'.

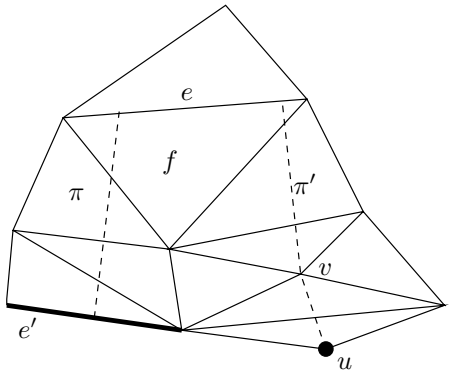


FIG. 7. *The anchor of* π *is the boundary edge-site* e'*; the anchor of* π' *is the vertex* v.

the second condition says that the path π reaches x through f and not through f'. The *shortest monotone path* from Q to a point $x \in e$ (with respect to (e, f)) is the smallest-length monotone path (with respect to (e, f)) from any $q \in Q$ to x. It is easy to see that the shortest path from Q to x is monotone with respect to either (e, f) or (e, f'). If e is a boundary edge that is not part of Q, a shortest path to x is the same as a shortest monotone path to x with respect to (e, f).

Let $\Phi(\pi) = (v_1 = q, \mathcal{E}_1, v_2, \ldots, v_k, \mathcal{E}_k, v_{k+1} = x)$ denote any monotone path π with respect to the edge-face pair (e, f) from $q \in Q$ to $x \in e$. Suppose q lies in the interior of a boundary edge $e' \in F$. Let v_2^* denote the unfolded image of v_2 when the edge sequence $\mathcal{E}_1$ is unfolded; note that the segment qv_2^* is the unfolded image of the subpath $\pi[q, v_2]$. The following proposition states a local optimality criterion for π to be a shortest monotone path. (See Figure 6.)

PROPOSITION 7.1. *Let* q*,* e'*, and* v_2^* *be the same as above. If* π *is the shortest montone path (with respect to* (e, f)*) from* Q *to* x *, then the segment* qv_2^* *is perpendicular to the edge* e'.

In the list $\Phi(\pi)$, if v_k is a vertex, we call it the *anchor* of π and $\mathcal{E}_k$ the *last edge sequence*. If v_k is not a vertex, the list is of the form $(q, \mathcal{E}, x)$, where q is a point lying in the interior of a boundary site e'. In this case, we call the edge e' the anchor of π and $\mathcal{E}$ the last edge sequence of π. See Figure 7. We now state some important properties of monotone paths.

The following lemma is analogous to Lemma 4.4 of Mitchell, Mount, and

Papadimitriou [22].

LEMMA 7.2. *For edge-face pair (e, f), edge sequence $\mathcal{E}$, and anchor r, the set I of points x on e for which there is a shortest monotone path from Q with r as the anchor and $\mathcal{E}$ as the last edge sequence is connected (and, therefore, a subsegment of e).*

We refer to I as the *interval of optimality* for r and $\mathcal{E}$ with respect to (e, f). Let $\overline{r}$ denote the unfolded image of r when $\mathcal{E}$ is unfolded onto the plane containing f. For any $x \in I$, let $\pi(x)$ denote the shortest monotone path to x (which has anchor r and last edge sequence $\mathcal{E}$). We define the *distance of x to r along $\mathcal{E}$* to be the Euclidean distance of x to $\overline{r}$. Note that this distance is the same as the length of the subpath of $\pi(x)$ from r to x. The following lemma can be proved using techniques similar to Lemmas 4.5 and 7.1 of Mitchell, Mount, and Papadimitriou [22] and Lemma 3.2 of Mount [23].

LEMMA 7.3. *Intervals of optimality with respect to edge-face pair (e, f) cover the edge e and have mutually disjoint interiors. Moreover, there are only $O(r)$ such intervals.*

7.1. The algorithm. We now describe an algorithm that computes the intervals of optimality with respect to each edge-face pair (e, f) of the polyhedral patch. If I is an interval of optimality with anchor r and last edge sequence $\mathcal{E}$, the algorithm also computes $\overline{r}$, the unfolded image of r when $\mathcal{E}$ is unfolded onto the plane containing f.

The algorithm maintains the following information. We keep a list of *candidate* intervals of optimality on each edge-face pair. A candidate interval (or interval for short) is a segment on an edge-face pair (e, f) which is a supersegment of some (possibly empty) interval of optimality. A candidate interval I has associated with it the anchor r_I and the last edge sequence $\mathcal{E}_I$ along which monotone paths from Q reach the points in I. We also store with each candidate interval I its unfolded anchor $\overline{r}_I$. We also store with each interval its *frontier point* α_I, which is the point in I that mimimizes the distance to r_I along $\mathcal{E}_I$ (the point that is closest to the unfolded anchor $\overline{r}_I$). If the anchor r_I is one of the edge sites, we define the *priority* of the frontier point α_I to be its distance to r_I along $\mathcal{E}_I$. If the anchor is a vertex, the algorithm will have already computed a value $d(r_I)$ (which is the shortest path distance to r_I). In this case, we define the priority of α_I to be $d(r_I)$ plus the distance between α_I and r_I along $\mathcal{E}_I$.

The algorithm also maintains a priority queue called the *event queue*. The event queue contains the frontier point α_I of each candidate interval I, with its priority, if α_I has not already been "permanently labeled."

We are now ready to describe the overall algorithm. It makes use of two procedures, `project` and `insert-interval`, which we describe after the overall algorithm. In the initialization phase, we perform the following steps.

1. On each boundary edge e in Q, we insert a candidate interval I extending over the entire edge. Since e has only one face f incident on it, we can think of I as being on the edge-face pair $(e, \emptyset)$, where $\emptyset$ is a "dummy" face. The anchor of I is the edge e itself, and its last edge sequence is empty. Note that the priority of this interval is zero.
2. For each vertex v in Q, we set $d(v) = 0$. For each face f incident to v and each edge e of f, we insert a candidate interval I, whose extent is the entire edge e, on the edge-face pair (e, f); the anchor of I is v and its last edge sequence is empty; we refer to v as the *predecessor* $\mathrm{pred}(I)$ of I. (By predecessor of an interval I, we mean the candidate interval or vertex whose "propagation" gives rise to I.) We then call the procedure `insert-interval`(I, e).

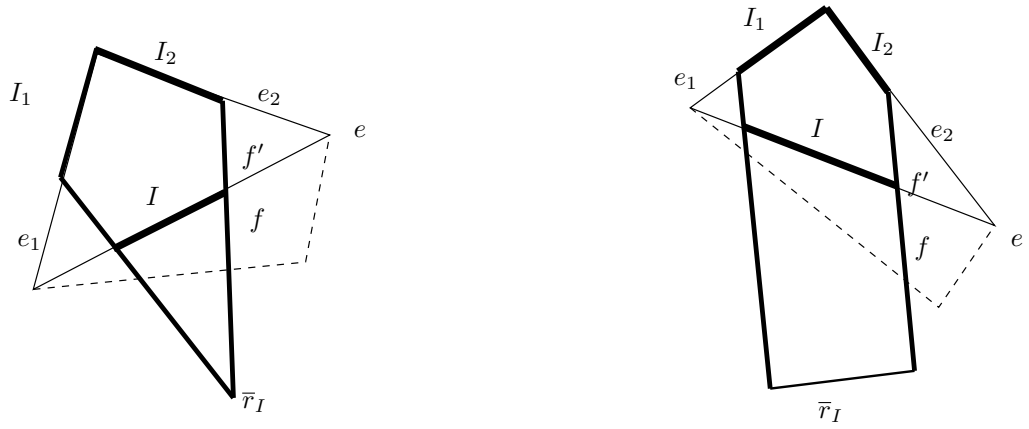


FIG. 8. *Projecting an interval I:* (i) *r_I is a vertex;* (ii) *r_I is an edge site.*

The main loop of the algorithm consists of iterating the following steps until the event queue becomes empty.

1. We remove from the event queue the frontier point α_I that has the smallest priority. Suppose that α_I belongs to candidate interval I on edge-face pair (e, f). We permanently label α_I with its priority.
2. If α_I is a vertex v, and a value has not already been assigned to $d(v)$, we assign $d(v)$ to be the priority of $v = \alpha_I$. We store a pointer $ptr(v)$ from v to the candidate interval I. We look at each face f' incident to v, and each edge e' on f'. We insert a candidate interval I on edge-face pair (e', f') whose extent is the entire edge e'. The anchor of I is v, and its last edge sequence is empty. We set v to be the predecessor $\text{pred}(I)$ of I. We then call the procedure `insert-interval`(I, e). If $d(v)$ has already been assigned to vertex v, we can skip the above steps.
3. Whether α_I is a vertex or not, we find the other face f' incident to e. If there is no such face, we do nothing. Let e_1 and e_2 be the two other edges of f'. We call `project`(I, e_1) to find the interval I_1 on e_1 that is hit by monotone paths through I. We set I to be the predecessor of $\text{pred}(I_1)$ of I_1. We then call the procedure `insert-interval`(I_1, e_1). We do the same with e_2.

We now describe the two procedures `project` and `insert-interval`. Let e be an edge incident to two faces f and f', and let e_1 and e_2 be the other two edges of f'. Suppose I is a candidate interval on the edge-face pair (e, f), with anchor r_I and last edge sequence $\mathcal{E}_I$. The procedure `project`(I, e_1) simply finds the interval I_1 on e_1 that is hit by the monotone paths through I. If r_I is a vertex, we just use the fact that monotone paths unfold into straight lines to determine how the "wedge" of paths with anchor r_I and edge sequence $\mathcal{E}_I$ is extended into f'. (See Figure 8.) If r_I is a boundary edge site, we also use the fact that an optimal monotone path extends onto f' such that it is perpendicular to the unfolded image (along $\mathcal{E}_I$) of r_I onto f'. (This is depicted in Figure 8.)

A call to the procedure `insert-interval`(I, e) restores the following invariants for the candidate intervals on an edge-face pair (e, f):

1. The intervals have pairwise disjoint interiors, which means they can be ordered along the edge e.
2. The ordering of the intervals along e is consistent with the ordering of the

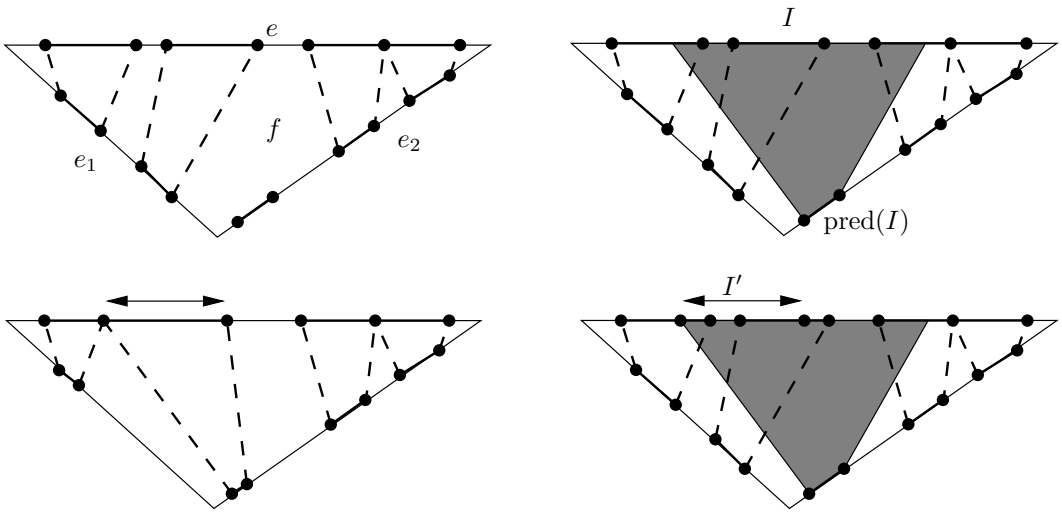


FIG. 9. *Clockwise from top left:* (1) *the candidate intervals on edge-face pair* (e, f)*, and (portions of) their predecessors on edges* e_1 *and* e_2 *on* f*;* (2) *the interval* I *is propagated from pred*(I) *and* `insert-interval`(I, e) *is called;* (3) *the trimmed subinterval* I' *of* I*; and* (4) *the list of intervals on* (e, f) *after the call is completed.*

predecessors of the intervals on the other two edges of f.

Suppose that the procedure `insert-interval`(I, e) is invoked on the edge-face pair (e, f), and that e_1 and e_2 are the other two edges of f. We assume that prior to the newly introduced interval I, the above invariants hold for the list of candidate intervals $\mathcal{I}$ on (e, f). Let us orient the edge e in an arbitrary way; this fixes an ordering $\prec$ of $\mathcal{I}$ and a corresponding ordering $\prec$ of the predecessors of intervals in $\mathcal{I} \cup I$. Let $\mathcal{L} \subseteq \mathcal{I}$ be the set of intervals I_j such that $\text{pred}(I_j) \prec \text{pred}(I)$, and let $\mathcal{R} = \mathcal{I} - \mathcal{L}$.

Let $I' \subseteq I$ be the set of points x such that the distance of x from Q through $\text{pred}(I)$ is smaller than the distance from Q through $\text{pred}(I_j)$ for any $I_j \in \mathcal{I}$. We can show that I' will be an interval. We introduce I' into the list of candidate intervals on (e, f). We truncate each $I_j \in \mathcal{L}$ (resp., $I_j \in \mathcal{R}$) to the subinterval that precedes (resp., succeeds) I' according to the $\prec$ ordering. This completes the description of procedure `insert-interval`(I, e). (See Figure 9 for an illustration.) At the end of the procedure, the invariants are restored. The key property of the procedure is that in restoring the invariants, it never trims a shortest monotone path. The implementation details of the procedure are identical to the corresponding procedure in Mitchell, Mount, and Papadimitriou [22].

This completes the description of the algorithm. At termination, we are left with a bunch of candidate intervals on each edge-face pair. The correctness of the algorithm, that is, the claim that these are actually the intervals of optimality, follows from the fact we never trim a shortest monotone path in any of the calls to `insert-interval`.

7.2. Running time analysis. To bound the running time of the algorithm, we first bound the number of events. An event corresponds to the "permanent labeling" of a frontier point α_I of candidate interval I on edge-face pair (e, f). Suppose I has anchor r_I and last edge sequence $\mathcal{E}_I$. Using the fact that we always pick the interval with the smallest priority, we can show that a portion of I (that includes α_I) is in fact an interval of optimality on (e, f) with anchor r_I and last edge sequence $\mathcal{E}_I$.

(This argument is similar in spirit to the argument for the correctness of Dijkstra's algorithm.) We charge the event to this interval of optimality I. Each interval of optimality is charged only once, so by Lemma 7.3, we get a charge of $O(r)$ per edge-face pair. As there are only $O(r)$ edge-face pairs, it follows that there are only $O(r^2)$ events. From this, we can show that the running time of the algorithm is $O(r^2 \log r)$, as in Mitchell, Mount, and Papadimitriou [22].

We now describe how the output of the algorithm is used to answer shortest distance queries. Given a query point x which is either a vertex of R or a point on the boundary edge of R, we locate the interval of optimality I in which it lies. (For a vertex v, I is the interval that $ptr(v)$ points to.) Suppose I is an interval on edge-face pair (e, f) with anchor r_I and last edge sequence $\mathcal{E}_I$. If r_I is a vertex, the distance of x from Q is $d(r_I)$ plus the distance of r_I from x along $\mathcal{E}_I$. If r_I is an edge, the distance of x from Q is the shortest distance of r_I to x along $\mathcal{E}_I$. These distances can be computed in constant time since $\overline{r_I}$, the image of r_I when $\mathcal{E}_I$ is unfolded onto f, is known. Since I can be found in $O(\log r)$ time using a binary search, we can answer shortest distance queries in $O(\log r)$ time. With minor modifications, we can also report the point $q \in Q$ that is closest to x in $O(\log r)$ time. If we desire a shortest path between x and q, we go back along the edge sequence $\mathcal{E}_I$ to r and find the shortest path between r and q. This "back-tracing" takes $O(r)$ time. Note that all these queries can be answered within the same time bounds even if x is a point on an internal edge e: we simply consider both intervals of optimality in which it lies and choose the one which gives the shorter distance to Q. We conclude by stating the main result of this section.

LEMMA 7.4. *Suppose that we are given a polyhedral patch R and a region (set of points) Q on R specified as a union of some vertices of R and some boundary edges of R. We can preprocess the patch in $O(r^2 \log r)$ time, where r is the number of faces of the patch, so that the following queries can be performed. Given a point x on any edge of the patch, we can report the closest point $q \in Q$ to x and the shortest path distance of q from x in $O(\log r)$ time. We can also report a shortest path between q and x in $O(r)$ time.*

8. Conclusions. After the initial submission of this paper, there have been some remarkable developments on computing shortest paths on a terrain. Kapoor [18] has claimed an $O(n \log^2 n)$-time algorithm for the exact shortest path problem, but his algorithm is complex, and we await the full version of the paper to verify the details. Aleksandrov, Maheshwari, and Sack [4] describe an $O((n/\varepsilon) \log n)$-time algorithm for computing a $(1 + \varepsilon)$-path, provided that the edges of the terrain satisfy certain geometric properties.

REFERENCES

[1] P. K. AGARWAL, S. HAR-PELED, M. SHARIR, AND K. R. VARADARAJAN, *Approximate shortest paths on a convex polytope in three dimensions,* J. ACM, 44 (1997), pp. 567–584.

[2] V. AKMAN, *Unobstructed Shortest Paths in Polyhedral Environments,* Lecture Notes in Comput. Sci. 251, Springer-Verlag, New York, 1987.

[3] L. ALEKSANDROV, M. LANTHIER, A. MAHESHWARI, AND J.-R. SACK, *An ϵ-approximation algorithm for weighted shortest paths on polyhedral surfaces*, Algorithm Theory—SWAT '98, Stockholm, 1998, Lecture Notes in Comput. Sci. 1432, Springer-Verlag, Berlin, pp. 11–22.

[4] L. ALEKSANDROV, A. MAHESHWARI, AND J.-R. SACK, *Approximation algorithms for shortest path problems*, in Proceedings of the 32nd Annual ACM Symposium on the Theory of Computing, Portland, OR, 2000, pp. 286–295.

[5] S. Arya, D. M. Mount, and M. Smid, *Randomized and deterministic algorithms for geometric spanners of small diameter*, in Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 703–712.

[6] J. Chen and Y. Han, *Shortest paths on a polyhedron. Part* 1: *Computing shortest paths*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 127–144.

[7] J. Choi, J. Sellen, and C. K. Yap, *Approximate Euclidean shortest paths in* 3*-space*, Internat. J. Comput. Geom. Appl., 7 (1997), pp. 271–295.

[8] K. L. Clarkson, *Approximation algorithms for shortest path motion planning*, in Proceedings of the 19th Annual ACM Symposium on the Theory of Computing, New York, 1987, pp. 56–65.

[9] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1990.

[10] X. Doug, *Finding a Geodesic Path on a* 3*D Triangulated Surface*, Master's thesis, Mississippi State University, Starkville, MS, 1995.

[11] W. R. Franklin and V. Akman, *Shortest paths between source and goal points located on/around a convex polyhedron*, in Proceedings of the 22nd Allerton Conference on Commun. Control Comput., 1984.

[12] W. R. Franklin and V. Akman, *Shortest paths in* 3*-space, Voronoi diagrams with barriers, and related complexity and algebraic issues*, in Proceedings of the NATO Advanced Study Institute on Fundamental Algorithms for Computer Graphics, NATO Adv. Sci. Intst. Series F Comput. Systems Sci. 17, Springer-Verlag, Berlin, 1985, pp. 895–917.

[13] W. R. Franklin, V. Akman, and C. Verrilli, *Voronoi diagrams with barriers and on polyhedra for minimal path planning*, Visual Comput., 1 (1985), pp. 133–150.

[14] G. N. Frederickson, *Fast algorithms for shortest paths in planar graphs, with applications*, SIAM J. Comput., 16 (1987), pp. 1004–1022.

[15] S. Har-Peled, *Constructing approximate shortest path maps in three dimensions*, SIAM J. Comput., 28 (1999), pp. 1182–1197.

[16] J. Hershberger and S. Suri, *Practical methods for approximating shortest paths on a convex polytope in* $\Re^3$, Comput. Geom., 10 (1998), pp. 31–46.

[17] K. Kant and S. Zucker, *Toward efficient trajectory planning: The path-velocity decomposition*, Internat. J. Robot. Res., 5 (1986), pp. 72–89.

[18] S. Kapoor, *Efficient computation of geodesic shortest paths*, in Proceedings of the 31st Annual ACM Symposium on the Theory of Computing, Atlanta, GA, 1999, pp. 770–779.

[19] M. Lanthier, A. Maheshwari, and J.-R. Sack, *Approximating weighted shortest paths on polyhedral surfaces*, in Proceedings of the 13th Annual ACM Symposium on Computational Geometry, Nice, France, 1997, pp. 274–283.

[20] R. J. Lipton and R. E. Tarjan, *A separator theorem for planar graphs*, SIAM J. Appl. Math., 36 (1979), pp. 177–189.

[21] C. Mata and J. S. B. Mitchell, *A new algorithm for computing shortest paths in weighted planar subdivisions*, in Proceedings of the 13th Annual ACM Symposium on Computational Geometry, Nice, France, 1997, pp. 264–273.

[22] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, *The discrete geodesic problem*, SIAM J. Comput., 16 (1987), pp. 647–668.

[23] D. M. Mount, *Voronoi Diagrams on the Surface of a Polyhedron*, Technical Report 1496, Department of Computer Science, University of Maryland, College Park, MD, 1985.

[24] C. H. Papadimitriou, *An algorithm for shortest-path motion in three dimensions*, Inform. Process. Lett., 20 (1985), pp. 259–263.

[25] B. Porter, S. Mohamad, and T. Crossley, *Genetic computation of geodesic on three-dimensional curved surfaces*, in Proceedings of the 1st International IEE/IEEE Conference on Genetic Algorithms in Engineering System: Innovations and Applications, 1995.

[26] J. S. Salowe, *Constructing multidimensional spanner graphs*, Internat. J. Comput. Geom. Appl., 1 (1991), pp. 99–107.

[27] M. Sharir and A. Schorr, *On shortest paths in polyhedral spaces*, SIAM J. Comput., 15 (1986), pp. 193–215.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1341–1367

TAKING A WALK IN A PLANAR ARRANGEMENT*

SARIEL HAR-PELED†

Abstract. We present a randomized algorithm for computing portions of an arrangement of n arcs in the plane, each pair of which intersect in at most t points. We use this algorithm to perform online walks inside such an arrangement (i.e., compute all the faces that a curve, given in an online manner, crosses) and to compute a level in an arrangement, both in an output-sensitive manner. The expected running time of the algorithm is $O(\lambda_{t+2}(m+n)\log n)$, where m is the number of intersections between the walk and the given arcs.

No similarly efficient algorithm is known for the general case of arcs. For the case of lines and for certain restricted cases involving line segments, our algorithm improves the best known algorithm of [M. H. Overmars and J. van Leeuwen, *J. Comput. System Sci.,* 23 (1981), pp. 166–204] by almost a logarithmic factor.

Key words. planar arrangements, levels, single face, computational geometry

AMS subject classifications. 65Y25, 68U05

PII. S0097539799362627

1. Introduction.

Let $\hat{S}$ be a set of n x-monotone arcs in the plane, each pair of which intersect in at most t points. Computing the whole (or parts of the) arrangement $\mathcal{A}(\hat{S})$, induced by the arcs of $\hat{S}$, is one of the fundamental problems in computational geometry, and has received a lot of attention in recent years [26]. One of the basic techniques used for such constructions is based on *randomized incremental* construction of the *vertical decomposition* of the arrangement (see [5]).

If we are interested in computing only parts of the arrangement (e.g., a single face or a zone), the randomized incremental technique can still be used, but it requires nontrivial modifications [9, 11]. Intuitively, the added complexity is caused by the need to "trim" parts of the plane as the algorithm advances, so that it will not waste energy on regions which are no longer relevant. In fact, this requirement implies that such an algorithm has to know in advance which regions we are interested in at any stage during the randomized incremental construction.

A variation of this theme, with which the existing algorithms cannot cope efficiently, is the following *online* scenario: We start from a point $p = p(0) \in \mathbb{R}^2$, and we find the face f of $\mathcal{A}(\hat{S})$ that contains $p(0)$. Now the point p starts moving and traces a connected curve $\{p(t)\}_{t\geq 0}$. As our walk continues, we wish to keep track of the face of $\mathcal{A}(\hat{S})$ that contains the current point $p(t)$. The collection of these faces constitutes the *zone* of the curve $p(t)$. However, the function $p(t)$ is not assumed to be known in advance, and it may change when we cross into a new face or abruptly change direction in the middle of a face (see [4] for an application where such a scenario arises). The only work we are aware of that can deal with this problem efficiently is due to Overmars and van Leeuwen [25], and it only applies to the case of lines (and, with some simple modifications, to certain restricted cases involving line segments as

*Received by the editors October 28, 1999; accepted for publication (in revised form) July 19, 2000; published electronically October 31, 2000. A preliminary version of the paper appeared in *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. This work was supported by a grant from the U.S.–Israeli Binational Science Foundation and is part of the author's Ph.D. thesis, prepared at Tel-Aviv University under the supervision of Prof. Micha Sharir.

http://www.siam.org/journals/sicomp/30-4/36262.html

†Department of Computer Science, University of Illinois, DCL 2111, 1304 West Springfield Ave., Urbana, IL 61801 (sariel@uiuc.edu, http://uiuc.edu/~sariel/).

well).[1] It can compute such a walk in (deterministic) $O((n+m)\log^2 n)$ time, inside an arrangement of n lines, where m is the number of intersections of the walk with the lines of $\hat{S}$. This is done by maintaining dynamically the intersection of half-planes that corresponds to the current face.

In this paper, we propose a new randomized algorithm that computes the zone of the walk in a general arrangement of arcs, as above, in $O(\lambda_{t+2}(n+m)\log n)$ expected time, where $\lambda_{t+2}(n+m)$ is the maximum length of a Davenport–Schinzel sequence of order $t+2$ having $n+m$ symbols [26]. The new algorithm can be interpreted as a third "online" alternative to the algorithms of [9, 11]. The algorithm is rather simple and appears to be practical. As a matter of fact, we had implemented and experimented with a variant of the algorithm [3].

As an application of the new algorithm, we present an algorithm for computing a level in an arrangement of arcs. It computes a single level in $O(\lambda_{t+2}(n+m)\log n)$ expected time, where m is the complexity of the level. We also show how to adapt the main algorithm to obtain a point-location algorithm that locates m points in an arrangement of n arcs, as above, in expected time $O(\lambda_{t+2}(n+m+w)\log n)$, where w is the minimum number of intersections between a spanning tree connecting those query points and the given arcs.

Both results improve by almost a logarithmic factor over the best previous result of [25] for the case of lines (and for certain cases involving line segments).[2] For the case of general arcs, we are not aware of any similarly efficient previous result.

The paper is organized as follows. In section 2 we describe the algorithm. In section 3 we analyze its performance. In section 4 we mention a few applications of the algorithm, including the construction of a single level, and multiple point location. Concluding remarks are given in section 6.

2. The algorithm. In this section, we present the algorithm for performing an online walk inside a planar arrangement.

Randomized incremental construction of the zone using an oracle. Given a set $\hat{S}$ of n x-monotone arcs in the plane, so that any pair of arcs of $\hat{S}$ intersect at most t times (for some fixed constant t), let $\mathcal{A}(\hat{S})$ denote the arrangement of $\hat{S}$; namely, the partition of the plane into faces, edges, and vertices as induced by the arcs of $\hat{S}$ (see [26] for details). In the following, we need two basic geometric primitives for splitting and merging vertical trapezoid, `SplitGeom`, `MergeGeom`, illustrated in Figure 1. We assume that $\hat{S}$ is in general position, meaning that no three arcs of $\hat{S}$ have a common point, and that the x-coordinates of the intersections and endpoints of the arcs of $\hat{S}$ are pairwise distinct. The *vertical decomposition* of $\mathcal{A}(\hat{S})$, denoted by $\mathcal{A}_{\mathcal{VD}}(\hat{S})$, is the partition of the plane into vertical pseudotrapezoids, obtained by erecting two vertical segments up and down from each vertex of $\mathcal{A}(\hat{S})$ (i.e., each point of intersection between a pair of arcs and each endpoint of an arc), and by extending each of them until it either reaches an arc of $\hat{S}$, or otherwise all the way to infinity. See, e.g., [5, 26] for more details concerning vertical decompositions. To simplify (though slightly abuse) the notation, we refer to the cells of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ as *trapezoids*. A *selection* R of $\hat{S}$ is an ordered sequence of distinct elements of $\hat{S}$. By a slight abuse of notation, we also denote by R the unordered set of its elements. Let $\sigma(\hat{S})$ denote

[1] Recently, an improvement was given by Chan [7], and further improvement was given by Brodal and Jacob [6]; their algorithm can perform an update in $O(\log n \log\log n)$ amortized time and answer queries (of the kind used in this application) in $O(\log n)$ time per query.

[2] Our results are also asymptotically faster and much simpler to implement than what is yielded by the recent results of Chan [7, 8] and Brodal and Jacob [6].

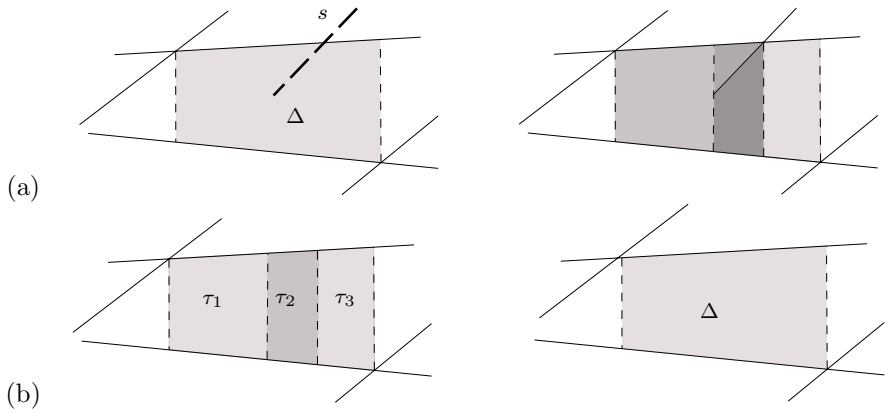


FIG. 1. *Geometric primitives:* (a) `SplitGeom`(Δ, s) *splits* Δ *into an* $O(1)$ *vertical trapezoids that cover the original trapezoid and their interior is not crossed by* s. (b) `MergeGeom`$(\{\tau_1, \tau_2, \tau_3\})$ *merge the adjacent trapezoids* τ_1, τ_2, τ_3 *with the same top and bottom arcs into a single trapezoid* Δ.

the set of all selections of $\hat{S}$. For a permutation S of $\hat{S}$, let S_i denote the subsequence consisting of the first i elements of S for $i = 0, \ldots, n$.

Computing the decomposed arrangement $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ can be done in a randomized incremental manner (see [5]). Let γ be the curve traced by the walk. For a selection $R \in \sigma(\hat{S})$, let $\mathcal{D}_\gamma(R)$ (resp., $\mathcal{Z}_\gamma(R)$) denote the *district* (resp., *zone*) of γ in $\mathcal{A}(R)$; these are, respectively, the set of all trapezoids of $\mathcal{A}_{\mathcal{VD}}(R)$ and the set of all faces of $\mathcal{A}(R)$ that have a nonempty intersection with γ. Let $\mathcal{A}_{\gamma,\mathcal{VD}}(R)$ denote the set of all trapezoids in $\mathcal{A}_{\mathcal{VD}}(R)$ that cover $\mathcal{Z}_\gamma(R)$. Our goal is to compute $\mathcal{A}_{\gamma,\mathcal{VD}}(\hat{S})$. (Alternatively, we may be interested only in computing the district $\mathcal{D}_\gamma(\hat{S})$ of γ.)

We assume for the moment that we are supplied with an oracle $\mathcal{O}(S_i, \gamma, \Delta)$, which can decide in constant time whether a given vertical trapezoid Δ is in $\mathcal{A}_{\gamma,\mathcal{VD}}(S_i)$. Equipped with this oracle, computing $\mathcal{A}_{\gamma,\mathcal{VD}}(S)$ is fairly easy, using a variant of the randomized incremental construction, outlined above. The algorithm, called `CompZoneWithOracle`, is depicted in Figure 2. We present this algorithm at a conceptual level only, because this is not the algorithm that we shall actually use. It is given to help us describe and analyze the actual online algorithm that we shall present later.

Note that the set of trapezoids $\mathcal{C}_i$ maintained by the algorithm in the ith iteration is a superset of $\mathcal{A}_{\gamma,\mathcal{VD}}(S_i)$. There might be trapezoids in $\mathcal{C}_i$ that are no longer in $\mathcal{Z}_\gamma(S_i)$. (Typically these are trapezoids that are separated from $\mathcal{Z}_\gamma(S_i)$ by an arc that does not cross their interior and is intersected after they have been created.) However, this implies that any such trapezoid will be eliminated the first time an arc that crosses it will be handled, or, if no such arc exists, at the final clean-up step of the algorithm. Moreover, the algorithm `CompZoneWithOracle` can be augmented to compute a *history* DAG (as in [12, 26]), whose nodes are the trapezoids created by the algorithm and where each trapezoid destroyed during the execution of the algorithm points to the trapezoids that were created from it. Let $\mathcal{HT}_\gamma(S_i)$ denote this structure after the ith iteration of the algorithm. Note that the out-degree of each node of $\mathcal{HT}_\gamma$ is bounded by a constant that depends on t.

Definitions. A trapezoid created by the `SplitGeom` operation of `CompZoneWithOracle` is called a *transient* trapezoid if it is later merged (in the same iteration) to form a larger trapezoid. A trapezoid generated by `CompZoneWithOracle` is

ALGORITHM `CompZoneWithOracle`($\hat{S}$, γ, $\mathcal{O}$).
`Input:` A set $\hat{S}$ of n arcs, a curve γ, an oracle $\mathcal{O}$
`Output:` $\mathcal{A}_{\gamma,\mathcal{VD}}(\hat{S})$
begin
Choose a random permutation $S = \langle s_1, s_2, \ldots, s_n \rangle$ of $\hat{S}$.
$\mathcal{C}_0 \leftarrow \{\mathbb{R}^2\}$
for i from 1 to n **do**
$\mathcal{D}_i \leftarrow \{\Delta \mid \Delta \in \mathcal{C}_{i-1}, int\,\Delta \cap s_i \neq \emptyset\}$
$Temp \leftarrow \emptyset$
for each $\Delta \in \mathcal{D}_i$ **do**
$Temp \leftarrow Temp \cup$ `SplitGeom`(Δ, s_i),
where `SplitGeom`(Δ, s) is the operation of splitting a vertical trapezoid Δ crossed by an arc s into a constant number of vertical trapezoids, as in [12], such that the new trapezoids cover Δ, and they do not intersect s in their interior.
end for
Merge all the adjacent trapezoids of $Temp$ that have the same top and bottom arcs. Let $Temp_1$ be the resulting set of trapezoids.
Let $Temp_2$ be the set of all trapezoids of $Temp_1$ that are in $\mathcal{A}_{\gamma,\mathcal{VD}}(S_i)$. Compute this set using $|Temp_1|$ calls to $\mathcal{O}$.
$\mathcal{C}_i \leftarrow (\mathcal{C}_{i-1} \setminus \mathcal{D}_i) \cup Temp_2$
end for
Remove from $\mathcal{C}_n$ all trapezoids not belonging to $\mathcal{A}_{\gamma,\mathcal{VD}}(\hat{S})$, by checking with $\mathcal{O}$ each trapezoid of $\mathcal{C}_n$.
return $\mathcal{C}_n$
end `CompZoneWithOracle`

FIG. 2. *A randomized incremental algorithm for constructing the zone of a walk in an arrangement of arcs, using an oracle.*

final if it is not transient. The *rank* $\mathrm{rank}(\Delta)$ of a trapezoid Δ is the maximum of the indices i, j of the arcs containing the bottom and top edges of Δ in the permutation S. We denote by $D(\Delta)$ the *defining set of a final trapezoid* Δ; this is the minimal set D such that $\Delta \in \mathcal{A}_{\mathcal{VD}}(D)$. It is easy to verify that $|D(\Delta)| \leq 4$. We can also define $D(\Delta)$ for a transient trapezoid Δ to be the minimal set D such that Δ can be transient during an incremental construction of $\mathcal{A}_{\mathcal{VD}}(D)$. Here it is easy to verify that $|D(\Delta)| \leq 6$. The *index* $\mathrm{index}(\Delta)$ of a trapezoid Δ is the minimum i such that $D(\Delta) \subseteq S_i$. For a trapezoid Δ, we denote by $\mathrm{cl}(\Delta)$ the *conflict list* of Δ, that is, the set of arcs of $\hat{S}$ that intersect Δ in its interior. $\mathrm{next}(\Delta)$ denotes the first element of $\mathrm{cl}(\Delta)$, according to the ordering of S.

In the algorithm, whenever we compute a trapezoid, we also compute its conflict list. The overall running time of the algorithm is dominated by the time required to compute and manipulate those conflict lists. Generally speaking, if a trapezoid is created from a parent trapezoid by splitting, we can compute the conflict list by scanning the parent conflict list and checking each arc to see if it intersects the new trapezoid. If a trapezoid is formed by merging several trapezoids, its conflict list can be computed by merging the trapezoid's conflict-list. This can be done in linear time in the size of the input conflict lists, as described below. Since a conflict list participates only in a constant number of such merge/split operations, the overall time required to manipulate those conflict lists is proportional to their overall size.

For a trapezoid Δ generated by `CompZoneWithOracle`, which was not merged into a larger trapezoid, we denote by $\mathrm{father}(\Delta)$ the trapezoid that Δ was generated from. A vertical side of a trapezoid Δ is called a *splitter*. A splitter ν is *transient* if it is not incident to the intersection point (or endpoint) that induced the vertical edge that

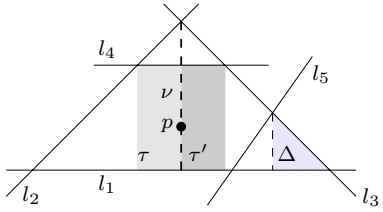


FIG. 3. *Illustration of the definitions:* ν *is a transient splitter, and thus* τ, τ' *are both transient. We have* $\mathrm{rank}(\tau) = \mathrm{index}(\tau) = 4$, $\mathrm{rank}(\Delta) = 3$, *and* $\mathrm{index}(\Delta) = 5$, *where* $S = \langle l_1, l_2, l_3, l_4, l_5 \rangle$.

contains ν. (This means that the two trapezoids adjacent to ν are transient and will be merged into a larger final trapezoid.) See Figure 3 for an illustration of some of these definitions. It is easy to verify that a trapezoid Δ is transient if and only if at least one of its bounding splitters is transient. Thus, one can decide whether a trapezoid is transient, by inspecting its splitters, in constant time.

The online algorithm constructs portions of $\mathcal{HT}_\gamma(S)$ incrementally, as they are needed. This is done by performing a sequence of point locations in the arrangement, where each such query constructs the final trapezoid of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that contains the query point, plus all ancestor trapezoids that lie on paths of $\mathcal{HT}_\gamma(S)$ from the root to that trapezoid. Informally, instead of building $\mathcal{HT}_\gamma(S)$ layer by layer, as is done by `CompZoneWithOracle`, the online algorithm constructs the DAG in an "orthogonal" depth-first search manner. The intuition behind the design and efficiency of the algorithm is that the expected number of nodes in $\mathcal{HT}_\gamma(S)$ is only $O(\lambda_{t+2}(n+m))$, whereas the overall expected running time of `CompZoneWithOracle` is $O(\lambda_{t+2}(n+m)\log n)$ (this is the expected overall size of the conflict lists of the computed nodes, which the algorithm needs to construct and manipulate). Both bounds are easy consequences of known results and will be discussed in section 3.2. Thus, in our quest to usurp the oracle, we can afford to pay an extra $O(\log n)$ time during the search for and construction of each node of $\mathcal{HT}_\gamma(S)$. This indeed will be the cost of a point-location query (ignoring the time required for constructing any new node of $\mathcal{HT}_\gamma(S)$ that the query has to pass through). Informally, the online algorithm performs essentially the same operations as the preceding algorithm, except that it executes them in a different order, and, in addition, it may revisit again and again portions of the DAG that have already been constructed as it searches down the DAG while performing point locations. However, since this extra cost is only logarithmic, it does not increase the asymptotic complexity of the algorithm.

An online algorithm for constructing the zone. Before describing the algorithm in detail, we refer the reader to Appendix A, which provides a pseudocode for some relevant procedures used by the algorithm, and Appendix B, which presents an example of how the following algorithm works. The reader is encouraged to consult with the appendices whenever the definitions and the descriptions become too vague.

Let us assume that the random permutation S of $\hat{S}$ has been fixed in advance. Note that S predetermines $\mathcal{HT}_\gamma = \mathcal{HT}_\gamma(S)$. The key observation in the online algorithm is that in order to construct a specific leaf of $\mathcal{HT}_\gamma(S)$ we do not have to maintain the entire DAG, and it suffices to compute only the parts of the DAG that lie on paths connecting the leaf with the root of $\mathcal{HT}_\gamma$. (There might be several such paths, since our structure is a DAG and not a tree.)

To facilitate this computation, we maintain a *partial history* DAG, denoted by T. The nodes of T are of two types: (i) *final nodes*, whose corresponding trapezoids

appear in $\mathcal{HT}_\gamma = \mathcal{HT}_\gamma(S)$; and (ii) *transient nodes,* which are some of the leaves of T, whose corresponding trapezoids are transient. A transient node can be easily detected since its trapezoid is transient. A final node is simply a node which is not transient. In particular, all the internal nodes of T are copies of identical nodes of $\mathcal{HT}_\gamma$ (whose corresponding trapezoids are final), while some of the leaves of T might be transient. Intuitively, T stores the portion of $\mathcal{HT}_\gamma$ that we have computed explicitly so far. The transient leaves of T delimit portions of $\mathcal{HT}_\gamma$ that have not been expanded yet. Inside each node of T, we also maintain the conflict list of the corresponding trapezoid Δ and its first element $\mathrm{next}(\Delta)$.

Suppose we wish to compute a leaf of $\mathcal{HT}_\gamma$ that contains a given point p. We denote this operation by `PointLocate`(p). We first locate the leaf of T that contains p. This is done by traversing a path in T, starting from the root of T, and going downward in each step into the child of the current trapezoid that contains p (each such step requires $O(1)$ time, because, as already noted, the out-degree of any node of $\mathcal{HT}_\gamma$, and thus of T, is bounded by a constant that depends on t). (A technical issue that we face is that usually p lies on some trapezoid boundary, so we need additional local information to determine which child to descend to at each of the above steps—see below for more details.) At the end we either reach a final leaf (with an empty conflict list), which is the required leaf of $\mathcal{HT}_\gamma$, or encounter a leaf v of T. In the latter case, we need to expand T further below v. If v represents a transient trapezoid, then the first step is to replace v by the corresponding node v^* of $\mathcal{HT}_\gamma$, obtained by merging the transient trapezoid of v with adjacent transient trapezoids, with identical top and bottom arcs, to form the final trapezoid associated with v^*. If v is a final node we expand it by splitting it with the first arc that crosses its trapezoid, using steps (iv) and (v) below.

Assume for the moment that we are supplied (for the case where v is transient) with a method (to be described shortly) to generate all those adjacent transient trapezoids, whose union forms the final trapezoid that is stored at v^* in $\mathcal{HT}_\gamma$. Then we do the following: (i) Merge all those transient trapezoids into a new (final) trapezoid Δ; (ii) compute the conflict list $\mathrm{cl}(\Delta)$ from the conflict lists of the transient trapezoids; (iii) compute the first element $s_\Delta = \mathrm{next}(\Delta)$ in $\mathrm{cl}(\Delta)$ according to the permutation S; (iv) compute all the transient or final trapezoids generated from Δ by splitting it by s_Δ (this generates $O(1)$ new trapezoids); and (v) extract from $\mathrm{cl}(\Delta)$ the conflict list $\mathrm{cl}(\Delta')$ of each new trapezoid Δ' and compute $\mathrm{next}(\Delta')$ as well.

Overall, this requires $O(k+l)$ time, where k is the number of transient trapezoids that are merged, and l is the total length of the conflict lists of these transient trapezoids. This is trivial to show for steps (i), (iii), (iv), and (v). To perform in step (ii) the merging of the conflict lists in linear time, one may use a global bit-vector structure. Namely, we initialize before the execution of the algorithm a bit-vector b of size n to be everywhere zero. To merge several conflict lists $L_1, \ldots, L_k$, we scan each list in turn, and for each of its elements s_j, we first test whether $b_j = 0$; if so, we add s_j to the output conflict list and change b_j to 1. After creating the output conflict list in this manner, we scan the output list, turning off all the bits that got turned on.

In this manner, we have upgraded a transient leaf v of T into a final node v^*. We denote this operation by `Expand`(v). We can now continue going down in T, passing to the child of Δ that contains p and repeating recursively the above procedure at that child, until constructing and reaching the desired leaf of $\mathcal{HT}_\gamma$ that contains p (namely, until we reach a node that is final and had empty conflict list).

To complete the presentation of this point-location mechanism, we describe `Expand`(v), the procedure that computes the "sibling" transient trapezoids that are adjacent to the transient trapezoid of v.

Let τ be the transient trapezoid. Then either the top arc or the bottom arc of τ are the cause of the splitting that generated τ. In particular, next(τ_f) is either the top or bottom arc of τ, where $\tau_f = \text{father}(\tau)$ denotes the trapezoid that τ was generated (split) from. This also implies that $\text{rank}(\tau) = \text{index}(\tau)$. Since τ is transient, one of the splitters of τ must be transient. Let ν denote such a transient splitter, and let us assume that ν is the right edge of τ. Note also that τ_f must be a final trapezoid, so in particular ν was generated from a final splitter (by the insertion of an arc that separated ν from the vertex that induced the bigger final splitter).

We compute the transient trapezoid τ' that lies to the right of τ and has the same top and bottom arcs, by taking the midpoint p of ν, and by performing a point-location query of p in T using (recursively) the same mechanism described above. During this point-location process, we always go down into the trapezoid Δ that contains p in its interior or on its left edge; see below for details. We stop as soon as we encounter a transient trapezoid τ' that has a left edge identical to ν. This happens when τ and τ' have the same top and bottom edges; namely, we stop when $\text{rank}(\tau) = \text{rank}(\tau')$. (Intuitively, if the trapezoid τ' has rank smaller than $\text{rank}(\tau)$, then either it fully contains ν in its interior or its left edge is longer than (and contains) ν; the first time when both τ and τ' have the same connecting edge is when their top and bottom edges are identical, namely, when $\text{rank}(\tau) = \text{rank}(\tau')$.) See below for more details in the proof of correctness of the algorithm. We continue this process of collecting adjacent transient trapezoids using point-location queries on midpoints of transient splitters, until the two extreme splitters (the left splitter of the leftmost trapezoid in the sequence and the right splitter of the rightmost trapezoid) are nontransient. We take the union of those trapezoids to be the new expanded trapezoid. See Figure 3 for a scenario where such a merging occurs. A more detailed illustration is given in Appendix B below.

Of course, during this point-location process, we might be forced into going into parts of $\mathcal{HT}_\gamma$ that are rather remote from the point p. In such a case, we will compute those parts in an online manner, by performing `PointLocate` and `Expand` calls on the relevant trapezoids that we encounter while going down T. Thus, the point-location process is recursive and might be quite substantial. Nevertheless, as will be argued below, the overall cost of the `PointLocate` and `Expand` operations is not excessive, so these operations are efficient in an amortized sense.

Let γ be the curve of the online walk whose zone we wish to compute. We consider γ to be a directed curve, supplied to us by the user through a function $\texttt{EscapePoint}_\gamma$. The function $\texttt{EscapePoint}_\gamma(p, \Delta)$ receives as input a point $p \in \gamma$, and a trapezoid Δ that contains p, and outputs the next intersection point of γ with $\partial\Delta$ following p. If γ ends before we reach $\partial\Delta$, the function returns `nil`. We assume (although this is not crucial for the algorithm) that γ does not intersect itself.

Let G_S denote the adjacency graph of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$. This is a graph having a vertex for each trapezoid in $\mathcal{A}_{\mathcal{VD}}(\hat{S})$, such that an edge connects two vertices if their corresponding trapezoids share a common vertical side. Under general position assumptions, each vertex in G_S has degree at most 4. It is easy to verify that a connected component of G_S corresponds to a face of $\mathcal{A}(\hat{S})$. Given a final leaf-trapezoid Δ, we can compute the face of $\mathcal{A}(\hat{S})$ that contains Δ by performing a depth-first search in G_S. This is done by performing point-location queries on appropriate points on

ALGORITHM `CompZoneOnline`($\hat{S}$, p, `EscapePoint`$_\gamma$).
`Input:` A set $\hat{S}$ of n arcs, a starting point p of the walk,
and a function `EscapePoint`$_\gamma$ that represents the walk
`Output:` The decomposed zone $\mathcal{A}_{\gamma,\mathcal{VD}}(\hat{S})$ of γ in $\mathcal{A}(\hat{S})$
begin
Choose a random permutation $S = \langle s_1, s_2, \ldots, s_n \rangle$ of $\hat{S}$.
$T \leftarrow \{(\mathbb{R}^2, S)\}$ - a partial history DAG with a root corresponding to
the whole plane; the conflict list of the root is the whole S.
$v \leftarrow$ `PointLocate`$(p, \cdot)$,
where `PointLocate`$(p, \cdot)$ is the leaf of $\mathcal{HT}_\gamma$ whose associated trapezoid
contains p. (The procedure has an additional flag parameter
that we ignore here; it is used in cases where p lies on trapezoid
boundaries; see Appendix A and below.)
/* All the paths in $\mathcal{HT}_\gamma$ from v to the root now exist in T. */
$\mathcal{D} \leftarrow \{\Delta_v\}$ (Δ_v is the trapezoid stored at v.)
while ($p \neq nil$) **do**
$p \leftarrow$ `EscapePoint`$_\gamma(p, \Delta_v)$
$w \leftarrow$ `PointLocate`$(p, +)$,
where $(p, +)$ denotes a point p^+ on γ just past p, and w is the next leaf
of $\mathcal{HT}_\gamma$, such that $p^+ \in \Delta_w$. This is done by performing a
point-location query in T, as described in the text, and expanding T
accordingly.
$v \leftarrow w$
$\mathcal{D} \leftarrow \mathcal{D} \cup \{\Delta_v\}$
end while
if only the district of γ needs to be computed **then**
return $\mathcal{D}$ (the district of γ in $\mathcal{A}_{\mathcal{VD}}(\hat{S})$)
$\mathcal{Z} \leftarrow \emptyset$
for each $\Delta \in \mathcal{D}$
Compute the face F of Δ_v in $\mathcal{A}_{\gamma,\mathcal{VD}}(S)$ (if it had not yet been computed).
$\mathcal{Z} \leftarrow \mathcal{Z} \cup F$
end for
return $\mathcal{Z}$.
end `CompZoneOnline`

FIG. 4. *Algorithm for constructing the zone of a walk in an arrangement of arcs in an online manner. See Appendix A for the pseudocode of the main subroutines used by* `CompZoneOnline`*, and Appendix B for an illustration of the execution of* `CompZoneOnline`.

the splitters of Δ, in a manner similar to that used in the `Expand` operation. This yields all the neighbors of Δ in G_S, and we continue in this manner until the entire connected component of G_S containing Δ is constructed.

Thus, given a walk γ, we can compute its zone by the algorithm `CompZoneOnline` depicted in Figure 4. See Appendix A for the pseudocode of the main subroutines used by `CompZoneOnline`, and Appendix B for an illustration of the execution of `CompZoneOnline`.

As will be shown in section 3.1, by the time the algorithm terminates, the final parts of T are contained in $\mathcal{HT}_\gamma$. (A proper inclusion might arise; see Remark 3.13.) In analyzing the performance of the algorithm, we first bound the overall expected time required to compute $\mathcal{HT}_\gamma$, which can be done by bounding the expected running time of `CompZoneWithOracle` (in an appropriate model of computation). Next, we will bound the additional time spent by the algorithm in traversing between adjacent trapezoids (i.e., the additional time spent in performing the point-location queries).

REMARK 2.1. *By skipping the expansion of the face that contains the current point p in* `CompZoneOnline`*, we get a more efficient algorithm that only computes the district $\mathcal{D}$ of the walk, that is, the collection of trapezoids in $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that γ crosses.*

There are cases where this will be sufficient; see section 4 *(e.g., in the adaptation of the algorithm for computing a level in an arrangement).*

3. Analysis of `CompZoneOnline`.

3.1. Correctness. In this section, we establish the correctness of `CompZoneOnline`. Before starting, we note that the correctness of `CompZoneWithOracle` is easier to establish and follows routinely from standard considerations. We therefore omit any further details concerning this issue.

The main technical issues that arise in the proof of correctness have to do with the potentially complex patterns of exploring the DAG T that can arise during the recursive execution of the `PointLocate` and `Expand` routines. The first main step in the proof is to show that the `Expand` routine always terminates properly, with a transient trapezoid that is compatible with the input one. Once this is shown, the rest of the proof is a routine, though somewhat involved, task, which employs induction on the structure of T and on the sequence of steps executed by the algorithm.

LEMMA 3.1. *During the execution of* `CompZoneOnline`*, the union of trapezoids of the leaves of* T *form a pairwise disjoint covering of the plane by vertical trapezoids.*

Proof. By induction on the steps of `CompZoneOnline`, noting that this is true initially, and that each step that modifies T either merges leaf trapezoids into a larger leaf trapezoid or splits a leaf trapezoid into subtrapezoids. □

COROLLARY 3.2. *Each conflict list, as computed for a (transient or final) trapezoid* Δ *by the procedure* `CompZoneOnline`*, is the list of all arcs of* S *that cross (the interior of)* Δ.

Proof. We prove the corollary by induction on the steps of `CompZoneOnline`. Observe that the region(s) from which Δ was generated cover Δ, and thus the union of their conflict lists must contain, by the induction hypothesis, the correct conflict list of Δ, which is thus correctly constructed by the appropriate `Expand` or `PointLocate` step. □

COROLLARY 3.3. *For a trapezoid* Δ *created by* `CompZoneOnline`*, all the arcs of* $D(\Delta)$ *appear in* S *before all the arcs of* $\mathrm{cl}(\Delta)$.

Consider an `Expand` operation that is triggered by point location at the midpoint p of a transient splitter ν that bounds a transient trapezoid τ, where τ has already been generated in T, and we wish to find the transient trapezoid τ' that shares ν with τ as a common splitter. (It is easily verified that τ' uniquely defined, given the permutation S.) Let i denote the rank of τ. As noted, i is also the index of τ. Clearly s_i must be either the top or the bottom arc of τ. Assume, without loss of generality, that ν is the right splitter of τ and that s_i is the top edge of τ. Let s_j be the bottom arc of τ, with $j < i$.

LEMMA 3.4. *During the execution of this* `Expand` *step, ignoring recursive calls, all the trapezoids that are either visited or generated fully contain* ν *either in their interior or within their left edge. Consequently, for any such trapezoid, except for the last one, either its conflict list contains* s_i *and* s_j*, or it contains* s_i *and* s_j *is the bottom arc of the trapezoid. Moreover, if this sequence of trapezoids includes a trapezoid that does contain* ν *on its left edge, then all subsequent trapezoids in the sequence have this property.*

Proof. This is shown by induction on the sequence of steps of this (nonrecursive portion of the) execution of `Expand`. For this proof, we assume that during this execution, any recursive call to `Expand` terminates correctly, with a transient trapezoid that is adjacent to the trapezoid that initiated the recursive call and has the same top and bottom arcs. If this is not the case, we assume that the algorithm aborts at

that point, and from that point on there is nothing to prove. (We thus modify the algorithm for the purpose of the proof, but a subsequent argument (in Lemma 3.5 below) will show that the algorithm never aborts.)

The first node that the `Expand` procedure visits is the root of T, and the claims clearly hold in this case. Assume they hold for all trapezoids up to and including a trapezoid Δ. Suppose first that Δ is final, and let $s_k = \text{next}(\Delta)$. By induction hypothesis, the conflict list of Δ contains s_i, so we must have $k \leq i$. Then Corollary 3.3 implies that s_k does not cross the relative interior of ν. This implies that the subtrapezoid Δ' of Δ that is split from it by s_k and contains (a point slightly to the right of) the midpoint of ν must fully contain ν in its closure. If Δ contained ν on its left side, then clearly this also holds for Δ'.

If Δ is transient, then there are two subcases: If the top and bottom edges of Δ are, respectively, s_i and s_j, then the `Expand` procedure terminates and returns Δ; the claims clearly hold in this case. Otherwise, the `Expand` procedure executes a recursive call with the midpoint of some (transient) splitter of Δ. As argued above, we may assume that this recursive call returns a transient trapezoid compatible with Δ, in the above sense. We repeat this step, if needed, until we obtain a sequence of compatible transient trapezoids, including Δ, which cannot be expanded any further. We then merge all these trapezoids into a trapezoid Δ', which clearly must be final. It is obvious, by construction, that Δ' satisfies the first assertion of the lemma.

To prove the second assertion (for transient trapezoids), assume that Δ contains ν on its left edge ν_0. It suffices to show that ν_0 is not transient (which will imply that Δ is not merged with other trapezoids on its left, so that this left splitter will be contained in ν_0), so assume to the contrary that it is transient. As already noted, a transient splitter like ν_0 is generated when we insert an arc s_ℓ that delimits ν_0 and separates it from the vertex w that induced it. Once this occurs, ν_0 will trigger a call to `Expand` which, if properly terminated, will erase ν_0 as it merges Δ with adjacent transient trapezoids. As argued above, we have $\ell < i$ (because $s_i \in \text{cl}(\Delta)$).

The same argument also implies that the trapezoid τ_f from which τ has been split (by the insertion of s_i) has a nontransient right splitter ν^* that contains ν and extends all the way to the vertex w. Thus, by Corollary 3.2, the conflict list of τ_f contains s_ℓ, so $\text{next}(\tau_f)$ cannot be s_i, contrary to assumption.

This induction step completes the proof of the lemma. □

As a consequence, it is easily verified by induction that, during the execution of this `Expand` step, including all recursive calls, no arc s_k with $k > i$ is processed.

We now show that each call to `Expand` terminates correctly.

LEMMA 3.5. *Each point-location query at the midpoint of a transient splitter generates a "compatible" transient trapezoid; that is, a transient trapezoid adjacent to the current transient trapezoid that has the same top and bottom arcs.*

Proof. Suppose the lemma is false, and consider the first call where this happens, where we order the calls in the order of returns from `Expand` (i.e., in postorder on the recursion forest). Let Δ be the transient trapezoid that initiated this call, and assume, without loss of generality, that the call started at the right splitter ν of Δ, and that the top and bottom edges of Δ are, respectively, s_i and s_j, with $i > j$. Arguing as in the proof of Lemma 3.4, we have that during the execution of the nonrecursive portion of this call, the procedure visits or generates a sequence Σ of trapezoids, each of which contains ν, and, by the induction hypothesis, all recursive calls that it executes terminate properly.

Let s_k, s_ℓ, with $k < \ell$, be the two arcs that intersect at the vertex w that induced

a splitter that contains ν (the case where w is an endpoint of an ark s_k is handled similarly). Clearly, we must have $k < \ell < i$. Moreover, w must lie above ν, or else the insertion of s_j would have disconnected ν from w; since s_i has not yet been inserted at that stage, it easily follows that ν could not have been formed at all. It is easily seen that any trapezoid $\tau \in \Sigma$ that contains ν in its interior must either contain s_k and s_ℓ in its conflict list or be bounded by s_k and contain s_ℓ in its conflict list. Hence, as can be easily verified, s_k and s_ℓ will eventually be processed in splitting operations during this execution and will consequently generate trapezoids in Σ that contain ν on their left edge. Moreover, either s_i and s_j appear in the conflict list of any such trapezoid, or s_i appears in the conflict list and s_j bounds the trapezoid on the bottom. Eventually, s_i will thus be inserted, and then the resulting trapezoid will be compatible with Δ and the procedure will terminate correctly, contrary to assumption. □

LEMMA 3.6. *For any final trapezoid Δ created by the* `Expand` *procedure, during the execution of* `CompZoneOnline`*, Δ is a trapezoid of $\mathcal{A}_{\mathcal{VD}}(S_i)$, where $i = \mathrm{index}(\Delta)$.*

Proof. The proof is shown by induction on the *depth* of the nodes in T, where the depth of a node is defined to be the length of the longest path from the root of T to this node.

The only node of depth 0 is the root of T, which is being computed during the initialization of the algorithm, and is also the only trapezoid in $\mathcal{A}_{\mathcal{VD}}(S_0)$.

Assume that the induction hypothesis holds for all trapezoids of depth $< k$, and let Δ be a final trapezoid of depth k in T. There are two cases to consider: (a) Δ has been generated by splitting a final trapezoid τ by inserting some arc s_i. (b) Δ has been obtained by merging several transient trapezoids.

In case (a), by induction hypothesis, τ is a trapezoid of $\mathcal{A}_{\mathcal{VD}}(S_j)$, where $j = \mathrm{index}(\tau)$. By Corollary 3.2, the conflict list of τ is computed correctly, so no arc s_ℓ, with $j < \ell < i$, crosses τ. Hence τ is also a trapezoid of $\mathcal{A}_{\mathcal{VD}}(S_{i-1})$, and, by construction, any final trapezoid obtained by splitting τ with s_i is a trapezoid of $\mathcal{A}_{\mathcal{VD}}(S_i)$. Since $i = \mathrm{index}(\Delta)$, this completes the induction step in this case.

In case (b), let $\tau_1, \ldots, \tau_m$ be the transient trapezoids whose merging forms Δ. By construction, all of them have the same top arc, say, s_i, and the same bottom arc, say, s_j. Suppose, without loss of generality, that $i > j$. Since these trapezoids are transient, we have, as argued above, $\mathrm{index}(\tau_\ell) = \mathrm{rank}(\tau_\ell) = i$ for each $\ell = 1, \ldots, m$. In particular, this also holds for the leftmost and rightmost trapezoids in this sequence, which is easily seen to imply that all the arcs in $D(\Delta)$ belong to S_i. Finally, since the conflict lists of the τ_ℓ's are computed correctly, and the conflict list of Δ is the union of these lists, it follows that no arc in that list belongs to S_i. In other words, Δ is a final trapezoid defined by arcs of S_i and no arc of S_i crosses its interior. This readily implies that Δ is a trapezoid of $\mathcal{A}_{\mathcal{VD}}(S_i)$, and this completes the induction step in this case, and thus completes the proof of the lemma. □

LEMMA 3.7. *All the (final) nonleaf nodes computed by* `CompZoneOnline` *appear in $\mathcal{HT}_\gamma$.*

Proof. What we really need to show is that each nonleaf-trapezoid Δ in T belongs to $\mathcal{A}_{\gamma,\mathcal{VD}}(S_{i-1})$, where s_i is the arc that has split Δ, thus making it a nonleaf. Indeed, any such trapezoid belongs to $\mathcal{HT}_\gamma$, by construction and by correctness of `CompZoneWithOracle`.

The proof proceeds by induction on the sequence of trapezoid-splitting steps taken by `CompZoneOnline`. That is, a final nonleaf-trapezoid Δ will be considered when it is split by an arc (thus becoming a nonleaf). The claim clearly holds initially for the

whole plane, stored at the root of T (and of $\mathcal{HT}_\gamma$). Let Δ be a nonleaf final trapezoid generated in T and then split by an arc s_i by `CompZoneOnline`, and suppose that all previously split nonleaf trapezoids in T appear in $\mathcal{HT}_\gamma$. The trapezoid Δ has been split as part of a point-location query with some point p. Suppose first that $p \in \gamma$ or p lies on a splitter of a final trapezoid of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$. (The later case occurs when we expand the district of γ into its zone.) Since Δ is split at that point, it follows by construction that $p \in \Delta$ and therefore Δ belongs to $\mathcal{A}_{\gamma,\mathcal{VD}}(S_{i-1})$.

Otherwise, p is the midpoint of some transient splitter ν, and the point location of p is part of some `Expand` operation. Again, p (and in fact the whole segment ν) belongs to Δ. Let τ be the transient trapezoid bounded by ν that has triggered that `Expand` operation, and let τ_0 be the first transient trapezoid (in the execution order of `CompZoneOnline`) in the sequence of compatible trapezoids that includes τ. Let τ_f be the father of τ_0, which is a final (nonleaf) trapezoid from which τ_0 has been split by an arc s_j. By induction hypothesis, τ_f belongs to $\mathcal{A}_{\gamma,\mathcal{VD}}(S_{j-1})$. Hence τ_0 is fully contained (as a set) in the zone of γ in $\mathcal{A}(S_{j-1})$. By the preceding analysis, we know that (a) s_j is the top or bottom arc of τ_0 and of τ, and (b) during the whole `Expand` operation that started at τ_0 no arc beyond s_j is inserted. It follows that p belongs to the zone of γ in $\mathcal{A}(S_{j-1})$, and that s_i, the arc that has split Δ, must satisfy $i \leq j$. Hence Δ belongs to the zone of γ in $\mathcal{A}(S_{i-1})$, which establishes the induction step and thus completes the proof of the lemma. □

LEMMA 3.8. *All the trapezoids of $\mathcal{A}_{\gamma,\mathcal{VD}}(\hat{S})$ are computed by* `CompZoneOnline`.

Proof. Consider first the sequence of trapezoids of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that constitute the district of γ in the full arrangement, in the order in which they are traversed by γ. The proof first proceeds by induction on this sequence, arguing that each of these trapezoids is produced by `CompZoneOnline`. As implied by the preceding analysis, each point-location query with a point p returns the trapezoid of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that contains p. Hence, if the kth trapezoid in the above sequence has been generated, the $EscapePoint_\gamma$ function produces a point that lies in the next trapezoid, which will therefore also be produced by `CompZoneOnline`. A similar argument holds for the subsequent stage of the algorithm that expands the district of γ into its zone. □

3.2. Running time. In this subsection, we first analyze the performance of `CompZoneWithOracle` and then use this analysis to bound the expected running time of `CompZoneOnline`. We assume that `CompZoneWithOracle` maintains for each trapezoid Δ its conflict list $\mathrm{cl}(\Delta)$ that stores the set of arcs that cross it. We also assume that each conflict list $\mathrm{cl}(\Delta)$ stores its minimal element $\mathrm{next}(\Delta)$ in the ordering of S and that each yet uninserted arc s maintains a list of all current trapezoids Δ for which $\mathrm{next}(\Delta) = s$. Then it is easy to see that the running time of the algorithm is proportional to the overall size of all the conflict lists that it generates. We also assume that a call to the Oracle $\mathcal{O}$ takes $O(1)$ time.

LEMMA 3.9. *The algorithm* `CompZoneWithOracle` *computes the zone of γ in $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ in $O\left(\lambda_{t+2}(n+m)\log n\right)$ expected time, and the expected number of trapezoids that it generates is $O\left(\lambda_{t+2}(n+m)\right)$.*

Proof. The proof is a straightforward adaptation of the proof of [9].[3] Specifically, we first make m cuts at the points where γ crosses the arcs of $\hat{S}$, thereby obtaining a collection of $m+n$ subarcs, so that $\mathcal{Z}_\gamma(\hat{S})$ becomes a single face in the new arrangement. We now insert the original arcs of $\hat{S}$ one by one in the random order S. It is easily checked that the expected number of subarcs of the r random arcs that have

[3]The algorithm of [9] has some additional overhead that is not required in `CompZoneWithOracle`.

been inserted is $O\left(r + \frac{m}{n}r\right)$. Thus, the expected number of trapezoids maintained in the rth iteration is $O\left(\lambda_{t+2}\left(r + \frac{m}{n}r\right)\right)$. Using the Clarkson–Shor sampling technique [10, 24] implies that the overall expected weight of those trapezoids in the rth iteration is $O(\lambda_{t+2}(n+m))$. However, the expected work in the rth iteration is the expected weight of the newly created trapezoids, and the probability of a trapezoid (that appears in the set of trapezoids maintained by the algorithm after the rth iteration) to be created in the rth iteration is $O(1/r)$. We conclude that the expected work in the rth iteration is $O(\lambda_{t+2}(n+m)/r)$. Summing over $r = 2, \ldots, n$, we conclude that the expected overall running time of the algorithm is $O(\lambda_{t+2}(n+m)\log n)$. ∎

LEMMA 3.10. *The expected number of transient trapezoids generated by* `CompZoneOnline` *is* $O(\lambda_{t+2}(n+m))$*, and the expected total size of their conflict lists is* $O(\lambda_{t+2}(n+m)\log n)$*.*

Proof. Each final trapezoid generated by `CompZoneOnline` might be split into $O(1)$ transient trapezoids. Each final (nonleaf) trapezoid computed by `CompZoneOnline` is also computed by `CompZoneWithOracle`, as follows from Lemma 3.7. By Lemma 3.9, the expected number of such trapezoids is $O(\lambda_{t+2}(n+m))$.

The second part of the lemma follows by a similar argument. ∎

DEFINITION 3.11. *A curve* γ *is* locally x-monotone *in* $\mathcal{A}(\hat{S})$ *if it can be decomposed inside each face of* $\mathcal{A}(\hat{S})$ *into a constant number of* x*-monotone curves.*

THEOREM 3.12. *The algorithm* `CompZoneOnline` *computes the zone of* γ *in* $\mathcal{A}(\hat{S})$ *in* $O\left(\lambda_{t+2}(n+m)\log n\right)$ *expected time, provided that* γ *is a locally* x*-monotone curve in* $\mathcal{A}(\hat{S})$*.*

Proof. The time spent by `CompZoneOnline` is bounded by the time required to construct the history DAG, by the time spent in maintaining the conflict lists of the trapezoids, and by the time spent on performing point-location queries, as we move from one trapezoid to another in $\mathcal{A}_{\gamma,\mathcal{VD}}(S)$.

By Lemmas 3.9 and 3.10, the expected time spent on maintaining the conflict lists of the trapezoids computed by the algorithm is $O(\lambda_{t+2}(n+m)\log n)$, since the total time spent on handling the conflict lists is proportional to their total length. By Lemma 3.10, the expected total size of those conflict lists is $O(\lambda_{t+2}(n+m)\log n)$.

Moreover, the depth of the history DAG constructed by the algorithm is $O(\log n)$ with a probability polynomially close to 1 [24]. Thus, the expected time spent directly on performing a single point-location query (that is, the number of trapezoids that contain the query point and are visited or generated during this point-location step) is $O(\log n)$. The curve γ is locally x-monotone, which implies that it intersects each splitter of any trapezoid of any $\mathcal{A}_{\gamma,\mathcal{VD}}(\hat{S})$ at most $O(1)$ times. Thus, the expected number of point-location queries performed by the algorithm is proportional to the expected number of transient and final trapezoids created, plus $O(m)$. By Lemma 3.10, we have that the expected running time is

$$O\left(\lambda_{t+2}(n+m)\log n + (\lambda_{t+2}(n+m)+m)\log n\right) = O\left(\lambda_{t+2}(n+m)\log n\right). \quad ∎$$

REMARK 3.13. *Note that* `CompZoneWithOracle` *computes the zone of* γ *in* $\mathcal{A}_{\mathcal{VD}}(S_i)$ *for each* $i = 1, \ldots, n$*. As a consequence, it might compute a trapezoid* $\Delta \in \mathcal{A}_{\gamma,\mathcal{VD}}(S_i)$ *that does not intersect the zone of* γ *in* $\mathcal{A}_{\gamma,\mathcal{VD}}(S)$*. In particular, such a trapezoid* Δ *will not be computed by* `CompZoneOnline`*. This is a slackness in our analysis that we currently do not know whether it can be exploited to further improve the analysis of the algorithm. (We suspect that it cannot improve the above asymptotic bound on the running time.)*

REMARK 3.14. *The only classical result of this type that we are aware of is due to Overmars and van Leeuwen* [25]. *It maintains dynamically the intersection of n half-planes in (deterministic) $O(\log^2 n)$ time for each insertion or deletion operation. This procedure can be used to perform walks inside line arrangements in (deterministic) $O((n+m)\log^2 n)$ time, where m is the number of intersections of the walk with the lines. Our algorithm is somewhat simpler and is faster than the algorithm of* [25] *by nearly a logarithmic factor, and, most importantly, applies to more general arrangements.*[4]

The technique of [25] *can also be used to perform* x-monotone *online walks in arrangement of segments. We simply regard each segment as the full line that contains it, but make it active only when the walk passes vertically above or below the segment. This means that, in addition to the usual updates that the algorithm performs, we also insert (resp., delete) segments when the walk becomes covertical with their left (resp., right) endpoint. (If the walk is not x-monotone, this may slow down the algorithm considerably.)*

As for general arcs (or nonmonotone walks in arrangements of segments), we are not aware of any result of this type in the literature. Of course, if the curve γ is known in advance (and is simple, in the sense that one can compute quickly its intersections with any arc of $\hat{S}$), we can compute the single face containing γ in an appropriately modified arrangement (as in the proof of the general planar zone theorem [26, *Theorem* 5.11]*; see also the proof of Lemma* 3.9 *using the algorithms of* [11, 9]. *These algorithms (especially the first one) are slightly simpler than the algorithm of Theorem* 3.12, *although they have the same expected performance. However, these algorithms are useless for online walks, and they are probably slower than our algorithm in practice, as they either maintain additional complicated data-structures* [9] *or perform additional redundant computation of regions that lie outside the zone of γ* [11].

Recently, several algorithms for performing online walks were implemented [3], *including a variant of the algorithm presented here, which exhibited satisfactory performance in practice.*

4. Applications. In this section we present several applications of the algorithm `CompZoneOnline`.

4.1. Computing a level in an arrangement of arcs. In this subsection we show how to modify the algorithm of the previous section to compute a level in an arrangement of x-monotone arcs.

DEFINITION 4.1. *Let $\hat{S}$ be, as above, a set of n x-monotone arcs in the plane, any pair of which intersect at most t times (for some fixed constant t). We also assume, as above, that $\hat{S}$ is in general position. The* level *of a point in the plane is the number of arcs of $\hat{S}$ lying strictly below it. Consider the closure E_l of the set of all points on the arcs of $\hat{S}$ having level l (for $0 \leq l < n$). E_l is an x-monotone (not necessarily connected) curve (which is polygonal in the case of lines or segments), which is called the* level l *of the arrangement $\mathcal{A}(\hat{S})$. At x-coordinates where a vertical line intersects less than $l+1$ arcs of S, we consider E_l to be undefined.*

Levels are a fundamental structure in computational and combinatorial geometry and have been subject to intensive research in recent years (see [1, 13, 27, 28]). Tight

[4]Recently, Chan [7] improved the result of [25], providing a data structure for maintaining intersection of half-planes in $O(\log^{1+\varepsilon} n)$ amortized time for each update. His data structure is considerably more complicated than ours and that of Overmars and van Leeuwen, and currently seems to be only of theoretical significance. Moreover, our algorithm is still faster than Chan's (by a factor of $O(\log^{\varepsilon} n)$). Very recently, this result was further improved by [6].

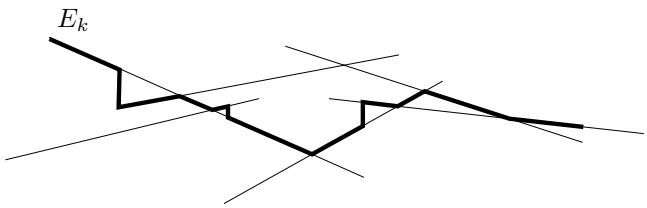


FIG. 5. *The first level in an arrangement of segments. (The vertical edges show the jump discontinuities of the level, but are not part of the level.)*

bounds on the complexity of a single level, even for arrarngements of lines, proved to be surprisingly hard to obtain. Currently, the best-known upper bound for the case of lines is $O(n(l+1)^{1/3})$ [13], while the lower bound is $\Omega(n \log(l+1))$ [14].[5] See also [1, 28] for weaker upper bounds for other classes of arcs.

First, note that if $\hat{S}$ is a set of lines, then, once we know the leftmost ray that belongs to E_l, the level l is locally defined: as we move from left to right along E_l, each time we encounter an intersection point (a vertex of $\mathcal{A}(\hat{S})$) we have to change the line that we traverse. (This is also depicted in Figure 5.) In particular, we can compute the level E_l in $O(\lambda_3(n+|E_l|)\log n)$ time using `CompZoneOnline`. The same procedure can be used to compute a level in an arrangement of more general arcs. The only nonlocal behavior we have to watch for are jump discontinuities of the level caused when an endpoint of an arc appears below the current level, or when the current level reaches an endpoint of an arc (see Figure 5). See below for details concerning the handling of those jumps.

In the following, let $0 \le l < n$ be a prescribed parameter. Let E_l denote the level l in the arrangement $\mathcal{A}(\hat{S})$.

The following adaptation of `CompZoneOnline` to our setting is rather straightforward, but we include it for the sake of completeness. We sort the endpoints of the arcs of $\hat{S}$ by their x-coordinates. Each time our walk reaches the x-coordinate of the next endpoint, we update E_l by jumping up or down to the next arc, if needed. This additional work requires $O(n \log n)$ time.

If the level reaches the x-coordinate x_0 of a right endpoint of an arc, past which there are fewer than $l+1$ arcs intersecting a vertical line, then the level lies on the highest arc just to the left of x_0 and it ceases to be defined just to the right of x_0. In this case, our walk climbs to the line $y = +\infty$ and moves along the line (effectively tracing a sequence of topmost trapezoids of $\mathcal{A}(\hat{S})$) until it reaches the x-coordinate of a left endpoint of an arc, following which we might have again $l+1$ arcs crossing a vertical line. If so, the walk then descends on the topmost arc and continues to trace the level l. For simplicity, we continue the discussion of the algorithm assuming that E_l is everywhere defined.

During the walk, we maintain the invariant that the top edge of the current trapezoid is part of E_l. To compute the first trapezoid in the walk, we compute the intersection of level l with the y-axis (this can be done by sorting the arcs according to their intersections with the y-axis). Let p_0 be this starting point. We perform a point-location query with p_0 in our virtual history DAG to compute the starting trapezoid Δ_0 (containing p_0 on its top arc).

Now, by walking to the right of Δ_0 we can compute the part of E_l lying to the

[5] Recently, a slightly larger lower bound has been announced by Tóth [29].

right of the y-axis. Let Δ be the current trapezoid maintained by the algorithm, such that its top edge is a part of E_l. Let $p(\Delta)$ denote the top right vertex of Δ. By performing point-location queries in our partial history DAG T, we can compute all the trapezoids of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that contain $p(\Delta)$. (By our general position assumption, the number of such trapezoids is at most 6; this number materializes when $p(\Delta)$ lies in the intersection of two arcs.) By inspecting this set of trapezoids, one can decide where E_l continues to the right of Δ and determine the next trapezoid having E_l as its roof. The algorithm sets Δ to be this trapezoid.

If the algorithm reaches an x-coordinate of an endpoint of an arc, we have to update E_l by jumping up (if this is the right endpoint of an arc and it lies on or below the level) or down (if it is a left endpoint and lies below the level); namely, we set Δ to be the trapezoid lying above (or below) the current Δ.

The algorithm continues in this manner until reaching the rightmost edge of E_l. The algorithm then performs a symmetric walk to the left of the y-axis to compute the other portion of the level.

Let `CompLevel` denote this modified algorithm. We summarize our result as follows.

THEOREM 4.2. *The algorithm* `CompLevel` *computes the level l in $\mathcal{A}(\hat{S})$ in $O\left(\lambda_{t+2}(n+|E_l|)\log n\right)$ expected time.*

REMARK 4.3. *Since* `CompLevel` *is online, we can use it to compute the first m' edges or vertices of E_l in expected $O(\lambda_{t+2}(n+m')\log n)$ time.*

REMARK 4.4. *A straightforward extension of* `CompLevel` *allows us to compute any connected path within the union of $\hat{S}$ (i.e., we restrict our "walk" to the arcs of $\hat{S}$) in an online manner, in randomized expected time $O\left(\lambda_{t+2}(m+n)\log n\right)$, where m is the number of vertices of the path. As above, the extended version can also handle vertical jumps between adjacent arcs during the walk.*

REMARK 4.5. *For the case of lines, one can use the algorithm of* [25] *to construct a level E_l in $O(n\log n+|E_l|\log^2 n)$ deterministic time, as described, e.g., in* [15]. *The same technique, with a simple modification, also works for the case of line segments, with the same complexity bounds. Our algorithm is faster in these cases by nearly a logarithmic factor.*

As already mentioned, recently Chan [7] *presented a faster algorithm for the dynamic maintenance of the intersection of half-planes, requiring $O(\log^{1+\varepsilon} n)$ amortized time for each operation. Thus, one can compute the level l in $O(n\log n+|E_l|\log^{1+\varepsilon} n)$ deterministic time. Chan* [8] *also showed that by using the algorithm of* [2] *one can compute the level l in $O(n+|E_l|\alpha(n)^2\log n)$ randomized expected time. Those results were very recently improved by Brodal and Jacob* [6]. *We note, however, that our algorithm is still faster and simpler than those two algorithms.*

4.2. Other applications. In this subsection, we provide some additional applications of `CompZoneOnline` and `CompLevel`.

THEOREM 4.6. *Let L be a set of n lines in the plane, and let $0<\varepsilon\leq 1$ be a prescribed constant. Then one can compute a $(1/r)$-cutting of $\mathcal{A}(L)$, having at most $(1+\varepsilon)(8r^2-2r+4)$ trapezoids. The expected running time of the algorithm is $O\left(\left(1+\frac{1}{\varepsilon}\right)nr\alpha(n)\log n\right)$, where $\alpha(n)$ is the inverse of the Ackermann function* [26].

Proof. We obtain the proof by plugging the algorithm of Theorem 4.2 and Remark 4.3 into the algorithm described in [18]. □

For a discussion of cuttings of small asymptotic size and their applications, see [22, 18].

REMARK 4.7. *Theorem* 4.6 *improves the previous result of* [18] *by almost a logarithmic factor.*

Once we have computed the level l (in an arrangement of general arcs), we can clip the arcs to their portions below the level. Using those clipped arcs as input, we can compute the portion of the arrangement below the level l (i.e., the first l levels of $\mathcal{A}(\hat{S})$) in $O((m+n)\log n+r)$ time, where $m = |E_l|$ is the complexity of the level l, and r is the complexity of the first l levels of $\mathcal{A}(\hat{S})$, using, e.g., the algorithm described in [24]. This improves over the previous result of [16] that computes this portion of the arrangement of lines in $O(n \log n + nl)$ time. (Note that this running time is *not* output sensitive: It is easy to come up with examples where the complexity of the first l levels is only $O(l^2)$.)

5. Implementation. The algorithm described in this paper was implemented and compared to some other heuristics/algorithms for constructing zones in an online manner for an arrangement of lines; see [3]. The source code of our program is available from [19]. We had implemented a competing algorithm, the following variant `CompZonePoly`, which differs from `CompZoneOnline` in the following two key points:

- `CompZonePoly` does not perform merging of adjacent compatible regions. Thus, the history DAG is now a tree.
- Each node in the history tree corresponds to a convex polygon of bounded complexity. Namely, if the polygon that corresponds to a node has more than c edges (where c is a prescribed constant), then the polygon is being further split into two regions, and the corresponding node becomes the parent of two new nodes. The motivation for this variant is that the average number of vertices of a face in an arrangement of lines is about 4. Thus, in this representation most faces (and intermediate faces) will correspond to a single node in the history tree.

Currently, we do not have any bounds on the performance of `CompZonePoly` and we leave it as an open question for further research. Nevertheless, `CompZonePoly` performs extremely well in practice and was one of the two fastest algorithms tested in [3].

5.1. Geometric filtering. To overcome the problems of robustness and degeneracies, we had used the exact arithmetic as provided by the rational numbers of LEDA [23]. Unfortunately, using exact arithmetic in a naive way slows the program by a factor of 20–40 [18]. One possible way to achieve reasonable performance is to use filtering techniques. Here, one uses representation of numbers that maintain the history of the computations that generated them, so that if necessary the computations are recomputed using a higher level of precision. Such arithmetic types are provided by LEDA real and LEDA rational kernel. While filtering may be a carefree approach to this problem, as one can apply it easily to a program without rewriting, its performance is still inferior compared to the technique described below.

The idea is to implement filtering in the geometric level. Here each geometric entity has two representations: one is a floating-point representation (i.e., inexact) and the other one is an implicit exact representation. For example, a point is represented in its floating-point Cartesian representation and its logical representation; that is, the geometric operations and entities used to create it. For example, a point p might be defined to be the point lying on the line l and having the same x-coordinate as a point q.

Thus, when a geometric primitive is being called, it is first computed using floating-point arithmetic. If the computation result lies below a certain threshold,

then the algorithm recomputes the primitive using exact arithmetic. This might require computing the exact representation of the points and lines used in this primitive. While in general this is worse than arithmetic filtering, it performs better in our scenario, as the depth of computation of vertices and edges in planar arrangement is bounded.

Furthermore, since this representation preserves the combinatorial information, one can use this information to resolve geometric decisions without resorting to exact arithmetic. For example, consider a point p that is defined to be the intersection point between the lines l_1, l_2, and the algorithm calls a primitive `isOnLine` to decide if p lies on the line l_1. Here, after the floating-point predicate had failed, the predicate decides using the logical representation of $p = l_1 \cap l_2$ that it lies on l_1, and thus it return `true`.

Note that this is a scenario where arithmetic filtering will perform badly, for in this case the arithmetic filtering will first carry out the computations using floating-point arithmetic, and after those operations fail the computations will be reperformed using a higher level of precision, using some kind of a gap-bound so that it resolves the predicate correctly.

We refer to the above approach as *geometric filtering*. It seems to be the most natural approach to the problem of robustness, although the considerable benefits of this approach in practice are not widely known. For example, in the case of the algorithms of [3], the usage of geometric filtering speeded up the algorithms by a factor of 2–3. In practice, virtually all computations are performed using floating-point arithmetic, and only negligible part of the computations resort to exact arithmetic.

A very similar idea was recently implemented by Funke and Mehlhorn [17]. For further details about our approach, see [19].

5.2. Empirical results. The testing was carried out using the inputs of [3], and the results are depicted in Table 1. The tests were performed on a dual Pentium II 450MHz with 512MB memory using Linux. Each entry in the table is the average of 25 executions of the program on this input. As can be seen from the table, `CompZoneOnline` is considerably slower (by a factor 2–8) than `CompZonePoly`.

The disappointing performance provided by `CompZoneOnline` is mainly caused by the expensive `Expand` operations (involving repeated point-location queries in the DAG). Of course, `CompZonePoly` does not perform `Expand` operations. Furthermore, as testified by Table 1, the usage of vertical trapezoids by `CompZoneOnline` is inherently inefficient as it blows up the number of nodes in the associated history structure by a factor of 2–3 compared to the number of nodes created by `CompZonePoly`.

In addition, the implementation of `CompZonePoly` associates with each line l in the conflict list of a region P, the two edges of ∂P that l intersects. When computing the conflict lists of the children of the node that corresponds to P, one can sometimes compute what conflict lists l belongs to without executing a single geometric primitive. It is not clear how one can implement a similarly efficient scheme for the computation of the conflict lists of vertical trapezoids.

6. Conclusions. In this paper we have presented a new randomized algorithm for computing a zone in a planar arrangement in an online fashion. This algorithm is the first efficient algorithm for the case of planar arcs, it performs faster (by nearly a logarithmic factor) than the algorithm of [25] for the case of lines and for the case of an x-monotone walk in an arrangement of segments, and it is considerably simpler. (It is also faster and much simpler than the recent algorithm of [7].) We also presented an efficient randomized algorithm for computing a level in an arrangement of arcs in

TABLE 1

Results for the two implemntations. Running times are in seconds. The number of nodes created by `CompZoneOnline` *is considerably larger than the final number of nodes in the resulting DAG, as the algorithm merge nodes during its execution.*

			`CompZoneOnline`			`CompZonePoly`	
Input	Lines #	Faces #	Time	Nodes #	Nodes created	Time	Nodes #
test	42	26	0.053	383	457	0.009	203
reg2000	2,004	667	1.449	17,651	21,419	0.228	6,735
rgl2000	2,004	6,271	6.331	93,729	117,324	0.890	36,853
rnd2000	2,004	1,411	1.458	25,072	32,064	0.295	9,844
zon2000	2,004	11,803	10.754	191,089	245,602	1.857	75,067
big2000	2,004	1	1.215	11,649	15,409	0.662	6,779
reg8000	8,004	2,664	6.122	70,409	85,376	1.007	26,841
zon8000	8,002	47,411	49.252	811,471	1,039,358	8.551	318,637
big8000	8,004	1	5.150	46,965	62,184	2.845	27,344

the plane, whose expected running time is faster than any previous algorithm for this problem.

The main result of this paper relies on the application of point-location queries to compute the relevant parts of an "offline" structure (i.e., the history DAG). The author believes that this technique should have additional applications. In particular, this approach might be useful also for algorithms in higher dimensions. We leave this as an open question for further research.

Although the resulting algorithm seems to be only a minor variant of previous algorithms [11, 9], the author believes that the new algorithm supersedes those algorithms: (i) Implementing the new algorithm was quite easy and does not require any advanced data-structure. In particular, since the algorithm does not keep geometric adjacency information in the vertical decomposition (unlike previous algorithms) its implementation is thus considerably easier. (ii) The algorithm only computes what it *must* compute, while [11] performs a lot of redundant computations. (iii) The algorithm provides a powerful data-structure for online computation of parts of an arrangement, where the computation time is the same as a randomized incremental algorithm that uses an oracle. This enables one to compute a portion of an arrangement in a completely arbitrary order, in time identical to the time spent by a optimal randomized incremental algorithm. See [21, 20] for results that use this observation.

It is somewhat surprising that in most applications that use Overmars and van Leeuwen [25] (or Chan [7] improvement) data-structure—which is more flexible than our data-structures since it allows insertion and deletion—one can use our algorithm instead and the resulting algorithms are always faster. See [21] for more details.

The empirical results testify that this algorithm is practical, although it is slower than the heuristic `CompZonePoly` we had also tested. We currently do not have any proof of performance bounds for `CompZonePoly`, and we leave this as a question for further research. Another striking conclusion from the empirical tests is that using vertical decomposition in practice is not as efficient as using polygons having constant complexity; see [3, 18] for similar results. It seems that planar vertical decomposition should be avoided in practice, as they give inferior performance. An additional reason for avoiding vertical decompositions in practice is their vulnerability to degeneracies (for example, several vertices of the arrangement having the save x-coordinate, etc). However, if one computes the zone in arrangement of segments, or of general arcs, it seems that the usage of vertical trapezoids is most natural. We believe that in such scenarios `CompZoneOnline` will perform reasonably well compared to other algorithms.

Appendix A. Pseudocode for subroutines of `CompZoneOnline`. See Figures 6–9.

```
ALGORITHM Split(v).
   Input: A final node v in the partial history DAG T
begin
   s ← next(Δ_v), where Δ_v is the trapezoid associated with the node v
   L ← SplitGeom(Δ_v, s),
         where SplitGeom(Δ_v, s), as above, returns the collection of trapezoids
         that cover Δ_v, so that s does not intersect any of them in its interior.
   for each τ ∈ L do
      Create a new node w and attach it as a child of v in T.
      Set Δ_w to τ
      Compute cl(Δ_w) from cl(Δ_v)
      Compute next(Δ_w), the first element of cl(Δ_w)
   end for
end Split
```

FIG. 6. *Splitting a final node in T and creating its children.*

```
ALGORITHM PointLocateLeftCompatible(v, p, r).
   Input:  v - current node of T
           p - query point
           r - target rank of output trapezoid
   Output: A transient trapezoid of rank r having p on its left splitter
begin
   if rank(v) = r then
      return v
   if isTransient(v) then
      v ← Expand(v)
   if isLeaf(v) then
      Split(v)
   Let w be the child of v, so that Δ_w contains p either in its interior
         or on its left splitter
   return PointLocateLeftCompatible( w, p, r )
end PointLocateLeftCompatible
```

FIG. 7. *Computing a transient trapezoid in T that is left "compatible" with an input transient trapezoid by carrying out a point-location query in T. The algorithm also uses a symmetric routine,* `PointLocateRightCompatible`, *whose code is omitted.*

```
ALGORITHM Expand(v).
   Input:  v - current transient leaf node of T
   Output: A final node of T whose trapezoid contains Δ_v
begin
   if isFinal(Δ_v) then
      return v
   L ← {v}

   /* collect the sequence of transient trapezoids adjacent to each other
      to the right of Δ_v */
   temp ← v
   while isTransientRightSplitter(Δ_temp) do
      temp ← PointLocateLeftCompatible( root(T),
                    midPointRightSplitter( Δ_temp), rank(Δ_temp) )
      L ← L ∪ {temp}
   end while

   /* Similarly collect the sequence of transient trapezoids to the left of Δ_v */
   temp ← v
   while isTransientLeftSplitter(Δ_temp) do
      temp ← PointLocateRightCompatible( root(T),
                    midPointLeftSplitter( Δ_temp), rank(Δ_temp) )
      L ← L ∪ {temp}
   end while

   Δ ← ⋃_{u∈L} Δ_u
   Compute cl(Δ) and next(Δ) from the conflict lists of the
      nodes of L, using the global bit-vector technique.
   Add a new leaf node x to the partial history DAG T, mark x as final,
         and replace all nodes of L in T by x.
   Set Δ_x to Δ

   return x
end Expand
```

FIG. 8. *Expanding a transient leaf trapezoid of T to a final trapezoid containing it.*

```
ALGORITHM PointLocate( p, flag ).
   Input:  p - a query point
           flag - since p usually lies on the boundary of a trapezoid, flag
                  indicates the side of the splitter or arc that contains p
                  where the point location should take place
   Output: The trapezoid of A_VD(Ŝ) that contains p (in its interior
           or on the appropriate edge dictated by flag)
begin
   v ← root(T)

   while ( cl(Δ_v) ≠ ∅ ) do
      Expand(v)
      Split(v)
      v ← child of v whose trapezoid contains p; if p lies on
          the boundary of several children trapezoids, choose the one
          that is compatible with flag
   end while

   return Δ_v
end PointLocate
```

FIG. 9. *The function that performs a point-location query; that is, it computes the necessary parts of the partial history DAG T and returns the trapezoid of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that contains a query point.*

Appendix B. Taking a walk in ten easy figures. In this appendix we illustrate step by step the action of processing a single point-location query by `CompZoneOnline`. (See Figures 10–19.)

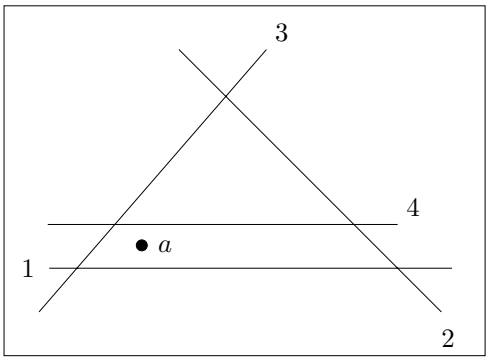


FIG. 10. *The input for* `CompZoneOnline` *is the set of segments $\hat{S} = \{1, 2, 3, 4\}$. We assume that the algorithm uses the permutation $S = (1, 2, 3, 4)$. We illustrate how* `CompZoneOnline` *carries out a point-location query to compute the trapezoid of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that contains the point a. We assume that this is the first query processed by the algorithm.*

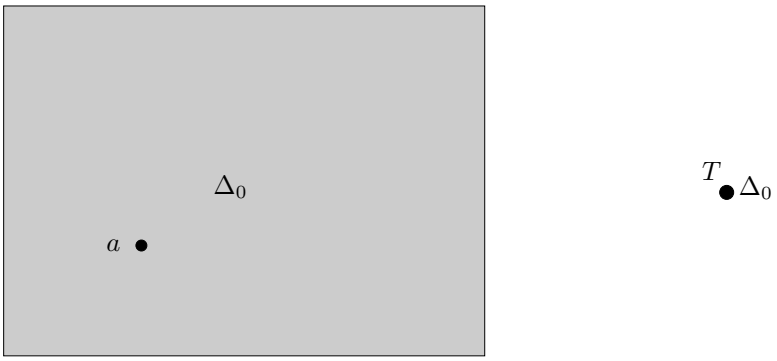


FIG. 11. `CompZoneOnline` *starts with the trapezoid* $\Delta_0 = \mathbb{R}^2$ *that corresponds to the root of the partial DAG* T.

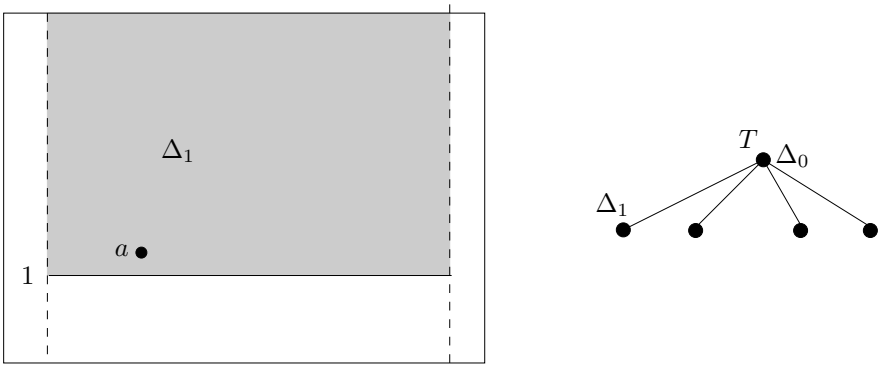


FIG. 12. `CompZoneOnline` *splits* Δ_0 *by the segment* $\text{next}(\Delta_0) = 1$, *and goes down in the DAG into the new node that corresponds to the trapezoid* Δ_1. *All children of* Δ_0 *(including* Δ_1*) are final.*

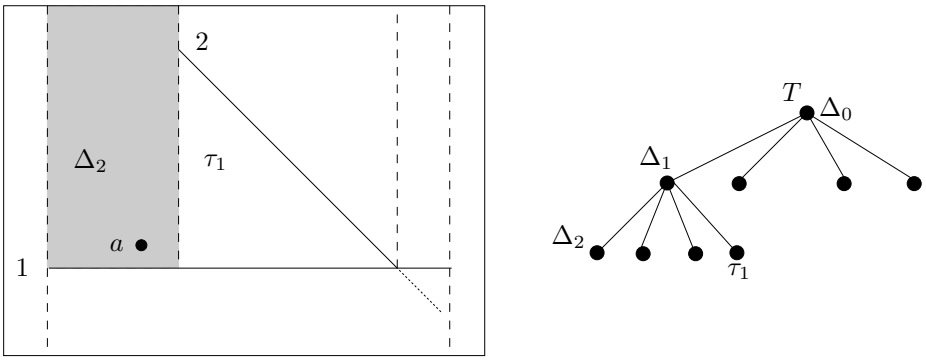


FIG. 13. *Since* Δ_1 *is final, it is split by* $\text{next}(\Delta_1) = 2$ *into four final subtrapezoids, and we go down to the child* Δ_2 *that contains* a.

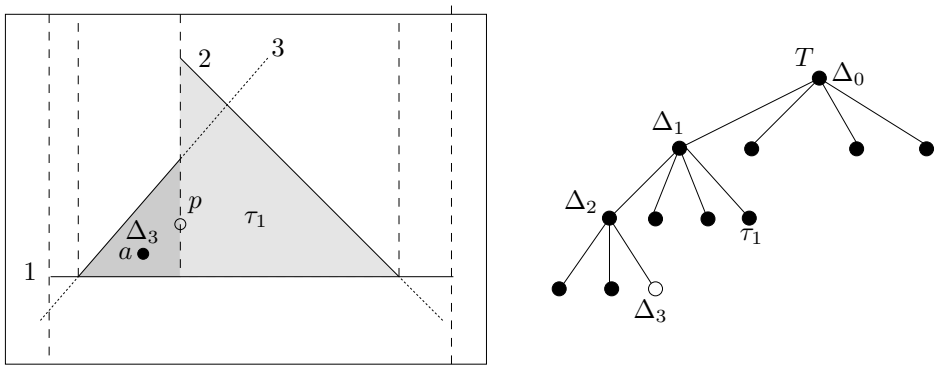


FIG. 14. `CompZoneOnline` *splits* Δ_2 *by* $\text{next}(\Delta_2) = 3$ *into three subtrapezoids. Two of them are final, and the third one,* Δ_3*, that contains* a *is transient (its right splitter is transient). We thus execute* `Expand`(Δ_3)*, which performs a point-location query (with the midpoint* p *of the right splitter). The point location goes down in the partial DAG, through* Δ_0 *and* Δ_1*, and reaches the (final) leaf that stores* τ_1*. Since* $\text{rank}(\Delta_3) = 3$ *and* $\text{rank}(\tau_1) = 2$*, those two trapezoids are not compatible (this holds also because* τ_1 *is final, whereas* Δ_3 *is not), and the algorithm continues by further splitting* τ_1 *by* $\text{next}(\tau_1) = 3$.

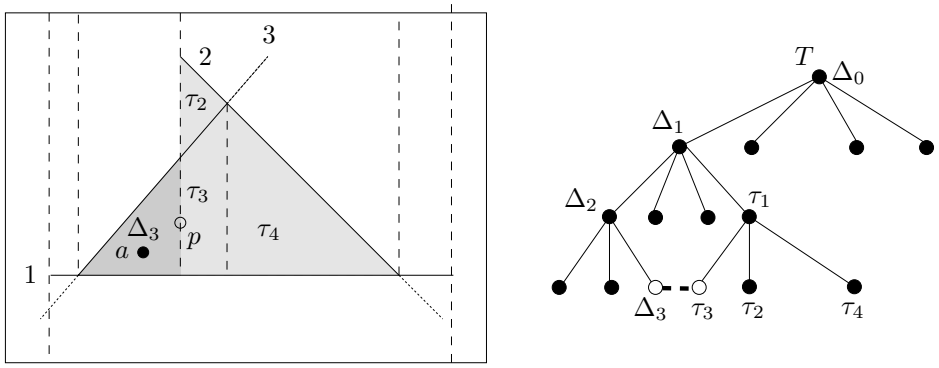


FIG. 15. *The splitting of* τ_1 *creates three children* τ_2, τ_3, τ_4*, of which* τ_2 *and* τ_4 *are final, whereas* τ_3 *is transient.* `CompZoneOnline` *goes down to the newly created* τ_3*, which contains* p *on its left splitter. Since* τ_3 *is transient and* $\text{rank}(\tau_3) = \text{rank}(\Delta_3)$*, it is compatible with* Δ_3*. Since the right splitter of* τ_3 *is final, and so is the (empty) left splitter of* Δ_3 *the execution of* `Expand`(Δ_3) *terminates.*

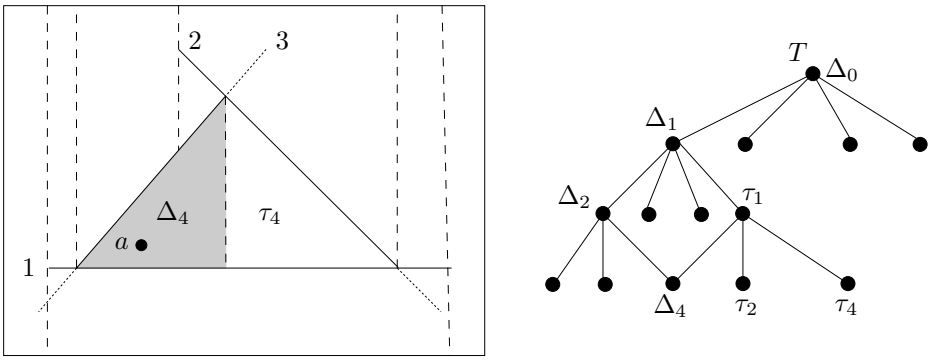


FIG. 16. *The trapezoids* Δ_3 *and* τ_3*, from the previous figure, are merged by* `CompZoneOnline` *to form the final trapezoid* Δ_4*. Since* $\mathrm{cl}(\Delta_4)$ *is not empty, we split it further by* $\mathrm{next}(\Delta_4) = 4$*.*

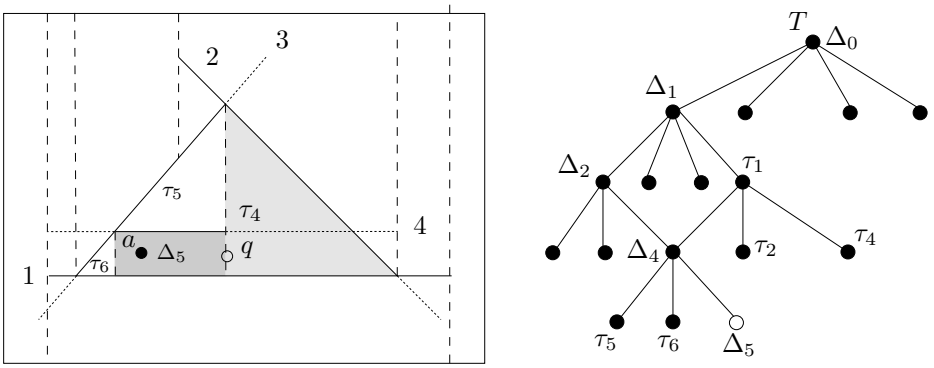


FIG. 17. *The splitting of* Δ_4 *creates three children* Δ_5, τ_5, τ_6*, of which* τ_5 *and* τ_6 *are final, whereas* Δ_5 *is transient and contains* a*.* `CompZoneOnline` *goes into* Δ_5*, and since it is transient,* `CompZoneOnline` *calls* `Expand`(Δ_5)*. A point-location query is performed at the midpoint* q *of the right splitter of* Δ_5*. This query traverses in* T *the path* $(\Delta_0, \Delta_1, \tau_1, \tau_4)$*. The (final) trapezoid* τ_4 *is not compatible with* Δ_5 *so it is further split by the segment* $\mathrm{next}(\tau_4) = 4$*.*

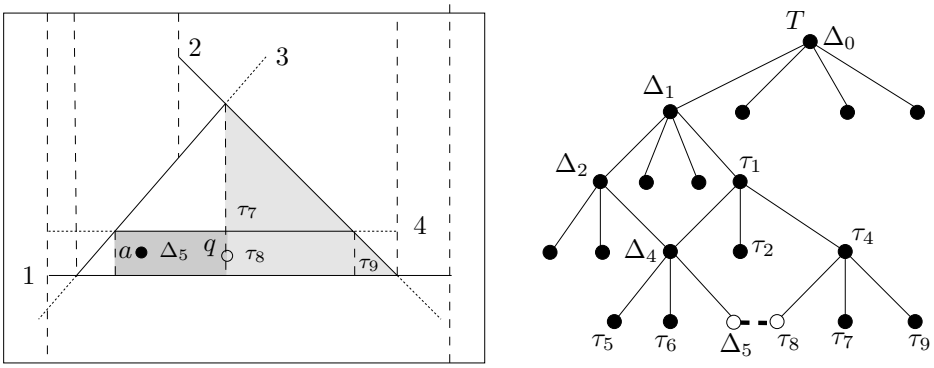


FIG. 18. τ_4 *is split into* τ_7, τ_8, τ_9*, of which only* τ_8 *is transient.* `CompZoneOnline` *goes into* τ_8*, which contains* q *on its left splitter. This trapezoid is compatible with* Δ_5*, and since both the left splitter of* Δ_5 *and the right splitter of* τ_8 *are final,* `Expand`(Δ_5) *terminates and merges both trapezoids.*

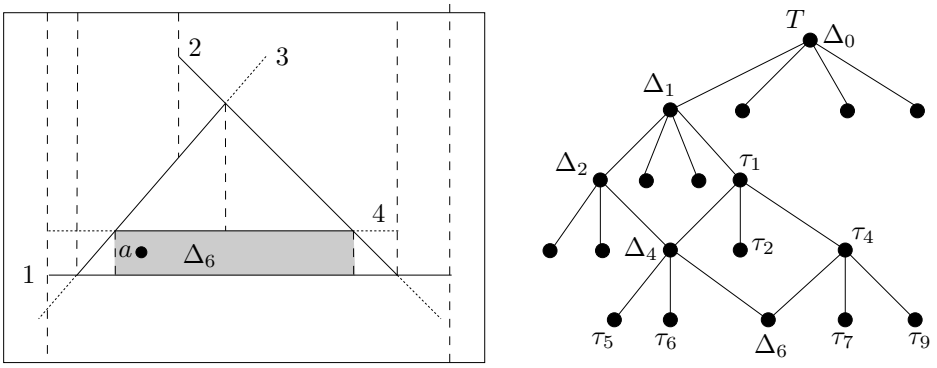


FIG. 19. *Voila! The newly formed final trapezoid Δ_6 contains our query point a, and its conflict list is empty. Thus, Δ_6 is the trapezoid of $\mathcal{A}_{\mathcal{VD}}(\hat{S})$ that contains a and is output as such by* `CompZoneOnline`.

Acknowledgments. The author wishes to thank Pankaj Agarwal, Danny Halperin, Chaim Linhart, and Micha Sharir for helpful discussions concerning the problems studied in this paper and related problems.

REFERENCES

[1] P. K. AGARWAL, B. ARONOV, T. CHAN, AND M. SHARIR, *On levels in arrangements of lines, segments, planes, and triangles*, Discrete Comput. Geom., 19 (1998), pp. 315–331.

[2] P. K. AGARWAL, M. DE BERG, J. MATOUŠEK, AND O. SCHWARZKOPF, *Constructing levels in arrangements and higher order Voronoi diagrams*, SIAM J. Comput., 27 (1998), pp. 654–667.

[3] Y. AHARONI, D. HALPERIN, I. HANNIEL, S. HAR-PELED, AND C. LINHART, *On-line zone construction in arrangements of lines in the plane*, in Proceedings of the 3rd International Workshop on Algorithm Engineering, London, 1999, Lecture Notes in Comput. Sci. 1668, Springer-Verlag, New York, pp. 139–153.

[4] K.-F. BÖHRINGER, B. DONALD, AND D. HALPERIN, *The area bisectors of a polygon and force equilibria in programmable vector fields*, Discrete Comput. Geom., 22 (1999), pp. 269–285.

[5] J.-D. BOISSONNAT AND M. YVINEC, *Algorithmic Geometry*, Cambridge University Press, Cambridge, UK, 1998.

[6] G. BRODAL AND R. JACOB, *Dynamic planar convex hull with optimal query time and $o(\log n \cdot \log\log n)$ update time*, in Proceedings of the 7th Scandinavian Workshop on Algorithm Theory, Lecture Notes in Comput. Sci. 1851, Springer-Verlag, Berlin, 2000, pp. 57–70.

[7] T. CHAN, *Dynamic planar convex hull operations in near-logarithmic amortized time*, in Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, pp. 92–99.

[8] T. CHAN, *Remarks on k-Level Algorithms in the Plane*, manuscript; also available from http://www.cs.uwaterloo.ca/~tmchan/pub.html, 1999.

[9] B. CHAZELLE, H. EDELSBRUNNER, L. GUIBAS, M. SHARIR, AND J. SNOEYINK, *Computing a face in an arrangement of line segments and related problems*, SIAM J. Comput., 22 (1993), pp. 1286–1302.

[10] K. L. CLARKSON AND P. W. SHOR, *Applications of random sampling in computational geometry* II, Discrete Comput. Geom., 4 (1989), pp. 387–421.

[11] M. DE BERG, K. DOBRINDT, AND O. SCHWARZKOPF, *On lazy randomized incremental construction*, Discrete Comput. Geom., 14 (1995), pp. 261–286.

[12] M. DE BERG, M. VAN KREVELD, M. H. OVERMARS, AND O. SCHWARZKOPF, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1997.

[13] T. K. DEY, *Improved bounds for planar k-sets and related problems*, Discrete Comput. Geom., 19 (1998), pp. 373–382.

[14] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[15] H. EDELSBRUNNER AND E. WELZL, *Constructing belts in two-dimensional arrangements with*

applications, SIAM J. Comput., 15 (1986), pp. 271–284.

[16] H. Everett, J.-M. Robert, and M. van Kreveld, *An optimal algorithm for the* ($\leq k$)*-levels, with applications to separation and transversal problems*, Internat. J. Comput. Geom. Appl., 6 (1996), pp. 247–261.

[17] S. Funke and K. Mehlhorn, *LOOK—a lazy object-oriented kernel for geometric computation*, in Proceedings of the 16th Annual ACM Symposium on Computational Geometry, Hong Kong, 2000, pp. 156–165.

[18] S. Har-Peled, *Constructing planar cuttings in theory and practice*, SIAM J. Comput., 29 (2000), pp. 2016–2039.

[19] S. Har-Peled, *Taking a Walk in a Planar Arrangement—the Implementation*, manuscript; also available from http://www.math.tau.ac.il/~sariel/papers/99/qwalk.html, 2000.

[20] S. Har-Peled and P. Indyk, *When crossings count—approximating the minimum spanning tree*, in Proceedings of the 16th Annual ACM Symposium on Computational Geometry, Hong Kong, 2000, pp. 166–175.

[21] S. Har-Peled and M. Sharir, *On-Line Point Location in Planar Arrangements and Its Applications*, manuscript, 1999.

[22] J. Matoušek, *On constants for cuttings in the plane*, Discrete Comput. Geom., 20 (1998), pp. 427–448.

[23] K. Mehlhorn and S. Näher, *LEDA: A platform for combinatorial and geometric computing*, Comm. ACM, 38 (1995), pp. 96–102.

[24] K. Mulmuley, *Computational Geometry: An Introduction through Randomized Algorithms*, Prentice–Hall, Englewood Cliffs, NJ, 1994.

[25] M. H. Overmars and J. van Leeuwen, *Maintenance of configurations in the plane*, J. Comput. System Sci., 23 (1981), pp. 166–204.

[26] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[27] H. Tamaki and T. Tokuyama, *A characterization of planar graphs by pseudo-line arrangements*, in Proceedings of the 8th Annual International Symposium on Algorithms and Computation, Lecture Notes in Comput. Sci. 1350, Springer-Verlag, New York, 1997, pp. 133–142.

[28] H. Tamaki and T. Tokuyama, *How to cut pseudo-parabolas into segments*, Discrete Comput. Geom., 19 (1998), pp. 265–290.

[29] G. Tóth, *Point sets with many k-sets*, in Proceedings of the 16th Annual ACM Symposium on Computational Geometry, Hong Kong, 2000, pp. 37–42.

SIAM J. COMPUT.
Vol. 30, No. 4, pp. 1368–1383

FINDING SETS COVERING A POINT WITH APPLICATION TO MESH-FREE GALERKIN METHODS*

XIAOXU HAN†, SUELY OLIVEIRA‡, AND DAVID STEWART†

Abstract. For a collection of sets in $\mathbf{R}^d$ we consider the task of finding all sets in the collection that cover or contain a given point. The algorithms introduced in this paper are based on quadtrees and their generalizations to $\mathbf{R}^d$. The advantages of our new splitting algorithm to find the covering sets of a point over the basic algorithm are detailed by means of hit rates and the expected depth traversed in the quadtree search, numerically and theoretically. This solves a difficult problem faced by mesh-free discretizations of partial differential equations.

Key words. quadtree and octree search methods, computational geometry

AMS subject classifications. Primary: 68U05, 65D18; Secondary: 65N30

PII. S0097539799359361

1. Introduction.

The task of this paper is, given a collection of support sets $S_i \subset \mathbf{R}^d$, $i = 1, 2, \ldots, N$, to efficiently answer the query "*Given a point $x \in \mathbf{R}^d$, find all S_i such that $x \in S_i$.*" This task arises when assembling the matrix for a mesh-free discretization of a partial differential equation. The sets S_i are the support sets of the basis functions used to solve the partial differential equation. They are typically rectangles with modest aspect ratios, or circles/spheres in $\mathbf{R}^d$ with d being 2, 3, or 4. An example of such a problem is illustrated in Figure 1.

One way to answer this query quickly is to organize the S_i using quadtrees and octrees [5, 9, 13]. Clearly the time needed to answer this query is at least $\Omega(\#\{\, i \mid x \in S_i \,\})$ where $\#A$ is the cardinality of the set A. The quantity $\#\{\, i \mid x \in S_i \,\}$ is the covering number of x; it is the number of support sets containing (that is, covering) the point x. Quadtrees and octrees have been found useful in geometric tasks such as finding which points from a collection belong to a given *range* [3]. For a task involving preprocessing a collection of points, quadtrees and octrees can be considered as *tries,* such as are used for string matching [2]. However, the task studied in this paper involves preprocessing a collection of *sets* $\{\, S_i \subset \mathbf{R}^d \mid i = 1, 2, \ldots, N \,\}$ rather than a collection of points. Also note that this task is not the point location problem previously studied [11, 12], in that here the sets S_i are usually not disjoint. Our analysis is also different to that of [11, 12] in that here we are concerned with the expected time for a random query point x, rather than with worst-case analysis. While the data structures described here are those of Samet [15], Samet considers their use only for the intersection problem: find all pairs (i, j) where $S_i \cap S_j \neq \emptyset$. Here we wish, given x, to find all i where $x \in S_i$.

For each point x and support set S_i, we assume that we can test if $x \in S_i$ in $\Theta(1)$ time. The point about using quadtrees/octrees or other similar data structures is that we can avoid doing unnecessary tests. The total time to answer a query can be

*Received by the editors July 26, 1999; accepted for publication (in revised form) July 10, 2000; published electronically October 31, 2000. This research was supported by NSF/DARPA grant DMS-9874015.

http://www.siam.org/journals/sicomp/30-4/35936.html

†Department of Mathematics, University of Iowa, Iowa City, IA 52242 (dstewart@math.uiowa.edu).

‡Department of Computer Science and Department of Mathematics, University of Iowa, Iowa City, IA 52242 (oliveira@cs.uiowa.edu).

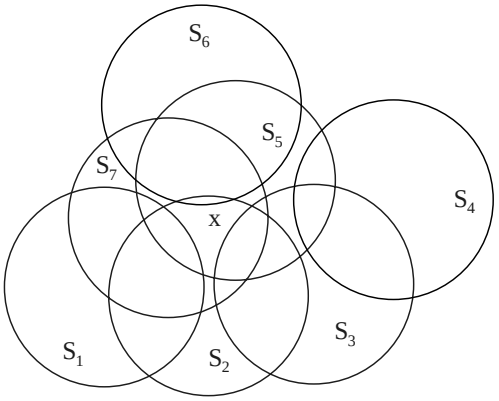


FIG. 1. *Example of distribution of support sets S_i.*

roughly divided up into the time needed to traverse the data structure plus the time to perform the tests to identify the support sets containing x. Since the best we can do is to have a "hit" (i.e., $x \in S_i$) whenever we test for "$x \in S_i$," we are interested in the "hit rate," which is the ratio of hits to the total number of tests, averaged over all the integration points. In particular, we don't want this to go to zero; and if the hit rate can be bounded away from zero, then the number of tests is, at worst, proportional to the average covering number of the sets $\{ S_i \mid i = 1, 2, \ldots, N \}$. We will show that the average time to traverse the quadtree is no worse than $O((\log N)/d)$ for both algorithms that we will present, so that the traversal time is a more easily controlled cost.

Worst-case deterministic bounds for the hit rate can be arbitrarily bad: the hit rate is zero if x is not covered by any support set and at least one test is done for x. If there are no restrictions on the shape of the support sets, then quadtrees/octrees can be made to perform badly, for example, by making the S_i's curves or thin ribbons. In practice, the support sets S_i are discs, balls, rectangles, rectangular solids, or similar shapes, with modest aspect ratios (the ratio between the longest and smallest dimension).

Our bounds on the hit rate are proven under the assumption that x is chosen randomly and uniformly over a set S which contains the union of the S_i's.

1.1. Application. Various new methods for solving partial differential equations have been introduced which do not require construction of grids or triangulations of regions in order to compute approximate solutions. These are known as mesh-free methods; there are a number of types of mesh-free methods such as *partition of unity finite element methods* (PUFEM), *reproducing kernel particle methods* (RKPM), *element free Galerkin* (EFG) methods, and *moving least squares* (MLS) methods. An overview of mesh-free methods in general can be found in [4].

All of these methods construct basis functions Ψ_i, $i = 1, 2, \ldots, N$, over the domain of interest $\Omega \subset \mathbf{R}^d$ from which a Galerkin discretization is developed. Thus, for example, for solving the partial differential equation

$$\begin{aligned} -\nabla^2 u(x) + u(x) &= f(x), \quad x \in \Omega, \\ u(x) &= 0, \quad x \in \partial\Omega, \end{aligned}$$

where $\partial\Omega$ is the boundary of Ω, we approximate $u \approx u^h$ where $u^h(x) = \sum_{i=1}^{N} u_i^h \Psi_i(x)$.

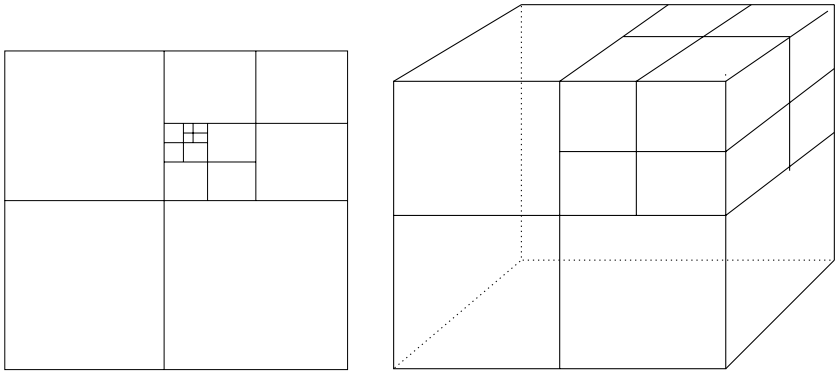


FIG. 2. *Quadtree and octree examples.*

Then we construct a linear system of equations for the unknown coefficients u_i^h: $\sum_{j=1}^{N} K_{ij} u_j^h = f_i$ for $i = 1, 2, \ldots, N$ where

$$K_{ij} = \int_\Omega \frac{1}{2} \left(\nabla \Psi_i(x)^T \nabla \Psi_j(x) + \Psi_i(x) \Psi_j(x) \right) \, dV(x).$$

For most mesh-free methods, either there are no analytical formulas available for $\Psi_i(x)$ or they are too complex to compute these integrals directly. Therefore numerical approximations of the form

$$K_{ij} \approx \sum_{k=1}^{P} \frac{1}{2} \left(\nabla \Psi_i(x_k)^T \nabla \Psi_j(x_k) + \Psi_i(x_k) \Psi_j(x_k) \right) \omega_k$$

are used to construct the linear system. In our notation, P is the number of integration points. Let $S_i = \operatorname{supp} \Psi_i = \overline{\{ z \mid \Psi_i(z) \neq 0 \}}$, the support of Ψ_i. Note that Ψ_i and all of its derivatives are zero outside its support S_i. The integration points x_k are scattered essentially uniformly throughout the region Ω. Rather than first looping over i and j where $S_i \cap S_j \neq \emptyset$ and then finding all k such that $x_k \in S_i \cap S_j$, we can compute the matrix K more efficiently by looping first over k and then finding all i where $x_k \in S_i$ by using quadtree/octree-type data structures. Note that the former strategy leads to both an intersection-finding problem and a range-searching problem. While finding all i and j where $S_i \cap S_j \neq \emptyset$ can be solved using quadtree structures (see, for example, Samet [15, pp. 199–225]), the costs are at least as high as for the task discussed here. The problem of finding all x_k where $x_k \in S_i \cap S_j$ is a range-searching problem which requires construction of a range-searching tree on the integration points. Apart from the fact that the number of integration points is usually much larger than the number of support sets in practice, these structures are complex and the searching time complexity is $O(\log^{d-1} P + \#\{ k \mid x_k \in S_i \cap S_j \})$ in d dimensions where P is the number of integration points. By contrast, the latter strategy can be accomplished with methods which have an expected time complexity of just $O(\log N + \mathbf{E}(\#\{ i \mid x_k \in S_i \}))$ in any fixed number of dimensions.

1.2. Quadtrees and octrees. Quadtrees and octrees are members of hierarchical decomposition data structures based on the principle of recursive decomposition of space. Examples of quadtrees and octrees are illustrated in Figure 2. They are widely used in image compression [14, 13], GIS (Geographic Information Systems)

[16, 6], computer vision [7], and computer graphics [8, 17]. Quadtrees can be used to represent a two-dimensional space. Similarly, octrees can be used to deal with the three-dimensional case. A quadtree is a tree data structure based on a rectangle with four children and an octree is a tree data structure based on a cube with eight children. However, both of them can be used in the generation of meshes for finite element analysis. Due to different applications, there are variants on this structure, such as region-based quadtrees and point quadtrees.

A quadtree can be described in this way using Java, for example:

```
class QtreeNode{
   private Object data;
   private QtreeNode children[2][2];
   private QtreeNode parent;
   private int depth;
   private int coords[2];
     ...
   }
```

An octree tree can be described in a similar way using Java:

```
class OctreeNode{
   private Object data;
   private OctreeNode parent;
   private OctreeNode children[2][2][2];
   private int depth;
   private int coords[3];
     ...
   }
```

We use the following notation and terminology to describe various aspects of quadtrees, octrees, and their d-dimensional analogues.

- A *support set* S_i is simply a bounded subset of $\mathbf{R}^d$; the quadtree or octree construction algorithm is, given a collection $\{S_i \mid i = 1, 2, \ldots, N\}$ of these sets, to build a corresponding quadtree or octree representation of these support sets S_i.
- A *rectangle* is a set of the form $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$ in $\mathbf{R}^d$; a *rectangle in a quadtree or octree* has the special property that $a_i = \gamma_i 2^{-k}$, $b_i = (\gamma_i + 1)2^{-k}$ for all i with k independent of i, and γ_i an integer. A *subrectangle* of a rectangle R in a quadtree or octree is a rectangle in the quadtree or octree contained in the rectangle R.
- We use *quadtree* to refer to quadtrees and their generalizations to 2^d-trees in $\mathbf{R}^d$. The term thus includes octrees in three dimensions and binary trees in one dimension.
- The *width* of a set A is the diameter of the set in the infinity norm; that is, $\text{width}(A) = \sup\{\, \|x - y\|_\infty \mid x, y \in A \,\} = \sup_{x,y \in A} \max_i |x_i - y_i|$.
- The *top* rectangle is the rectangle at the root of the quadtree or octree. It is usually considered to be the rectangle $[0, 1] \times [0, 1] \times \cdots \times [0, 1]$.
- The *expected value* of a random variable X is denoted $\mathbf{E}(X)$.
- The *d-dimensional volume*, or *d-volume* of a set S is denoted $\text{vol}_d(S)$.

Each rectangle in the quadtree has an associated linked list containing references to support sets S_i; if R is a rectangle in the quadtree whose linked list contains a reference to S_i, we say that S_i *is linked to* R, or that R *links* S_i.

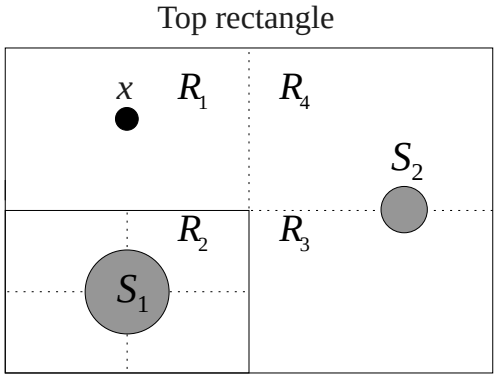


FIG. 3. *Example of the basic quadtree.*

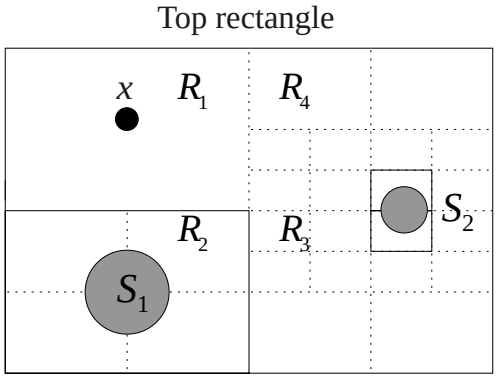


FIG. 4. *Example of the expanded quadtree.*

2. Algorithms. In this section we present two quadtree search algorithms to find the number of support sets covering each point x. In our application, x is an integration point and is part of a set of integration points x_k that are approximately uniformly distributed in a domain Ω. The first algorithm for building quadtree structures is the called the *basic algorithm*, and the second is called the *splitting algorithm*. These data structures constructed are discussed in Samet [15], where the former is called the MX-CIF quadtree and the latter the expanded MX-CIF quadtree. In this paper they will be referred to as the basic and expanded quadtrees, respectively. However, neither data structure was considered in Samet [15] in the context of identifying sets containing a given query point. The splitting algorithm has advantages over the basic algorithm in finding the number of support sets covering the integration point x. The total expected query time for randomly selected points x can be bounded in terms of the *hit rate* and the *expected depth traversed* in the quadtree. We will provide detailed theoretical analysis and numerical simulation in terms of these quantities later. We illustrate the basic and expanded quadtrees in Figures 3 and 4.

In both the basic algorithm and the splitting algorithm, each rectangle in the quadtree has a corresponding linked list. In the basic approach we try to link each S_i to a rectangle R in the quadtree if it is contained in R and there is no subrectangle of R in which S_i is completely contained. Note that S_i is not necessarily linked to a leaf node of the quadtree, even at the time of linking. Also note that there can be

more than one support set linked to a single rectangle R, creating a linked list. In the interests of efficiency, we usually want these linked lists to be short. When using this algorithm, in the worst case, the construction of the quadtree takes $O(N \log(1/\varepsilon))$ time where $\varepsilon = \min_i \operatorname{diam} S_i$.

The basic algorithm constructs a quadtree which is the MX-CIF quadtree discussed by Samet [15, pp. 200–213] and the data structure discussed by Abel and Smith [1]. Unlike Samet, however, we are interested here in finding sets S_i containing query points, rather than intersection detection.

PSEUDOCODE FOR BUILDING THE BASIC QUADTREE (BASIC ALGORITHM).

```
void basic(S,top,N)
/* Constructing a quadtree for support sets S_i. */
for i = 1, 2, ..., N
    R ← top
    repeat
        for each subrectangle R' of R
            if S_i ⊆ R'
              R ← R' & break for loop
            end if
        end for
    until (no subrectangle of R contains S_i)
    link S_i to the rectangle R
end for
```

For example, in Figure 3, S_1 is not linked to the top rectangle R but it is linked to the subrectangle R_2; since S_2 is not contained in any of the subrectangles of R, it is linked to R.

In our splitting algorithm (our second method), if a support set S_i is a subset of a rectangle R but is not a subset of any subrectangle of R in the quadtree and if $\operatorname{diam}(S_i) < \frac{1}{2}\operatorname{width}(R)$, then we don't link S_i to R; rather we "split" S_i for each subrectangle R' of R, if $S_i \cap R' \neq \emptyset$, and recursively call the splitting algorithm with $S_i \cap R'$. We give the pseudocode for the splitting algorithm below. Note that the splitting algorithm builds a recursive extension of the MX-CIF tree of Samet which is the expanded MX-CIF tree of Samet [15, pp. 213–215].

PSEUDOCODE FOR BUILDING THE EXPANDED QUADTREE (SPLITTING ALGORITHM).

```
procedure splitting(S_i,R)
/* Add support set S_i to quadtree
   using R as the root of the quadtree */
loop
   if R' is a subrectangle of R where S_i ⊂ R'
      R ← R'
    else if diam(S_i) < (1/2)width (R)
        for each subrectangle R' of R where R' ∩
        S_i ≠ ∅ (allocate R' if necessary)
          call splitting(S_i ∩ R',R')
   else
     link S_i to R and exit loop
end loop
```

The splitting algorithm is illustrated by Figure 5. The subscripts in this figure refer to the subrectangle number, being 1 (northwest), 2 (southwest), 3 (southeast), or 4 (northeast). In this figure $R^{j+p}_{k,l,\ldots}$ denotes the rectangle at level $j + p$, where

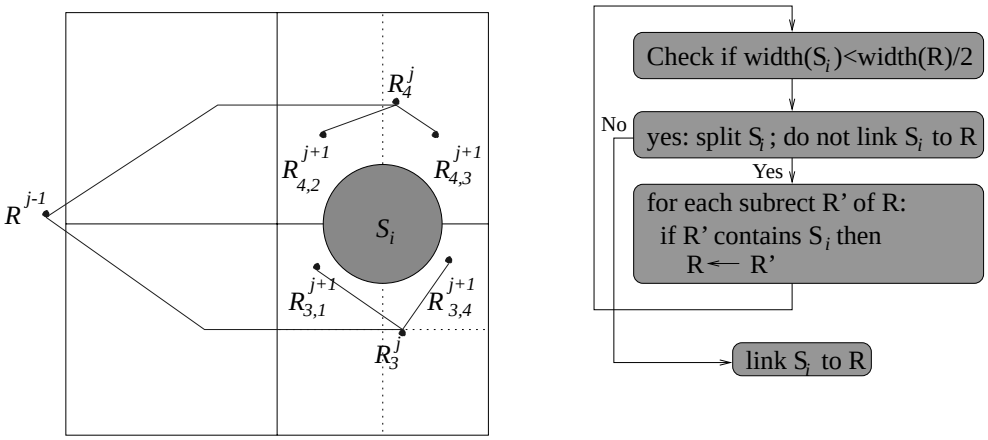


FIG. 5. *Example for describing the splitting algorithm.*

$k, l, \ldots$ denotes the path from R^{j-1} to this rectangle within the quadtree.

To construct the complete quadtree, we apply `splitting` to all support sets, i.e.,

```
for i = 1, 2, ..., N
    call splitting(S_i, top)
```

Note that in the expanded quadtree, each S_i can be linked to more than one rectangle in the quadtree. In fact, each S_i can be linked to at most 2^d rectangles. As given, the time and space requirements for the splitting algorithm are $O(2^d N\, h)$, where h is the depth (or height) of the expanded quadtree. Since splitting can produce very small sets in the computation of $R' \cap S_i$, the worst-case space and time requirements are unbounded. However, if *path compression* [2] is used so that rectangles with only one allocated subrectangle and no S_i linked to it are removed from the quadtree, the worst-case space requirements reduce to $O(2^d N)$. Furthermore, the depth of the path-compressed quadtree is bounded above by $c_{\max} + \lceil \log_2 N \rceil$, where $c_{\max} = \max_x \#\{\, i \mid x \in S_i \,\}$ is the maximum covering number. In applications, the maximum covering number is bounded, or grows polylogarithmically in N.

The path-compressed quadtree can be built incrementally using a number of different techniques. A path-compressed version of the basic quadtree can start by finding the depth and coordinates of the smallest enclosing quadtree rectangle. For each S_i we find a bounding box $[a_1, b_1] \times \cdots \times [a_d, b_d] \subseteq [0,1]^d$ and expand each coordinate a_i and b_i in binary. The number of leading bits of a_i and b_i that match is the depth of the smallest enclosing quadtree rectangle, and the matching bits are the (integer) coordinates of that rectangle. Using the Java data structures above, the current path-compressed quadtree can be traversed downwards until a rectangle is found at the correct depth, or the correct place to insert a new such rectangle can be found. Once this is found, standard techniques can insert a suitable rectangle in $O(1)$ time. A similar approach can be used to find the $\leq 2^d$ rectangles which are linked to a support set for the splitting algorithm by first finding matching bits. This makes the time to construct the quadtree for the splitting algorithm with path compression $O(2^d N(c_{\max} + \log N + d \log\log(1/\varepsilon)))$, where $\varepsilon = \min_i \operatorname{diam}(S_i)$ assuming $O(1)$ time bit operations (and, or, invert, and shift) on integers. Alternatives to pointer-based data structures include balanced trees and hash-tables using the tuple (depth, coord_1,..., coord_d) as the key, where coord_i is the ith coordinate of the rectangle.

Alternatively, if in the splitting algorithm the set $R' \cap S_i$ is expanded to have diameter no less than $0 < \eta \leq 1/2$ times the diameter of S_i, the maximum depth of the expanded quadtree is no more than $\log_2(1/\varepsilon) + \log_2(1/\eta)$. Choosing a particular value (e.g., $\eta = 1/4$) gives an upper bound on the depth of the expanded quadtree generated by the modified splitting algorithm with the same asymptotic behavior. Furthermore, the performance bounds for the search given below remain unaltered for the modified splitting algorithm.

Once the quadtree is constructed, we need to use it to find the support sets covering a given point x. Below is our pseudocode for doing this with a quadtree constructed using either the basic or splitting algorithms.

```
list function search(x,top)
/* Given x, find all S_i that cover x: x ∈ S_i */
R ← top
while R ≠ null
    for each S_i linked to R
    if x ∈ S_i then
        add S_i to list of covering support sets
        increment covering number
    R ←allocated subrectangle of R containing x, if any
    R ← null if there is no such subrectangle
end while
```

In Figure 3, we show the behavior of the basic algorithm. We need to make a test to determine if x is in S_2 or not because S_2 is linked to the top rectangle, though x is far from S_2.

In Figure 4, the advantages of the splitting algorithm are exemplified. We will test if a given $x \in S_2$ only if it is in the two smallest rectangles shown containing S_2.

2.1. The advantage of the expanded quadtree over the basic quadtree. Just as we discussed above, in the basic quadtree there are many unnecessary tests to obtain the sets covering each integration point. Here, we give the plot of the number of wasted tests (unnecessary tests) per integration point in two algorithms in Figure 6. Here, we randomly generate 2,000 support sets (circles) with radius = 0.2 and use 10,000 uniformly distributed integration points.

We observe that the number of wasted tests in the basic algorithm is 15–20 times the number in the expanded quadtree.

Figure 7 gives plots of the lengths of the linked lists in the basic and expanded quadtrees. We can see, for the chosen 200 support sets, that the linked lists are generally much shorter for the expanded quadtree, so that fewer tests are needed for each rectangle. Furthermore, the few rectangles with long linked lists in the basic quadtree are the top rectangles, and the rectangles near the top. All support sets linked to the top rectangle are tested for *any* integration point, making this data structure much less efficient. The expanded quadtree also links support sets deeper in the quadtree, as shown in Figure 8.

2.2. The hit rate. The hit rate is the ratio between the average number of sets covering integration point x and the average number of tests for finding all covering sets for all integration points x, where x is a point randomly chosen from a uniform distribution of points over the top rectangle.

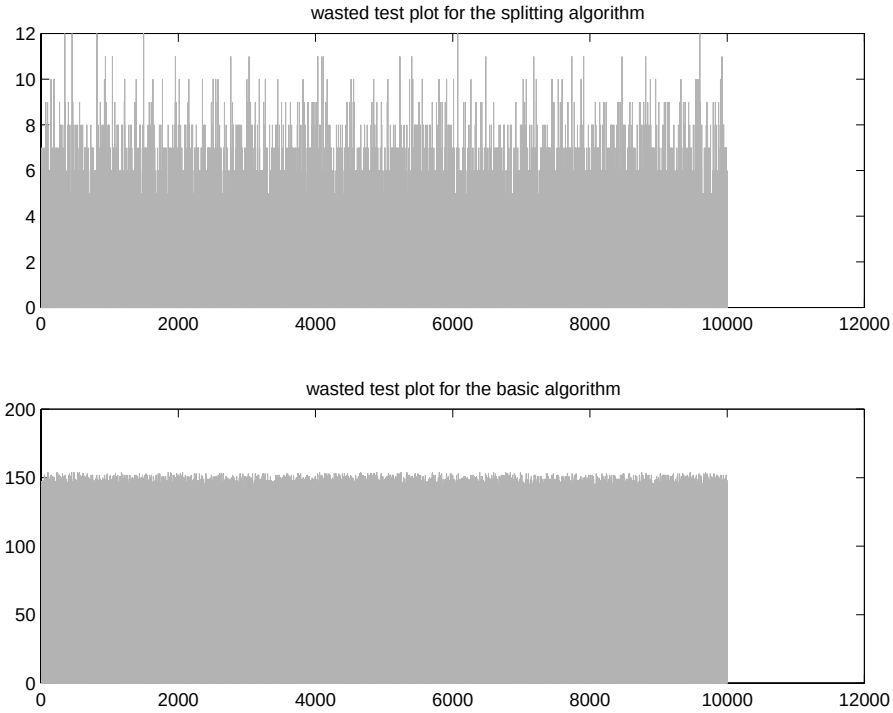


FIG. 6. *The number of wasted tests in the basic and expanded quadtrees.*

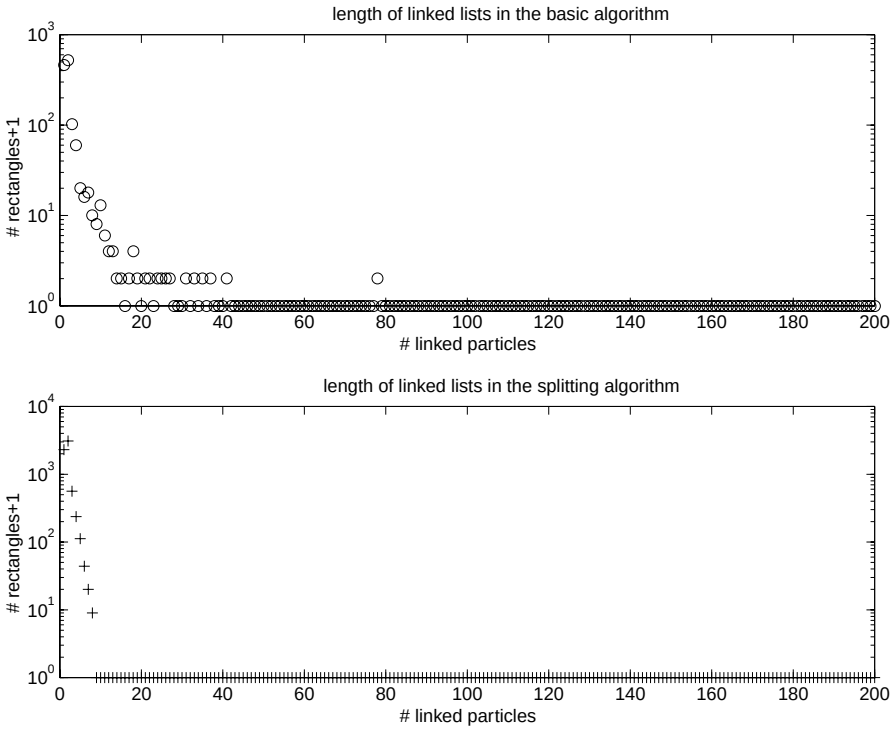


FIG. 7. *The length of linked lists in the basic and expanded quadtrees.*

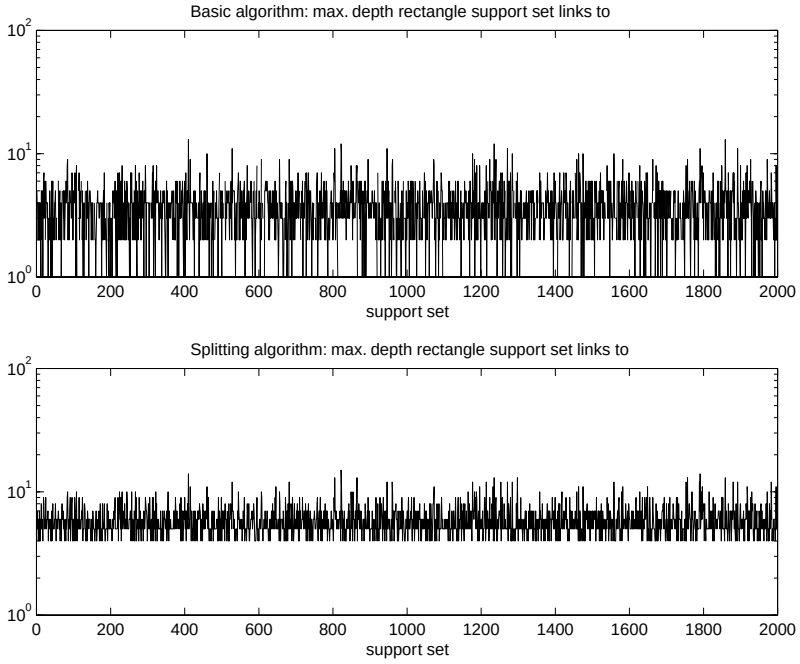


FIG. 8. *Depth of rectangle linked to each support set.*

We give a more precise definition of the hit rate:

$$\text{hit rate} = \frac{\mathbf{E}(covering\#)}{\mathbf{E}(\#tests)}.$$

Here *covering#* is the number of sets covering the randomly chosen point x, while $\#tests$ is the number of tests needed in the algorithms to determine the list of sets covering x. That is, hit rate is the ratio between the mathematical expectation of the covering (support sets) numbers of the integration points and the expectation of the number of tests to get the coverings for the randomly chosen and uniformly distributed integration points.

3. Analysis of the expected search time. We randomly choose integration points x and go down through the quadtree to find the sets S_i which contains x in our integration region. The total search time will depend on how many tests we will do for the integration points we use, and how deeply we traverse the tree structure. So our expected search time is

$$\mathbf{E}(\text{search time}) = O(\mathbf{E}(depth) + \mathbf{E}(\#tests)),$$

where *depth* is the depth traversed in the quadtree when searching for sets covering a given point x, and $\#tests$ is the number of tests $x \in S_i$ that are performed during the search. Since $\mathbf{E}(\#tests) = (\text{hit rate}) \times \mathbf{E}(covering\#)$, provided the hit rate is bounded away from zero, we get $O(\mathbf{E}(depth) + \mathbf{E}(covering\#))$ expected search time per query point.

We will show that, for both the splitting and basic algorithms, the expected depth is bounded above by $(\log N)/d$; i.e, $\mathbf{E}(depth) = O((\log N)/d)$. For the basic quadtree,

the hit rate can go to zero. In fact, for the basic quadtree the hit rate is $O(N^{-1/2})$ in two dimensions in practice. The expanded quadtree, however, has a hit rate that is bounded away from zero independently of N, although it can decrease exponentially as the dimension d increases. In the following sections we will analyze both the hit rate and the expected depth traversed in the quadtree.

3.1. Analysis of the hit rate. We assume bounds on the *aspect ratio*, which is the ratio of the longest to the shortest side. First, we give the lower and upper bounds of $\mathbf{E}(covering\#)$, then we get the inequality of reflecting relationship between $\mathbf{E}(covering\#)$ and $\mathbf{E}(\#tests)$. We combine these to bound the hit rate.

3.1.1. The hit rate for the basic algorithm. First, we note that the *hit rate* can be arbitrarily bad for the basic quadtree. In two dimensions, if every support set S_i was chosen to intersect either the horizontal or vertical lines bisecting the *top* rectangle, then every support set would be linked to the *top* rectangle of the quadtree. That would mean that all the support sets would have to be tested for all integration points $x \in top$. This would give a hit rate of $\mathbf{E}(covering\#)/N$, which can be arbitrarily close to zero, and goes to zero if the average covering number is bounded as $N \to \infty$.

Even for uniform or random distributions of the sets S_i, if $\mathrm{diam}(S_i)$ are all near $\varepsilon > 0$, the number of sets linked to the *top* rectangle is $\Theta(N\varepsilon)$; if $\varepsilon = \Omega(N^{-1/d})$ as we would expect if the S_i's covered *top*, then the number of sets linked to the *top* rectangle would be $\Omega(N^{1-(1/d)})$. While this is an improvement over the naive approach of directly testing a point x against all sets S_i, it is far from optimal. This confirms the $O(\sqrt{N})$ query time found in [1] for planar queries.

3.1.2. Using the number of links to express $\mathbf{E}(\#test)$ for the expanded quadtree. We work in d dimensions ($d = 2$ for quadtrees, $d = 3$ for octrees). Let $N_j = \#\text{links at level } j$. Also, let $\widehat{N}_j$ be the number of rectangles which have at least one support set linked at level j. Note that $\widehat{N}_j \le N_j$, and every support set has at most 2^d links to it. So, for all support sets N for the integration point x , we have

$$\sum_{j=0}^{\infty} N_j \le 2^d N.$$

Every rectangle at level j has d-volume 2^{-jd} times the volume of the *top* rectangle and each link from a rectangle requires a test for any point contained in that rectangle. Suppose that the rectangles at level j which are linked to a support set are $R_{j,1}, R_{j,2}, \ldots, R_{j,\widehat{N}_j}$. Since there are N_j links at level j, then the expected number of links to search from level j is

$$\begin{aligned}\mathbf{E}(\text{tests at level } j) &= \sum_{k=1}^{\widehat{N}_j} \Pr(x \in R_{j,k}) \times (\#\text{support sets linked to } R_{j,k}) \\ &= \sum_{k=1}^{\widehat{N}_j} 2^{-jd}(\#\text{support sets linked to } R_{j,k}) = 2^{-jd} N_j.\end{aligned}$$

So, $\mathbf{E}(\#tests)$ is the sum of the expected number of links in all levels: $\mathbf{E}(\#tests) = \sum_{j=0}^{\infty} 2^{-jd} N_j$.

3.1.3. Bounds on E(*covering#*) for the expanded quadtree. Adding total volumes of all support sets S_i, $i = 1, 2, \ldots, N$, in d dimensions, we get the value of $\mathbf{E}(covering\#) = \mathbf{E}(\#\{\, i \mid x \in S_i \,\})$. Let a be the maximum *aspect ratio (the ratio of the largest side to the shortest side for rectangles)* of the support sets. Assume that $\mathrm{vol}_d(top) = 1$. Then

$$\begin{aligned}\mathbf{E}(covering\#) &= \sum_{i=1}^{N} \mathrm{vol}_d(S_i) \geq \sum_{i=1}^{N} \frac{1}{a^{d-1}} \mathrm{width}(S_i)^d \\ &= \frac{1}{a^{d-1}} \sum_{i=1}^{N} \mathrm{width}(S_i)^d.\end{aligned}$$

We suppose that, at level j, S_i has n_{ij} links. Then the total links of S_i in the whole tree is less than 2^d:

$$\sum_{j=0}^{\infty} n_{ij} \leq 2^d \text{ for all } i.$$

On the other hand, the total number of links in level j for all supports of our integration points are $N_j = \sum_i n_{ij}$. Remember that for the expanded quadtree a support set S_i is not linked at level j if the width of S_i is no more than half the width of a rectangle at level j. That is,

$$n_{ij} = 0 \text{ if } \mathrm{width}(S_i) \leq 2^{-j-1} \mathrm{width}(top).$$

Since in the expanded quadtree, the volume of S_i is split into disjoint parts, if we add all volumes in each linked rectangle for S_i, we have the following bound:

$$\mathrm{vol}_d(S_i) \leq \sum_{j=0}^{\infty} n_{ij} 2^{-jd} \mathrm{vol}_d(top).$$

Assuming $\mathrm{vol}_d(top) = 1$, we have

$$\mathrm{vol}_d(S_i) \leq \sum_{j=0}^{\infty} n_{ij} 2^{-jd}.$$

An upper bound for $\mathbf{E}(covering\#)$ is easily obtained:

$$\begin{aligned}\mathbf{E}(covering\#) &\leq \sum_{i=1}^{N} \sum_{j=0}^{\infty} n_{ij} 2^{-jd} = \sum_{j=0}^{\infty} \left(\sum_{i=1}^{N} n_{ij} \right) 2^{-jd} \\ &= \sum_{j=0}^{\infty} N_j 2^{-jd}.\end{aligned}$$

However, what we really need is a lower bound. For each set S_i let $j_i^* = \min\{\, j \mid S_i \text{ is linked to a rectangle } R \text{ at level } j \,\}$. For at least one rectangle R on level j_i^*, the aspect ratio of $S_i \cap R$ is no more than $2a$. Then $\mathrm{vol}_d(S_i) \geq \mathrm{vol}_d(R \cap S_i) \geq (2a)^{1-d} \mathrm{width}(R \cap S_i)^d \geq (2a)^{1-d} 2^{-d} \mathrm{width}(S_i)^d \geq 2a(4a)^{-d} 2^{-(j_i^*+1)d}$. Since $\sum_{j=0}^{\infty} n_{ij} \leq 2^d$ and $n_{ij} = 0$ for $j < j_i^*$, $\mathrm{vol}_d(S_i) \geq \sum_{j=0}^{\infty} 2a(4a)^{-d} 2^{-(j+1)d} \times 2^{-d} n_{ij}$.

Then

$$\mathrm{vol}_d(S_i) \geq 2a(16a)^{-d} \sum_{j=0}^{\infty} n_{ij} 2^{-jd}.$$

So, we can get a lower bound of $\mathbf{E}(covering\#)$ as follows:

$$\begin{aligned}\mathbf{E}(covering\#) &= \sum_{i=1}^{N} \mathrm{vol}_d(S_i) \geq 2a(16a)^{-d} \sum_{j=0}^{\infty} \sum_{i=1}^{N} n_{ij} 2^{-jd} \\ &= 2a(16a)^{-d} \sum_{j=0}^{\infty} N_j 2^{-jd} = 2a(16a)^{-d} \mathbf{E}(\#tests).\end{aligned}$$

Thus the hit rate for the expanded quadtree is bounded away from zero for fixed d.

3.1.4. The lower bound of the average hit rate for the expanded quadtree. By the definition of the hit rate, we have lower bound for average hit rate:

$$\text{hit rate} = \frac{\mathbf{E}(covering\#)}{\mathbf{E}(\#tests)} \geq 2a(16a)^{-d}.$$

From here, we can see it is impossible to get zero hit rate in our quadtree search. That is to say, if each set S_i is a square in two dimensions, for each set S_i covering x, there will be an average of at most 128 tests using the splitting algorithm. However, this bound is not sharp: the hit rate is usually much larger, as can be seen in Figure 9 and Table 1.

3.2. The simulation results on hit rate. From the point of view of the *hit rate*, we can also demonstrate the advantage of the expanded over the basic quadtree. For 10,000 randomly chosen and uniformly distributed integration points and 2,000 support sets with radius 0.2, we get the following results. The average hit rate in the expanded quadtree is almost 30 times larger than that of in the basic case: *expanded quadtree:* 0.1694; *basic quadtree:* 0.0057. This result agrees with the result of the wasted tests counted in the two algorithms. Note that in this simulation, many of the support sets overlap, which can cause difficulties for other data structures.

Figure 9 shows the hit rate plots for the basic and expanded quadtrees.

Computational results were also obtained for actual mesh-free methods. In particular, for a two-dimensional elasto-plastic problem computations were done using a number of different levels of refinement. The particular mesh-free method used was the RKPM (reproducing kernel particle method) [10]. The basis functions were tensor products of B-splines (that is, $\Psi_i(x) = B_{i,1}(x_1)B_{i,2}(x_2)$, where $B_{i,1}$ and $B_{i,2}$ are cubic splines), so that the support sets were axis-aligned rectangles. The aspect ratios of the support sets were all similar and not extreme. However, each support set was relatively large, resulting in a great deal of overlap; the average covering number is around 15. Table 1 shows the main results. Note that "$\#R$" is the number of rectangles allocated for the quadtree, N is the number of sets, P is the number of integration (i.e., query) points, and h is the depth (or height) of the quadtree. The naive algorithm tests each point against each support set. This example does not show the full power of the data structures since the problems are relatively small and two-dimensional. However, the difference between the basic and expanded quadtrees is clearly evident. The average covering numbers for the problems are shown in Table 2. As expected, the basic quadtree's hit rate is roughly proportional to $N^{-1/2}$,

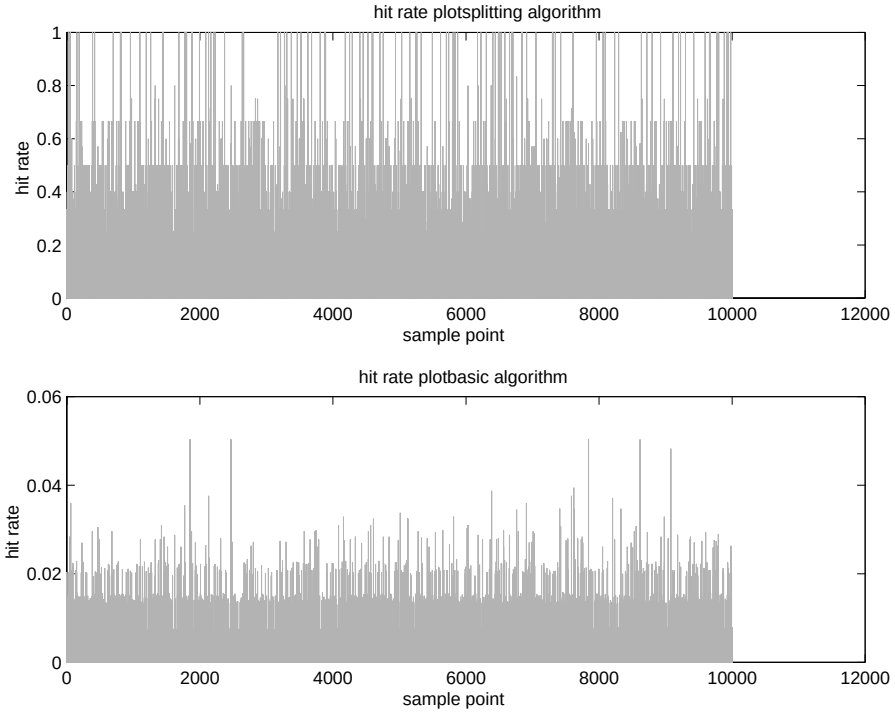


FIG. 9. *Hit rate plot in the basic and expanded quadtrees.*

TABLE 1
Quadtree statistics for mesh-free problems.

Data structure			Basic			Expanded			Naive
Problem	N	P	$\#R$	Hit rate	h	$\#R$	Hit rate	h	Hit rate
coarse	71	557	4	0.2372	2	16	0.3904	5	0.1998
medium	152	1286	5	0.1443	2	42	0.3033	7	0.1058
fine	349	3139	12	0.0775	3	106	0.2781	7	0.0419
fine2	605	5582	20	0.0516	3	223	0.2442	10	0.0246
fine3	985	8262	20	0.0396	3	240	0.1937	10	0.0155

and the naive algorithm's hit rate is roughly proportional to N^{-1}, while the expanded quadtree's hit rate is relatively stable. There is clearly additional overhead in setting up the quadtree using the splitting algorithm, but the number of rectangles allocated is much less than the number of support sets, and the depth of the resulting quadtree grows only slowly, probably logarithmically, in N. Note that the implementation of the splitting algorithm did not use any modifications to avoid deep quadtrees. Even so, path compression would achieve little here since the number of rectangles not linked by support sets for the splitting algorithm were, for the various problems, coarse 2; medium 6; fine 9; fine2 16. Thus for the largest problem (fine2), path compression could save no more than 16 of the 223 allocated rectangles. Thus it appears that for our application, path compression will save little in space or time.

3.3. The upper bounds of E(*depth*). As in the previous section, we let N_j denote the number of links at level j; $\widehat{N}_j$ denotes the number of rectangles linked to at least one support set in level j; and $\widetilde{N}_j$ denotes the number of allocated rectangles in

TABLE 2
Average covering numbers.

Problem	coarse	medium	fine	fine2	fine3
Avg. covering #	14.18	16.09	14.61	14.90	15.22

the quadtree at level j. Clearly, $\widehat{N}_j \le N_j$. Since every allocated rectangle must have a subrectangle that is linked to a support set, $\widetilde{N}_j \le \sum_{k=j}^{\infty} \widehat{N}_k \le \sum_{k=j}^{\infty} N_k$. On the other hand, $\widetilde{N}_j \le 2^{dj}$ since there are at most 2^{dj} rectangles in a quadtree at level j.

Note from the previous section that $\sum_{j=0}^{\infty} N_j \le N$ for the basic quadtree, but $\sum_{j=0}^{\infty} N_j \le 2^d N$ in the expanded quadtree.

3.4. Analysis of E(*depth*). Recall that *depth* is the depth of the quadtree traversed while searching for the sets covering a given point x, chosen at random. The basis of these calculations are bounds on the probability of *depth* having a particular value. A detailed discussion on related issues can be found in [2].

$$\begin{aligned}\mathbf{E}(depth) &= \sum_{j=0}^{\infty} j \Pr(depth = j) \\ &= \sum_{j=0}^{\infty} j \left[\Pr(depth \ge j) - \Pr(depth \ge j+1)\right] \\ &= \sum_{j=1}^{\infty} \Pr(depth \ge j).\end{aligned}$$

The problem now is to bound $\Pr(depth \ge j)$. This is the probability that x lies in an allocated rectangle at a depth j. A simple bound is

$$\Pr(depth \ge j) = 2^{-dj} \widetilde{N}_j \le 2^{-dj} \min(2^{dj}, 2^d N) = \min(1, 2^{-d(j-1)} N).$$

This leads to bounds of the form $\mathbf{E}(depth) = O(\log N)$:

$$\begin{aligned}\mathbf{E}(depth) &= \sum_{j=1}^{\infty} \Pr(depth \ge j) \\ &\le \sum_{j=1}^{\infty} \min(1, 2^{-d(j-1)} N).\end{aligned}$$

Suppose that $2^{d(j^*-1)} < N \le 2^{dj^*}$. Then for $j \ge j^*$, $2^{-d(j-1)} N \le 2^{-d(j-j^*-1)} \le 1$. Thus

$$\begin{aligned}\mathbf{E}(depth) &\le \sum_{j=1}^{j^*} 1 + \sum_{j=j^*+1}^{\infty} 2^{-d(j-j^*-1)} \\ &= j^* + \sum_{l=0}^{\infty} 2^{-dl} = j^* + \frac{1}{1-2^{-d}} \le j^* + 2.\end{aligned}$$

Note that $j^* = \lceil (\log_2 N)/d \rceil$, and so

$$\mathbf{E}(depth) \le \left\lceil \frac{\log_2 N}{d} \right\rceil + 2 = O\left(\frac{\log_2 N}{d} \right).$$

4. Conclusions. The expected total search time using the expanded quadtree for a random point $x \in top$ is $O(\mathbf{E}(depth)) + O(\mathbf{E}(\#tests))$. Since the number of tests needed is the covering number divided by the hit rate (which is bounded below by $2a(16a)^{-d}$) we get an expected search time which is $O((\log N)/d + (16a)^d \mathbf{E}(\#\{\, i \mid x \in S_i \,\}))$. For fixed d, this is $O(\log N + \mathbf{E}(\#\{i \mid x \in S_i\}))$. This means that for applications where the covering number grows at least as fast as $\log N$, we have an asymptotically optimal method. Otherwise, the expected time for the search grows like $O(\log N)$ which, while not asymptotically optimal, still gives good performance. By comparison, the expected total search time for the basic quadtree is $\Theta(N)$ for a worst-case collection of sets $\{S_i\}$, and $\Theta(N^{1-1/d})$ for collections of sets which occur in practical problems.

Acknowledgments. We would like to thank Prof. J. S. Chen and his students Hui-Ping Wang and Cheng-Tang Wu for providing the mesh-free method codes on which the computational results in Tables 1 and 2 are based. We would also like to thank the referees whose comments have helped to improve this paper.

REFERENCES

[1] D. J. Abel and J. L. Smith, *A data structure and algorithm based on a linear key for a rectangle retrieval problem*, Computer Vision, Graphics and Image Processing, 24 (1983), pp. 1–13.

[2] A. Andersson and S. Nilsson, *Faster searching in tries and quadtrees—an analysis of level compression*, in Lecture Notes in Comput. Sci. 855, Jan van Leeuwen, ed., Algorithms—ESA '94 (Utrecht), Springer, Berlin, 1994, pp. 82–93.

[3] A. Andersson and K. Swanson, *On the difficulty of range searching*, Comput. Geom., 8 (1997), pp. 115–122.

[4] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, *Meshless methods: An overview and recent developments*, Comput. Methods Appl. Mech. Engrg., 139 (1996), pp. 3–47.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry*, Algorithms and Applications, Springer-Verlag, Berlin, 1997.

[6] P. Gao and T. R. Smith, *Space efficient hierarchical structures: Relatively addressed compact quadtrees for GISs*, Image and Vision Computing, 7 (1989), pp. 173–177.

[7] I. Gargantini and H. H. Atkinson, *Linear quadtrees: A blocking technique for contour filling*, Pattern Recognition, 17 (1984), pp. 285–293.

[8] D. Hearn and M. P. Baker, *Computer Graphics*, 2nd ed., C Version, Prentice–Hall, Englewood Cliffs, NJ, 1994.

[9] M. J. Laszlo, *Computational geometry and computer graphics in C++*, Prentice–Hall, Upper Saddle River, NJ, 1996.

[10] S. Li and W. K. Liu, *Reproducing kernel hierarchical partition of unity: Part* I: *Formulation and theory*, Internat. J. Numer. Methods Engrg., 45 (1999), pp. 251–288.

[11] M. H. Overmars, *Point location in fat subdivisions*, Inform. Process. Lett., 44 (1992), pp. 261–265.

[12] F. P. Preparata and R. Tamassia, *Fully dynamic point location in a monotone subdivision*, SIAM J. Comput., 18 (1989), pp. 811–830.

[13] H. Samet, *The quadtree and related hierarchical data structures*, Comput. Surveys, 16 (1984), pp. 187–260.

[14] H. Samet, *Applications of Spatial Data Structures*, Addison–Wesley, Reading, MA, 1990.

[15] H. Samet, *The Design and Analysis of Spatial Data Structures*, Series in Computer Science, Addison–Wesley, Reading, MA, 1990.

[16] C. A. Shaffer, H. Samet, and R. C. Nelson, *A geographic information system based on quadtrees*, International Journal of Geographic Information Systems, 4 (1990), pp. 103–131.

[17] K. Yamaguchi, T. L. Kunii, K. Fujimura, and H. Toriya, *Octree-related data structures and algorithms*, IEEE Computer Graphics and Applications, 4 (1984), pp. 53–59.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1385–1404

AN $O(n \log n)$ ALGORITHM FOR THE MAXIMUM AGREEMENT SUBTREE PROBLEM FOR BINARY TREES*

RICHARD COLE†, MARTIN FARACH-COLTON‡, RAMESH HARIHARAN§, TERESA PRZYTYCKA¶, AND MIKKEL THORUP‖

Abstract. The maximum agreement subtree problem is the following. Given two rooted trees whose leaves are drawn from the same set of items (e.g., species), find the largest subset of these items so that the portions of the two trees restricted to these items are isomorphic. We consider the case which occurs frequently in practice, i.e., the case when the trees are binary, and give an $O(n \log n)$ time algorithm for this problem.

Key words. algorithms, agreement subtree

AMS subject classifications. 68Q25, 68W40

PII. S0097539796313477

1. Introduction.

Suppose we are given two rooted trees T_1 and T_2 with n leaves each. The internal nodes of each tree have at least two children each. The leaves in each tree are labeled with the same set of labels, and further, no label occurs more than once in a particular tree. An *agreement subtree* of T_1 and T_2 is defined as follows. Let L_1 be a subset of the leaves of T_1, and let L_2 be the subset of those leaves of T_2 which have the same labels as leaves in L_1. The subtree of T_1 *induced* by L_1 is an agreement subtree of T_1 and T_2 if and only if it is *isomorphic* to the subtree of T_2 induced by L_2. The maximum agreement subtree problem (henceforth called $MAST$) asks for the largest agreement subtree of T_1 and T_2.

We need to define the terms *induced subtree* and *isomorphism* used above. Intuitively, the subtree of T induced by a subset L of the leaves of T is the topological subtree of T restricted to the leaves in L, with branching information relevant to L preserved. More formally, for any two leaves a, b of a tree T, let $\mathrm{lca}_T(a, b)$ denote their lowest common ancestor in T. If $a = b$, $\mathrm{lca}_T(a, b) = a$. The *subtree* U of T *induced* by a subset L of the leaves is the tree with leaf set L and interior node set $\{\mathrm{lca}_T(a, b) | a, b \in L\}$ inheriting the ancestor relation from T; that is, for all $a, b \in L$, $\mathrm{lca}_U(a, b) = \mathrm{lca}_T(a, b)$.

Intuitively, two trees are isomorphic if the children of each node in one of the trees can be reordered so that the leaf labels in each tree occur in the same order and the

*Received by the editors December 16, 1996; accepted for publication (in revised form) April 2, 2000; published electronically November 8, 2000. This work was supported by NSF grants CCR-9202900 and CCR-9503309, by a Sloan and Department of Energy Postdoctoral Fellowship for Computational Biology, and by grant GM29458.

http://www.siam.org/journals/sicomp/30-5/31347.html

†Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012 (cs-chair@cs.nyu.edu).

‡DIMACS and Rutgers University, Piscataway, NJ 08855 (farach@cs.rutgers.edu).

§Indian Institute of Science, Bangalore 560012, India (ramesh@csa.iisc.ernet.in). This work was partly done while this author was at Max Planck Institut für Informatik, Germany, and while visiting New York University.

¶Johns Hopkins School of Medicine, 701 WBSB, 725 North Wolf Street, Baltimore, MD 21205 (przytyck@grserv.med.jhmi.edu).

‖AT&T Labs-Research, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932 (mthorup@research.att.com). Some of this work was done while this author was at the University of Copenhagen, Denmark, and while visiting MIT.

shapes of the two trees become identical. Formally, we say two trees U_1 and U_2 with the same leaf labels are *isomorphic* if there is a 1–1 mapping μ between their nodes, mapping leaves to leaves with the same labels and such that for any two different leaves a, b of U_1, $\mu(\text{lca}_{U_1}(a,b)) = \text{lca}_{U_2}(\mu(a), \mu(b))$.

Motivation. The *MAST* problem arises naturally in biology and linguistics as a measure of consistency between two evolutionary trees over species and languages, respectively. An evolutionary tree for a set of *taxa*, either species or languages, is a rooted tree whose leaves represent the taxa and whose internal nodes represent ancestor information. It is often difficult to determine the true phylogeny for a set of taxa, and one way to gain confidence in a particular tree is to have different lines of evidence supporting that tree. In the biological taxa case, one may construct trees from different parts of the DNA of the species. These are known as *gene trees*. For many reasons, these trees need not entirely agree, and so one is left with the task of finding a consensus of the various gene trees. The maximum agreement subtree is one method of arriving at such a consensus. Notice that a gene is usually a binary tree, since DNA replicates by a binary branching process. Therefore, the case of binary trees is of great interest.

Another application arises in automated translation between two languages [GY95]. The two trees are the parse trees for the same meaning sentences in the two languages. A complication that arises in this application (due in part to imperfect dictionaries) is that words need not be uniquely matched, i.e., a word at the leaf of one tree could match a number (usually small) of words at the leaves of the other tree. The aim is to find a maximum agreement subtree; this is done with the goal of improving context-using dictionaries for automated translation. So long as each word in one tree has only a constant number of matches in the other tree (possibly with differing weights), the algorithm given here can be used and its performance remains $O(n \log n)$. More generally, if there are m word matches in all, the performance becomes $O((m+n)\log n)$. Note, however, that if there are two collections of equal meaning words in the two trees of sizes k_1 and k_2, respectively, they induce $k_1 k_2$ matches.

Previous work. Finden and Gordon [FG85] gave a heuristic algorithm for the *MAST* problem on binary trees which had an $O(n^5)$ running time and did not guarantee an optimal solution. Kubicka, Kubicki, and McMorris [KKM95] gave an $O(n^{(.5+\epsilon)\log n})$ algorithm for the same problem. The first polynomial time algorithm for this problem was given by Steel and Warnow [SW93]; it had a running time of $O(n^2)$. Steel and Warnow also considered the case of nonbinary and unrooted trees. Their algorithm takes $O(n^2)$ time for fixed degree rooted and unrooted trees and $O(n^{4.5}\log n)$ time for arbitrary degree rooted and unrooted trees. They also give a linear reduction from the rooted to the unrooted case. Farach and Thorup gave an $O(nc^{\sqrt{\log n}})$ time algorithm for the *MAST* problem on binary trees; here c is a constant greater than 1. For arbitrary degree trees, their algorithm takes $O(n^2 c^{\sqrt{\log n}})$ time for the unrooted case [FT95] and $O(n^{1.5}\log n)$ time for the rooted case [FT97]. Farach, Przytycka, and Thorup [FPT95a] obtained an $O(n\log^3 n)$ algorithm for the *MAST* problem on binary trees. Kao [Ka95] obtained an algorithm for the same problem which takes $O(n\log^2 n)$ time. This algorithm takes $O(\min\{nd^2 \log d \log^2 n, nd^{\frac{3}{2}}\log^3 n\})$ for degree d trees. Finally, Cole and Hariharan [CR96] improved the algorithm from [FPT95a] to an $O(n\log n)$ algorithm.

The *MAST* problem for more than two trees has also been studied. Amir and Keselman [AK97] showed that the problem is NP-hard for even 3 unbounded degree

trees. However, polynomial bounds are known [AK97, FPT95b] for three or more bounded degree trees.

Our contribution. This paper is the combined journal version of [FPT95a] and [CR96] and presents an $O(n \log n)$ algorithm for the *MAST* problem for two binary trees.

The $O(n \log^3 n)$ algorithm of [FPT95a] can be viewed as taking the following approach (although the authors do not describe it this way). It identifies two special cases and then solves the general case by interpolating between these cases.

Special case 1. The internal nodes in both trees form a path. The *MAST* problem reduces to essentially a size n longest increasing subsequence problem in this case. As is well known, this can be solved in $O(n \log n)$ time.

Special case 2. Both trees T_1 and T_2 are complete binary trees. For each node v in T_2, only certain nodes u in T_1 can be usefully mapped to v, in the sense that the subtree of T_1 rooted at u and the subtree of T_2 rooted at v have a nonempty agreement subtree. There are $O(n \log^2 n)$ such pairs (u, v). This can be seen as follows. Note that for (u, v) to be such a pair, the subtree of T_1 rooted at u and the subtree of T_2 rooted at v must have a leaf-label in common. For each label, there are only $O(\log^2 n)$ such pairs obtained by pairing each ancestor of the leaf with this label in T_1 with each ancestor of the leaf with this label in T_2. The total number of interesting pairs is thus $O(n \log^2 n)$.

For each pair, computing the *MAST* takes $O(1)$ time, as it is simply a question of deciding the best way of pairing their children.

The interpolation process takes a centroid decomposition of the two trees and compares pairs of centroid paths, rather than individual nodes as in the complete tree case. The comparison of a pair of centroid paths requires finding matchings with special properties in appropriately defined bipartite graphs, a nontrivial generalization of the longest increasing subsequence problem. This process creates $O(n \log^2 n)$ interesting (u, v) pairs, each of which takes $O(\log n)$ time to process.

In [CR96] two improvements are given, each of which gains a $\log n$ factor.

Improvement 1. The complete tree special case is improved to $O(n \log n)$ time as follows. A pair of nodes (u, v), $u \in T_1$, $v \in T_2$, is said to be *interesting* if there is an agreement subtree mapping u to v. As is shown below, for complete trees, the total number of interesting pairs (u, v) is just $O(n \log n)$. Consider a node v in T_2. Let L_2 be the set of leaves which are descendants of v. Let L_1 be the set of leaves in T_1 which have the same labels as the leaves in L_2. The only nodes that may be mapped to v are the nodes u in the subtree of T_1 induced by L_1. The number of such nodes u is O(size of the subtree of T_2 rooted at v). The total number of interesting pairs is thus the sum of the sizes of all subtrees of T_2, which is $O(n \log n)$.

This reduces the number of interesting pairs (u, v) to $O(n \log n)$. Again, processing a pair takes $O(1)$ time. (This is less obvious, for identifying the descendants of u which root the subtrees with which the two subtrees of v can be matched is nontrivial.) Constructing the above induced subtree itself can be done in $O(|L_1|)$ time, as will be detailed later. The basic tool here is to preprocess trees T_1 and T_2 in $O(n)$ time so that least common ancestor (LCA) queries can be answered in $O(1)$ time.

Improvement 2. As in [FPT95a], when the trees are not complete binary trees, we take centroid paths and match pairs of centroid paths. The $O(\log n)$ cost that the algorithm in [FPT95a] incurs in processing each such interesting pair of paths arises when there are large (polynomial in n size) instances of the generalized longest increasing subsequence problem. At first sight, it is not clear that large instances of

these problems can be created for sufficiently many of the interesting pairs; unfortunately, this turns out to be the case. However, these problem instances still have some useful structure. By using (static) weighted trees, we process pairs of interesting vertices in $O(1)$ time per pair, on the average, as is shown by an appropriately parameterized analysis.

The paper is organized as follows. Section 2 gives some basic definitions and primitives. Section 3 outlines the algorithm. Section 4 provides further details of the algorithm, and section 5 gives the analysis. The remaining sections deal with problems raised in sections 3–5.

2. Definitions and preliminaries. All trees henceforth refer to binary trees whose internal nodes have exactly two children.

The tree $T(x)$ denotes the subtree of T rooted at vertex x. The *size* of a tree T, denoted by $|T|$, is the number of leaves in it. In our problem, $|T_1| = |T_2| = n$.

Given a binary tree T, its *centroid decomposition* is a partitioning of its vertices into disjoint paths obtained as follows. First, for each internal node x in T, the edge to the child with the maximal number of leaves below it is called a *centroid edge*. Here ties are broken arbitrarily. Now, the centroid edges form a collection of disjoint paths, called *centroid paths*. The *beginning* of such a centroid path P is defined to be the vertex x closest to the root of T. Then, if we remove P from $T(x)$, we get a forest of trees called the *side trees* of $T(x)$, and then each side tree is of size at most $|T(x)|/2$. Note that the full centroid decomposition could also be found recursively by first finding the centroid path from the root, and then recursing on each side tree. Also note that the centroid decomposition of T can easily be found in $O(n)$ time.

A tree T can be preprocessed in $O(|T|)$ time so that given any subset L of its leaves in left to right order, the subtree induced by L can be computed in $O(|L|)$ time. The details of this procedure are described in section 8.

The set of labels at the leaves of T_1 is identical to that at the leaves of T_2. For a leaf l in one of these trees, the leaf with the same label in the other tree is called its *twin*. Two subtrees, one from each tree, are said to *intersect* if and only if some leaf in one subtree has a twin in the other. The subtree of T_2 induced by some subset of the leaves of T_1 is the subtree of T_2 induced by the twins of these leaves of T_1.

3. Algorithm outline. We need some definitions to outline the algorithm.

Definitions. Let π be the centroid path containing the root of T_1. Let $p = |\pi|$, and let $u_1, u_2, \ldots, u_{p-1}, u_p$ be the vertices on this path in order from the root. Let $M_1, M_2, \ldots, M_{p-1}$ comprise the forest of side trees created by the removal of π from T_1. Let $m_i = |M_i|$ be the number of leaves in M_i (see Figure 3.1). Recall that $m_i \leq n/2$ as π is a centroid path. For technical reasons, we define M_p to be the tree consisting of the vertex u_p and set $m_p = 1$. Then, $\sum_{i=1}^{p} m_i = n$.

Given trees T_1, T_2, our aim is to determine the maximum agreement subtree of T_1 and T_2 efficiently. To do so, we will need to compute not just this agreement subtree but the maximum agreement subtree of T_1 and $T_2(w)$ for each w in T_2. All these subtrees will be computed implicitly by a procedure *Agree*(T_1, T_2).

Agree(T_1, T_2) proceeds broadly as follows. See Figure 3.1. First, it recursively computes the maximum agreement subtree of $M_i, T_2(w)$ for each side tree M_i of T_1 and each vertex w in T_2. This is not done explicitly, though, as we will see shortly. Second, it uses the information gathered in this process to compute the maximum agreement subtree of $T_1, T_2(w)$ for each vertex w in T_2. Both steps together take $O(n \log n)$ time.

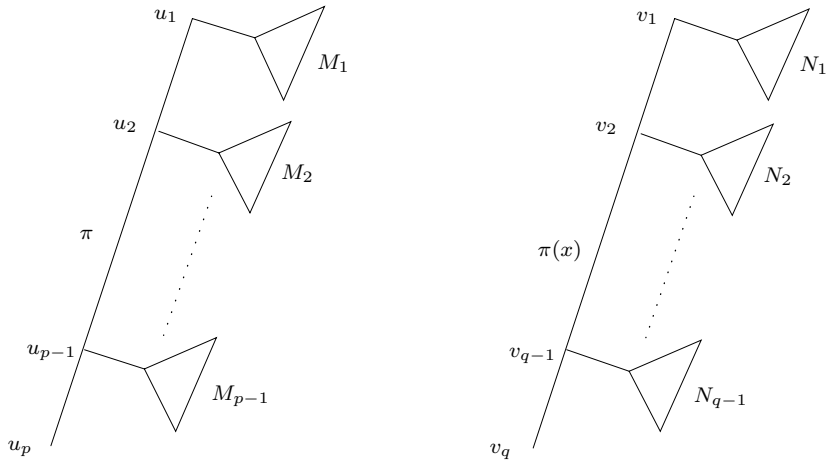


FIG. 3.1. *Centroid paths in T_1 and T_2 with π starting in root of T_1 and $\pi(x)$ starting in $x = v_1$ in T_2.*

We will describe $Agree(T_1, T_2)$ in detail shortly. Two observations will be used in this description.

First, as in the discussion of interesting pairs in Improvement 1, note that it is not necessary to compute the maximum agreement subtrees of $M_i, T_2(w)$ for all $w \in T_2$. The first reason is that the maximum agreement subtree of $M_i, T_2(w)$ is empty if M_i does not intersect with $T_2(w)$. The second reason is that the maximum agreement subtrees of $M_i, T_2(w)$ and $M_i, T_2(\text{parent}(w))$ are identical if M_i does not intersect with the the subtree of T_2 rooted at w's sibling. For these two reasons, it suffices to compute $Agree(M_i, S_i)$, where S_i is the subtree of T_2 induced by the leaves of M_i. This implicitly computes the maximum agreement subtrees of $M_i, T_2(w)$ for all $w \in T_2$.

Second, note that if the maximum agreement subtree of $T_1, T_2(w)$ does not have any vertex on the centroid path π of T_1, then all vertices in this agreement subtree belong to a single side tree M_i of T_1. For, if these vertices were distributed over two or more side trees of T_1, then these vertices would have a common ancestor on π which would be in the maximum agreement subtree as well. Thus, when the maximum agreement subtree of $T_1, T_2(w)$ does not have any vertex on π, it can be determined using some $Agree(M_i, S_i)$ from the previous paragraph.

Algorithm outline. $Agree(T_1, T_2)$ has three steps.

Step 1. The centroid decompositions of T_1 and T_2 are computed. This takes $O(n)$ time.

Step 2. For each i, $1 \le i \le p-1$, $Agree(M_i, S_i)$ is computed recursively, where S_i is the subtree of T_2 induced by the leaves of M_i. We will inductively assume that this takes $O(m_i \log m_i)$ for each i. Recall that if the maximum agreement subtree of T_1 and T_2 contains no vertex from π, then it will be found in Step 2. Step 3 handles the other case.

Step 3: *Matching π.* For each $w \in T_2$, the largest agreement subtree for the trees T_1 and $T_2(w)$ is found using information from Step 2. Informally, we call this the process of "matching" π at each of the vertices w of T_2. We will show how this is

done in $O(\sum_{i=1}^{p} m_i \log \frac{n}{m_i})$ time.

Clearly, the total time over all three steps is $O(n \log n)$. The following sections show how Step 3 is performed in $O(\sum_{i=1}^{p} m_i \log \frac{n}{m_i})$ time. Starting in the next section, we will define some bipartite graphs. Each graph will correspond to a particular centroid path in the centroid decomposition of T_2 and will be used to match π as in Step 3 at all the vertices in this centroid path. These graphs will have the property that a particular kind of matching, which can be computed efficiently, will correspond to the relevant agreement subtrees.

4. The matching graphs $G(x)$ and the π matching algorithm.

Definitions. Recall that the centroid decomposition of T_2 partitions its vertices into disjoint paths and that the beginning of such a path is defined to be the vertex closest to the root of T_2 in that path. Let X denote the set of vertices in T_2 at which paths in the above decomposition begin.

We define a number of bipartite graphs, one for each $x \in X$. The graph $G(x)$ corresponding to vertex x is defined as follows.

Vertices of $G(x)$. The left vertex set $L(x)$ of $G(x)$ is a subset of $\{u_1, \ldots, u_p\}$. Vertex u_i, $1 \leq i \leq p-1$, is in the set if and only if M_i and $T_2(x)$ intersect. Vertex u_p is in the set if and only if its twin is in $T_2(x)$.

The right vertex set $R(x)$ of $G(x)$ is exactly the set of vertices in the centroid path beginning at vertex x.

Since both sets of vertices are drawn from centroid paths, we order the vertices on each side in the order they occur on their respective centroid paths. The *topmost* vertex is the closest to the root and the *bottommost* is the farthest. Further, two edges (a, b) and (a', b') in $G(x)$ are said to *cross* if a is above a' and b is below b', or vice versa. In addition, edge (a, b) is said to *dominate* (a', b') in $G(x)$ if a is above a' and b is above b'. The *topmost* edge in a set of edges, if any, is the edge which dominates all other edges in that set.

Before defining the edges of $G(x)$, we need the following definitions.

Definitions. Let $\pi(x)$ be the centroid path containing x. Let q be the length of this path. Let $v_1, v_2, \ldots, v_q$ be the vertices on this path in order from the root. Let $N_1, N_2, \ldots, N_{q-1}$ comprise the forest of side trees of $T_2(x)$ created by the removal of $v_1, \ldots, v_q$ from $T_2(x)$. Let $n_i = |N_i|$ for $i = 1, \ldots, q-1$ (see Figure 3.1). For technical reasons, we define N_q to be the tree consisting of the vertex v_q and set $n_q = 1$. Then $\sum_{i=1}^{q} n_i = |T_2(x)|$.

4.1. Motivation for defining edges of $G(x)$.

We first motivate the definition of edges of the graph $G(x)$. The purpose of $G(x)$ is to determine maximum agreement subtrees of T_1 and $T_2(v_k)$ for each k, $1 \leq k \leq q$. The edges of $G(x)$ will be defined so that maximum weight matchings of a certain kind (called *agreement matchings*) in $G(x)$ will correspond to maximum agreement subtrees of T_1 and $T_2(v_k)$, $1 \leq k \leq q$. Clearly, the edges of $G(x)$ and the agreement matchings must capture the structural properties of these maximum agreement subtrees. We outline these structural properties next and show how they lead to the edge definitions.

Consider the maximum agreement subtree $\mathcal{A}$ of $T_1, T_2(v_k)$. Note that $\mathcal{A}$ has the following properties.

Case 1. If $\mathcal{A}$ has no vertices in $\pi(x)$, then it must be the maximum agreement subtree of (T_1, N_j) for some j, $k \leq j \leq q-1$.

Case 2. Similarly, if $\mathcal{A}$ has no vertices in π, then it must be the maximum agreement subtree of $(M_i, T_2(v_k))$ for some i, $1 \leq i \leq p-1$.

Case 3. Next, suppose $\mathcal{A}$ has at least one vertex both from π and from $\pi(x)$. In other words, there is at least one vertex in $\mathcal{A}$ which is in π and which maps to a vertex in $\pi(x)$.

Suppose vertex $u_i \in \pi$ is one such vertex and it maps to vertex $v_j \in \pi(x)$. If u_i is not the bottommost such vertex in π and z is the unique child of u_i in $\mathcal{A}$ which is not in π, then $\mathcal{A}(z)$ must be the maximum agreement subtree of (M_i, N_j).

Now if u_i is indeed the bottommost such vertex, we divide into subcases.

Case 3.0. If u_i is the leaf u_p, it must map to the other leaf v_q since leaves can only map to leaves. Then the subtree of $\mathcal{A}$ rooted at u_i is just the vertex u_i, which is also the maximum agreement subtree of $(T_1(u_i), N_q)$.

For the remaining subcases, we assume that u_i is not the leaf u_p.

Case 3.1. The subtrees of $\mathcal{A}$ rooted at the two children of u_i in $\mathcal{A}$ are the maximum agreement subtrees of the pairs $(T_1(u_{i+1}), N_j)$ and $(M_i, T_2(v_{j+1}))$. The following facts will be of use in this case. If M_i and N_{j+1} do not intersect, then the maximum agreement subtree of $(M_i, T_2(v_{j+1}))$ is identical to that of $(M_i, T_2(v_{j'}))$, where $j' > j$ is the topmost vertex below v_j in $\pi(x)$ such that M_i and $N_{j'}$ intersect. And if M_{i+1} and N_j do not intersect, then the maximum agreement subtree of $(T_1(u_{i+1}), N_j)$ is identical to that of $(T_1(u_{i'}), N_j)$, where $i' > i$ is the topmost vertex below u_i in π such that $M_{i'}$ and N_j intersect.

Note that if this subcase does not hold, then the subtree of $\mathcal{A}$ rooted at one of the two children of u_i is just the maximum agreement subtree of (M_i, N_j). In addition, all descendants of the other child of u_i in $\mathcal{A}$ must come from either a single side tree below v_j in T_2 or a single side tree below u_i in T_1. These situations are handled by the second and third cases, respectively.

Case 3.2. The subtrees of $\mathcal{A}$ rooted at the two children of u_i in $\mathcal{A}$ are the maximum agreement subtrees of (M_i, N_j) and $(T_1(u_{i+1}), N_{j'})$ for some j', $j < j' \leq q$.

Case 3.3. The subtrees of $\mathcal{A}$ rooted at the two children of u_i in $\mathcal{A}$ are the maximum agreement subtrees of (M_i, N_j) and $(M_{i'}, T_2(v_{j+1}))$, respectively, for some i', $i < i' \leq p$.

The following properties of $\mathcal{A}$ can be inferred from the above case analysis.

Property 1. $\mathcal{A}$ has a path (which is possibly empty) comprising vertices in π which map to vertices in $\pi(x)$, with the property that off-path subtrees are maximum agreement subtrees of the following three kinds: maximum agreement subtrees of (M_i, N_j) for some i, j, maximum agreement subtrees of $(T_1(u_i), N_j)$ for some i, j, and maximum agreement subtrees of $(M_i, T_2(v_j))$ for some i, j.

Property 2. $\mathcal{A}$ contains at most one maximum agreement subtree of the second kind and at most one of the third kind. $\mathcal{A}$ also contains at least one maximum agreement subtree of either the second or the third kind. More specifically, in Case 3.1, there is one of each kind, in Cases 1 and 3.2, there is one of the second kind but none of the third kind, and in Cases 2 and 3.3, there is one of the third kind but none of the second kind. Finally, Case 3.0 can be viewed as either kind since $M_p = T_1(u_p)$ and $N_q = T_1(v_q)$.

Property 3. In Case 3, there are zero or more maximum agreement subtrees of the first kind, all of which occur above subtrees of the second and third kinds. Here, by above we refer to the relative positions of the nearest ancestors u_i, v_j on the centroid paths. In Cases 1 and 2, there are no subtrees of the first kind.

Property 4. If two maximum agreement subtrees of the first kind occur in $\mathcal{A}$, for

instance, the maximum agreement subtrees of (M_i, N_j) and $(M_{i'}, N_{j'})$, then $i < i'$ implies $j < j'$.

Property 5. If subtrees of both the second kind and the third kind exist in $\mathcal{A}$, for instance, the maximum agreement subtrees of $(T_1(u_{i'}), N_j)$ and $(M_i, T_2(v_{j'}))$, respectively, then $i < i'$ and $j < j'$.

To model these three different kinds of maximum agreement subtrees, we need three different kinds of edges in $G(x)$, namely white edges, red edges, and green edges, respectively. The details of these edges are described next, followed by the definition of agreement matching which captures the above structural properties.

4.2. Edges of $G(x)$. $G(x)$ is actually a multigraph, where each multiedge consists of three edges—a *white* edge, a *red* edge, and a *green* edge—each of which has a distinct weight associated with it. A multiedge between $u_i \in L(x)$, $1 \leq i \leq p-1$, and $v_j \in R(x)$, $1 \leq j \leq q-1$, exists if and only if M_i and N_j intersect. The white edge in this multiedge has weight equal to the size of the maximum agreement subtree of M_i and N_j. The red edge in this multiedge has weight equal to the size of the maximum agreement subtree of $T_1(u_i)$ and N_j. The green edge in this multiedge has weight equal to the size of the maximum agreement subtree of M_i and $T_2(v_j)$. If $u_p \in L(x)$, then there is a multiedge between u_p and v_j such that either $j \neq q$ and u_p's twin is in N_j or $j = q$ and u_p's twin is v_q; all three edges in this multiedge have weight 1. In addition, there is a multiedge between u_i and v_q such that either $i \neq p$ and v_q's twin is in M_i or $i = p$ and v_q's twin is u_p; all three edges in this multiedge have weight 1.

4.3. Agreement matchings in $G(x)$.

Definitions. We define a *proper crossing* in $G(x)$ to be either a single red edge, a single green edge, or a red-green edge pair such that the two edges cross and, further, the endpoint of the green edge in $L(x)$ is above that of the red edge.

A matching in $G(x)$ is an *agreement matching* if

1. it has zero or more white edges and one proper crossing, and
2. no white edge crosses any other edge; further, all white edges dominate the edges in the proper crossing.

See Figure 4.1. The weight of such a matching is just the sum of the weights of its edges. The following property of agreement matchings in G is crucial.

The key property. Each maximum weight agreement matching corresponds to a maximum agreement subtree, and vice versa, as is made precise below.

LEMMA 4.1. *A maximum weight agreement matching $\mathcal{M}$ containing only edges incident upon or below vertex w in $R(x)$ corresponds to an agreement subtree $\mathcal{A}$ of $T_1, T_2(w)$, $w \in \pi(x)$, having the same weight.*

Proof. We associate with each white edge (u_i, v_j) the maximum agreement subtree of (M_i, N_j). Similarly, we associate with each red edge (u_i, v_j) the maximum agreement subtree of $(T_1(u_i), M_j)$ and with each green edge (u_i, v_j) the maximum agreement subtree of $(N_i, T_2(v_j))$. The tree $\mathcal{A}$ is defined as follows.

$\mathcal{A}$ will have a path containing one vertex for each white edge (see Figure 4.1). These vertices occur in the same sequence from top to bottom as their corresponding edges. The off-path children of these vertices will be the roots of the associated maximum agreement subtrees defined above. It remains to define the remaining child of the bottommost vertex on this path. This child will depend upon the nature of the proper crossing. We will define a tree $\mathcal{A}'$ for the proper crossing as follows. This child will be the root of $\mathcal{A}'$.

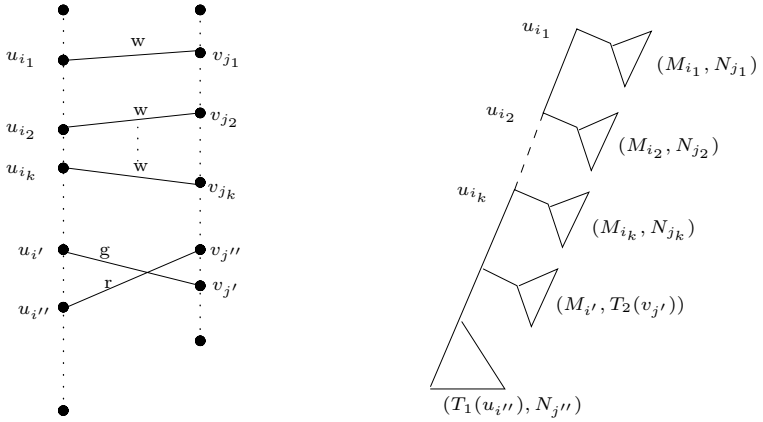


FIG. 4.1. *An agreement matching with the associated agreement tree.*

If the proper crossing has a green edge and a red edge, then $\mathcal{A}'$ is a tree whose left and right subtrees are the maximum agreement subtrees associated with the two edges. On the other hand, if the proper crossing has exactly one edge (red or green), then $\mathcal{A}'$ is just the maximum agreement subtree associated with that edge.

And finally, if there are no white edges at all, then $\mathcal{A} = \mathcal{A}'$. The equivalence of weights is easy to check in all cases. □

LEMMA 4.2. *A maximum agreement subtree $\mathcal{A}$ of T_1 and $T_2(w)$, $w \in \pi(x)$, has a unique corresponding agreement matching which has the same weight and contains only edges incident upon or below w in $R(x)$.*

Proof. We sketch how the agreement matching corresponding to $\mathcal{A}$ is constructed. Recall Cases 1–3 and Properties 1–5 in section 4.1.

If Case 1 occurs, then $\mathcal{A}$ is of the maximum agreement subtree of (T_1, N_j) for some v_j coinciding with or below w in $\pi(x)$. Let u_i be the topmost vertex in π such that M_i intersects with N_j. Then the maximum agreement subtree of $(T_1(u_i), N_j)$ has the same weight as that of (T, N_j). Further, there is a red edge between u_i and v_j in $G(x)$; this red edge constitutes the agreement matching. Clearly, the weight of this matching is identical to that of $\mathcal{A}$.

If Case 2 occurs, a similar argument shows that the matching comprises a solitary green edge with the same weight as $\mathcal{A}$. In Case 3, a similar argument can be used to show that the matching contains a sequence of zero or more noncrossing white edges, lying above a proper crossing. □

Thus, in order to determine the maximum agreement subtree of T_1 and $T_2(w)$, $w \in \pi(x)$, it suffices to determine the maximum weight agreement matching in $G(x)$ containing only edges incident upon or below vertex w in $R(x)$.

4.4. Finding maximum weight agreement matching in $G(x)$. To describe the algorithm for matching π, we need the following definitions followed by an important theorem. The theorem itself will be proved in section 7.

Definitions. The *degree* of a vertex in $G(x)$ is defined as the number of white edges incident on it. Let $d_x(u_i)$ denote the degree of u_i. A vertex in $L(x)$ is called a *singleton* vertex if it has degree one; the white edge incident on it is called a *singleton* edge. Let $nswe(x)$ denote the number of nonsingleton white edges in $G(x)$.

Let $nsav(x)$ denote the number of vertices in $R(x)$ which have at least one incident nonsingleton white edge. Let $SV(x)$ denote the set of singleton vertices in $L(x)$.

THEOREM 4.3. *Consider a particular $x \in X$. For each $u_i \in L(x)$, the largest weight agreement matching in $G(x)$ containing only edges incident on or below u_i in $L(x)$ can be found in time*

$$O\left(\sum_{i|d_x(u_i)>1} d_x(u_i)\log\frac{nsav(x)}{d_x(u_i)} + \sum_{(u_i,v_j)\in G(x)|d_x(u_i)=1} \log\frac{|T(x)|}{n_j}\right).$$

Further, for each $v_j \in R(x)$, the largest weight agreement matching in $G(x)$ containing only edges incident on or below v_j in $R(x)$ can also be found in the same time.

Theorem 4.3 is achieved by storing the vertices of $R(x)$ in an appropriately weighted search tree. The construction is described in section 7.

Algorithm outline for matching π. The matching graphs $G(x)$ for all $x \in X$ will be constructed in time proportional to the sum of the sizes of these graphs. This construction is described in section 6. Then each matching graph $G(x)$ is processed as follows (see Theorem 4.3). For each $u_i \in L(x)$, the largest weight agreement matching in $G(x)$ containing only edges incident on or below u_i in $L(x)$ is found. Further, for each $v_j \in R(x)$, the largest weight agreement matching in $G(x)$ containing only edges incident on or below v_j in $R(x)$ is also computed. This computation of agreement matchings is described in section 7. For each $w \in T_2$, the largest agreement subtree of T_1 and $T_2(w)$ can be determined easily from the above information as it is given by the largest weight agreement matching in $G(x)$ comprising only edges incident upon or below vertex w in $R(x)$. Section 5 shows that the total time taken above is $O(\sum_{i=1}^{p} m_i \log\frac{n}{m_i})$, as required.

Inferring maximum agreement subtrees. Consider a vertex $w \in T_2$; let x be the beginning of the centroid path in T_2 containing w. Then $w \in R(x)$. The maximum agreement subtree of T_1 and $T_2(w)$ is given by the largest weight agreement matching in $G(x)$ comprising only edges incident upon or below vertex w in $R(x)$.

5. The analysis. We need the following preliminary lemmas before beginning the analysis.

LEMMA 5.1. *Consider graph $G(x)$. Then*

$$\sum_{i|d_x(u_i)>1} d_x(u_i)\log\frac{nsav(x)}{d_x(u_i)} \le \sum_{i|d_x(u_i)>1} d_x(u_i)\log\frac{n}{m_i}.$$

Proof. Multiplying each side by $\ln 2$, we get the following equivalent inequality:

$$A = \sum_{i|d_x(u_i)>1} d_x(u_i)\ln\frac{nsav(x)}{d_x(u_i)} \le \sum_{i|d_x(u_i)>1} d_x(u_i)\ln\frac{n}{m_i} = B.$$

Note that $\sum_{i|d_x(u_i)>1} d_x(u_i) = nswe(x)$. Let $\alpha_i(x) > 0$ be such that $d_x(u_i) = \alpha_i(x)\frac{m_i}{n} nswe(x)$. Then $\sum_{i|d_x(u_i)>1} \alpha_i(x) m_i = n$. Also note that $nsav(x) \le nswe(x)$. Therefore,

$$A \le B - \sum_{i|d_x(u_i)>1} \alpha_i(x)\frac{m_i}{n} nswe(x)\ln\alpha_i(x).$$

It suffices to show that

$$C = \sum_{i|d_x(u_i)>1} \alpha_i(x) m_i \ln \alpha_i(x) \geq 0.$$

We split C into two terms:

$$C_1 = \sum_{i|d_x(u_i)>1, \alpha_i(x)\geq 1} \alpha_i(x) m_i \ln \alpha_i(x)$$

and

$$C_2 = \sum_{i|d_x(u_i)>1, 0<\alpha_i(x)<1} \alpha_i(x) m_i \ln \alpha_i(x),$$

$$C_1 \geq \sum_{i|d_x(u_i)>1, \alpha_i(x)\geq 1} (\alpha_i(x) - 1) m_i.$$

Further,

$$\sum_{i|d_x(u_i)>1, \alpha_i(x)\geq 1} (\alpha_i(x) - 1) m_i - \sum_{i|d_x(u_i)>1, \alpha_i(x)<1} (1 - \alpha_i(x)) m_i$$
$$= \sum_{i|d_x(u_i)>1} (\alpha_i(x) - 1) m_i = \sum_{i|d_x(u_i)>1} \alpha_i(x) m_i - \sum_{i|d_x(u_i)>1} m_i \geq n - n \geq 0.$$

Therefore $C \geq \sum_{i|d_x(u_i)>1, 0<\alpha_i(x)<1} (1 - \alpha_i(x) + \alpha_i(x) \ln \alpha_i(x)) m_i \geq 0$. □

Recall that S_i denotes the subtree of T_2 induced by the leaves of M_i.

LEMMA 5.2. $\sum_{x\in X|d_x(u_i)>1} d_x(u_i) = O(m_i)$.

Proof. Consider $G(x)$ such that $d_x(u_i) > 1$. Then all but one of the vertices of $R(x)$ adjacent to u_i are also in S_i; this is because M_i intersects both the right and the left subtrees of all but the bottommost of the vertices adjacent to u_i in $R(x)$. Since each vertex in S_i is in at most one matching graph $G(x)$ and since $|S_i| = m_i$, the lemma follows. □

Consider a vertex $u_i \in \pi$. From Theorem 4.3 and Lemma 5.1, the following work is assigned to u_i when considering the matching graph $G(x)$, $x \in X$.

1. If M_i and $T_2(x)$ do not intersect, then no work is assigned to u_i as u_i is not in $G(x)$.
2. If $d_x(u_i) = 1$, then the work assigned to u_i is $O(\log \frac{|T_2(x)|}{n_j})$, where v_j is the vertex in $\pi(x)$ adjacent to u_i.
3. If $d_x(u_i) > 1$, then the work assigned to u_i is $O(d_x(u_i) \log \frac{n}{m_i})$.

The following is a corollary of Lemma 5.2 and the above bounds.

COROLLARY 5.3. *The work assigned to vertex u_i over all matching graphs $G(x)$ with $d_x(u_i) > 1$ is $O(m_i \log \frac{n}{m_i})$.*

It now suffices to account for the work assigned to vertex u_i over all matching graphs $G(x)$ with $d_x(u_i) = 1$. Next, we show that this work is also $O(m_i \log \frac{n}{m_i})$. We use the tree S_i for this analysis.

Analyzing over S_i. Note that for each $x \in X$ such that $d_x(u_i) = 1$, x is not in S_i, i.e., either it lies on the path in T_2 between the endpoints of some edge e in S_i, or it lies on the path in T_2 between the root of S_i and the root of T_2. In the former case, x is said to *lie* on edge e of S_i. To handle the latter case, we add a dummy edge

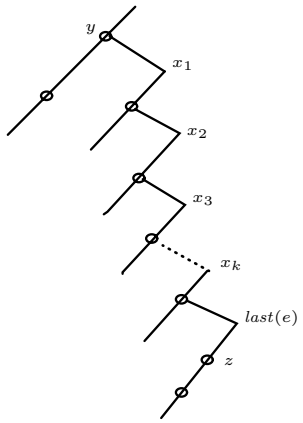


FIG. 5.1. *Portion of T_2 showing vertices in $H(e)$ and last(e).*

to S_i connecting its root to a new node, representing the root of T_2. This new node now becomes the root of S_i. This addition of the dummy edge is only for the next few paragraphs, until the work done on that edge is accounted for. Subsequent references to S_i will not have the dummy edge.

Consider the maximal subset $H(e)$ of vertices x in X which lie on edge e of S_i and for which $d_x(u_i) = 1$. Let $H(e) = \{x_1, x_2, \ldots, x_k\}$; here the vertices appear in increasing order of distance from the root of T_2. Let $e = (y, z)$, with y as the parent of z in S_i. Let *first*$(e) = x_1$. Let *last*(e) be the first vertex x in X such that $d_x(u_i) > 1$ and x is on the path from x_k to z in T_2, if any; otherwise, let *last*(e) be z (and then z is a leaf). See Figure 5.1.

LEMMA 5.4. *The sum of $|T_2($first$(e))|$ over any subset of edges of S_i, no two of which lie on the same root-to-leaf path in S_i, is $O(n)$.*

Proof. The above subtrees of T_2 are all disjoint. □

LEMMA 5.5. *The work assigned to u_i on edge e, i.e., in processing graphs $G(x)$, $x \in H(e)$, is $O(\log \frac{|T_2(\text{first}(e))|}{|T_2(\text{last}(e))|})$.*

Proof. The work assigned to u_i in processing $G(x_j)$ is $O(\log \frac{|T_2(x_j)|}{|T_2(x_{j+1})|})$ for $1 \leq j < k$ and $O(\log \frac{|T_2(x_k)|}{|T_2(\text{last}(e))|})$ for $j = k$. Thus the sum of the work assigned to u_i at the graphs $G(x_j)$ is $O(\log \frac{|T_2(\text{first}(e))|}{|T_2(\text{last}(e))|})$. □

COROLLARY 5.6. *The work assigned to u_i on the dummy edge e is $O(\log \frac{n}{m_i})$.*

Proof. $|T_2(\text{last}(e))| \geq m_i$. □

It remains to analyze the work assigned to u_i over the nondummy edges in S_i. From now onwards, we can ignore the dummy edge in S_i.

LEMMA 5.7. *Consider edges $e, e' \in S_i$ such that e is on the path from e' to the root of S_i. If $H(e), H(e')$ are nonempty, then $|T_2(\text{last}(e))| \geq |T_2(\text{first}(e'))|$.*

Proof. *first*(e') is a descendant of *last*(e) in T_2. □

We claim that sum of the work assigned to u_i over all the edges of S_i is $O(m_i \log \frac{n}{m_i})$. We show this next by applying tree contraction on S_i.

Removal step. First, remove all edges e in S_i incident upon leaves in S_i. The work done on these edges is bounded by the sum over all such edges e of $O(\log |T_2(\text{first}(e))|)$; further, the number of such edges is at most m_i. By Lemma 5.4, this sum is at most $O(m_i \log \frac{n}{m_i})$.

Contract step. Next, contract all paths consisting only of degree two vertices in S_i into a single edge. The H set for such an edge e is defined to be the union of the H sets for the edges comprising the path which was contracted to give e. The work done on e is also defined to be the sum of the work done on the relevant edges. *first*(e) and *last*(e) are again defined as before. As is easily seen, Lemmas 5.4, 5.5, and 5.7 hold for the new contracted tree as well. Further, this new tree has at most $m_i/2$ leaves.

Wrapping up. $O(\log m_i)$ phases of the removal and contract steps are performed. In the jth phase, the work done on the edges removed is $O(\frac{m_i}{2^{j-1}} \log \frac{n2^{j-1}}{m_i})$. Summing up over all phases, we get the following lemma.

LEMMA 5.8. *The work assigned to vertex u_i over all matching graphs $G(x)$ with $d_x(u_i) = 1$ is $O(m_i \log \frac{n}{m_i})$.*

The following lemma is needed in the next section.

LEMMA 5.9. *The total number of edges incident on u_i over all matching graphs is $O(m_i \log \frac{n}{m_i})$.*

Proof. Recall that the work attributed to a singleton edge is of the form $\log \frac{|T_2(x)|}{n_j} \geq 1$. Thus, Lemma 5.8 implies that there are $O(m_i \log \frac{n}{m_i})$ singleton edges incident to u_i, and by Lemma 5.2, there are $O(m_i)$ nonsingleton edges incident to u_i. ∎

THEOREM 5.10. *There is an algorithm for the MAST problem for two binary trees with an $O(n \log n)$ running time.*

Proof. Corollary 5.3 and Lemma 5.8 imply that the total work assigned to u_i is $O(m_i \log \frac{n}{m_i})$. Hence the total matching cost from Step 3 in section 3 is $O(\sum m_i \log \frac{n}{m_i})$. Further, Step 1 takes linear time. Also, in section 8, it will be shown that we can construct the recursive subproblems in Step 2 in linear time, so if we exclude the recursive calls, our cost is bounded by $c \sum m_i \log \frac{n}{m_i}$ for some sufficiently large c. Inductively, we assume that each recursive call takes at most $cm_i \log m_i$ time, and then the total time is at most

$$c \sum_i \left(m_i \log \frac{n}{m_i} + m_i \log m_i \right) = cn \log n,$$

as desired. ∎

The above analysis assumes (a) that we can construct the recursive subproblems in linear time, which will be done in section 8, (b) that we can construct the matching graphs in time proportional to their sizes, which will be done in sections 6 and 8, and (c) that Theorem 4.3 holds true, which will be proved in sections 7 and 9.

6. Constructing the matching graphs. We show how all the matching graphs can be set up in time proportional to the sum of their sizes, which by Lemma 5.9 is $O(\sum_{i=1}^{p-1} m_i \log \frac{n}{m_i})$. First, we show how to set up the vertices and edges in each graph. Then we show how the weights on the edges are computed.

Preprocessing. T_2 is preprocessed in linear time to compute a pointer from each vertex to the beginning of the centroid path containing it. It is also preprocessed to enable induced subtree computations in the same time bounds.

6.1. Setting up vertices and edges. The matching graphs in which vertex u_i appears along with the multiedges incident upon it in these graphs are determined in time proportional to the sum of the number of such multiedges over all such graphs as follows.

Processing u_p. First, consider the leaf u_p of T_1. The only matching graphs containing u_p are those which correspond to centroid paths beginning at vertices x of T_2 such that x is an ancestor of the twin of u_p in T_2. Further, if $u_p \in L(x)$, then there is a multiedge between u_p and vertex $y \in R(x)$ if and only if y is the nearest ancestor of u_p's twin in the centroid path beginning at x. Thus the matching graphs to which u_p belongs and the multiedges incident on u_p in these graphs can be determined in time proportional to the number of such graphs, given pointers from each vertex in T_2 to the beginning of the centroid path containing it.

Lemmas 6.1 and 6.2 are needed for the next step.

LEMMA 6.1. *If vertex v_j in the centroid path beginning at vertex x of T_2 is in S_i, then u_i is adjacent to v_j in $G(x)$.*

Proof. Clearly, if $j \neq q$, then M_i and N_j intersect, and if $j = q$, then v_j's twin is in M_i. □

LEMMA 6.2. *If vertex v_j in the centroid path beginning at vertex x of T_2 is not in S_i, then u_i is adjacent to v_j in $G(x)$ if and only if $v_j \neq v_q$ and there exists some vertex $y \in S_i$ which is in N_j.*

Proof. We assume that $j \neq q$. For if $j = q$, then $v_j = v_q$ is a leaf of T_2, and since it does not appear in S_i, its twin is not in M_i, and therefore, there is no edge between u_i and v_j.

First, suppose some vertex $y \in S_i$ is in N_j. Then, clearly, N_j intersects M_i. Therefore, there must be an edge between u_i and v_j in $G(x)$. Next, suppose that there is such an edge. Then N_j intersects M_i. Therefore, there exists some $y \in S_i$ which is in N_j. □

Processing $u_i, 1 \leq i \leq p-1$. The subtrees of T_2 induced by leaves of each M_i are computed in $O(\sum_1^{p-1} m_i)$ time as described in section 8. Let S_i denote this induced subtree. For each vertex z in S_i, perform the following in T_2 until a vertex in the centroid path containing the parent of z in S_i is reached: repeatedly jump to the parent of the beginning of the centroid path in T_2 containing the current vertex. By Lemmas 6.1 and 6.2, there is a multiedge from u_i to each vertex y of T_2 (in the corresponding matching graph containing y) encountered in this following procedure. Thus this procedure takes time proportional to the sum of the number of multiedges incident on u_i over all matching graphs it lies in, given pointers from each vertex in T_2 to the beginning of the centroid path containing it.

Remark. For an edge between u_i and v_j, $i \neq p$, $j \neq q$, define $map(i, j)$ as follows. If $v_j \in S_i$, then $map(i, j) = v_j$. Otherwise, if $v_j \notin S_i$, then $map(i, j)$ is that vertex in S_i which is closest to the root of S_i and a descendant of v_j in T_2. Note that $map(i, j)$ can be easily computed in the course of the above procedure.

6.2. Determining edge weights in $G(x)$. Recall that for a multiedge between u_i and v_j in $G(x)$, we need to determine the sizes of the maximum agreement subtrees of the following pairs of trees.

1. M_i, N_j: white edge weight.
2. $T_1(u_i)$, N_j: red edge weight.
3. M_i, $T_2(v_j)$: green edge weight.

Also recall that the multiedge itself indicates that M_i and N_j intersect.

Assume that the agreement matchings in graphs $G(x')$ have already been determined, where x' is a descendant of x in T_2. Using this information and the information computed in Step 2, we show how the above required information can be computed for multiedge (u_i, v_j) in graph $G(x)$ in constant time. Recall that in Step 2, the max-

imum agreement subtrees of M_i and the subtrees rooted at each vertex w of S_i were determined.

White edge weight. Let $y = map(i, j)$. If $y \neq v_j$, then the maximum agreement subtree of M_i and the subtree of S_i rooted at y gives the desired information. Suppose $y = v_j$, i.e., $y \in S_i$. Let z be the child of $y \in S_i$ such that $z \in N_j$. The maximum agreement subtree of M_i and the subtree of S_i rooted at z gives the desired information in this case. This takes constant time.

Green edge weight. The maximum agreement subtree of M_i and the subtree of S_i rooted at $y = map(i, j)$ gives the desired information in constant time.

Red edge weight. Let y be the root of N_j. Recall that the agreement matchings in graphs $G(x')$ have already been determined, where $x' \in X$ is a descendant of x in T_2. Since $y \in X$, agreement matchings in graph $G(y)$ would already have been computed. Recall from Theorem 4.3 that for each vertex in $L(y)$, the maximum weight agreement matching containing only edges incident on or below that vertex in $L(y)$ has been computed.

Note that since M_i intersects with $T_2(y)$ (since a multiedge exists between u_i and v_j), $u_i \in L(y)$. The largest weight agreement matching in $G(y)$ containing only edges incident on or below vertex u_i in $L(y)$ gives the desired information. This information is computed as graph $G(y)$ was processed, so it can be accessed in constant time now.

7. Computing agreement matchings. Consider graph $G(x)$. Recall that for each vertex in $L(x)$, we need to compute the largest weight agreement matching containing only edges incident on or below it in $L(x)$, and likewise for each vertex in $R(x)$. We outline the algorithm before giving details. The algorithm is similar to that in [FPT95a], but the data structure we use and the associated operations are different.

Algorithm outline. First, a weight balanced binary search tree $\mathcal{T}$ whose leaves are the vertices in $R(x)$ is set up; here, the vertices in $R(x)$ are given appropriate weights yet to be described. Next, the vertices in $L(x)$ are considered in turn in bottom-to-top order. For each vertex $u_i \in L(x)$, the vertices adjacent to it in $R(x)$ are searched for in $\mathcal{T}$; the largest weight agreement matching with each white edge incident on u_i as topmost edge is found in the course of this search, as is the largest weight proper crossing for each green edge incident on u_i. From the above information, the largest weight agreement matching containing only edges incident on or below u_i in $L(x)$ is easily found. Following the above search, the information stored in $\mathcal{T}$ is updated. The time taken for processing u_i will be $O(d_x(u_i) \log \frac{nsav(x)}{d_x(u_i)})$ if $d_x(u_i) > 1$ and $O(\log \frac{|T(x)|}{n_j})$ if $d_x(u_i) = 1$ and u_i is adjacent to v_j in $G(x)$. After all vertices in $L(x)$ have been processed, the vertices in $R(x)$ are processed. For all such vertices v_j, the largest weight agreement matching containing only edges incident on or below v_j in $R(x)$ are found in $O(|R(x)|)$ time by a single scan of $\mathcal{T}$. The bounds in Theorem 4.3 follow.

Weighted search tree $\mathcal{T}$. Vertex $v_j \in R(x)$ is given weight $n_j + \frac{|T(x)|}{nsav(x)}$ if some nonsingleton edge in $G(x)$ is incident upon it, and weight n_j otherwise. The sum of the weights of vertices in $R(x)$ is at most $2|T(x)|$. The construction of $\mathcal{T}$ using these weights is dealt with in section 9.

Tree $\mathcal{T}$ has the following three crucial characteristics.

1. $\mathcal{T}$ can be constructed in $O(|R(x)|)$ time.
2. Searching for v_j in $\mathcal{T}$ takes $O(\log \frac{|T(x)|}{n_j})$ time.

3. Searching for an ordered subset $\{v_{j_1}, \ldots, v_{j_k}\}$ of $R(x)$, each vertex in which has an incident nonsingleton edge, takes $O(k \log \frac{nsav(x)}{k})$ time. The procedure used here is to first search for v_{j_1}, starting at the root, and then to search for v_{j_2}, starting at v_{j_1} in the obvious way, and so on.

Auxiliary information in $\mathcal{T}$. We maintain the following auxiliary information at each internal vertex in $\mathcal{T}$. Recall that we process the vertices of $L(x)$ in order; an edge of $G(x)$ is said to be *in* $\mathcal{T}$ if its endpoint in $L(x)$ has already been processed. Further, we say that an edge of $G(x)$ is *in* $\mathcal{T}(z)$ if it is in $\mathcal{T}$ and its endpoint in $R(x)$ is located in $\mathcal{T}(z)$. Let $anc(z)$ denote the set of ancestors of z in $\mathcal{T}$, z inclusive. For a leaf $v_j \in \mathcal{T}$, $lfringe(v_j)$ is the set of vertices in $\mathcal{T}$ which are left children of vertices in $anc(v_j)$ but not themselves in $anc(v_j)$. $rfringe(v_j)$ is defined analogously.

The following information is maintained at each vertex z of $\mathcal{T}$.

1. $g(z)$: For each z, $\max_{z' \in anc(z)} g(z')$ will be the heaviest green edge in $\mathcal{T}$ which forms a proper crossing with each red edge in $\mathcal{T}(z)$.
2. $x(z)$: This is largest weight proper crossing among the edges in $\mathcal{T}(z)$.
3. $m(z)$: This is largest weight agreement matching containing a white edge such that the topmost white edge is in $\mathcal{T}(z)$.
4. $y(z)$: This the largest weight proper crossing such that the green edge in this crossing is in $\mathcal{T}$ but not in $\mathcal{T}(z)$, the red edge in this crossing is in $\mathcal{T}(z)$, and the green edge does not form a proper crossing with all the red edges in $\mathcal{T}(z)$.
5. $r(z)$: This is simply the heaviest red edge in $\mathcal{T}(z)$.

Next, we show how vertex u_i is processed, given that vertices below it in $L(x)$ have been processed. For the moment, assume that $d_x(u_i) = 1$. The case when $d_x(u_i) > 1$ will be addressed later.

Case 1. $d_x(u_i) = 1$. Let v_j be the only vertex to which u_i is adjacent. First, v_j is found in $\mathcal{T}$; this takes $O(\log \frac{|T(x)|}{n_j})$ time. Next, the white, red, and green edges incident on u_i are processed as described below in the same time bound. An important fact to note is that in each case, the information in $\mathcal{T}$ will be read and updated only at vertices in the set $anc(v_j)$ and vertices which are children of vertices in this set; the time taken in this process will be proportional to the depth of v_j, i.e., $O(\log \frac{|T(x)|}{n_j})$.

Processing white edge $e = (u_i, v_j)$. First, the largest weight agreement matching with e as the topmost edge is determined. Then the $m()$ values at vertices in $anc(v_j)$ are updated according to the weight of this matching. All other information remains unchanged.

The above desired matching is computed as follows. There are two cases. In the first case, this matching contains another white edge. The largest weight matching among all such matchings is given by $1 + \max_{z \in lfringe(v_j)} m(z)$. The other case occurs when this matching contains only edge e plus a proper crossing. Thus, it suffices to compute the largest proper crossing containing edges dominated by e. This is given by

$$\max \left\{ \max_{z \in lfringe(v_j)} x(z), \max_{z \in lfringe(v_j)} \left(\max_{z' \in anc(z)} g(z') \right) + r(z), \max_{z \in lfringe(v_j)} y(z) \right\}.$$

The first term here is the largest weight proper crossing in which both edges are in $\mathcal{T}(z)$ for some $z \in lfringe(v_j)$. The second term is the largest weight proper crossing in which the red edge is in $\mathcal{T}(z)$ for some $z \in lfringe(v_j)$; the green edge is not in this subtree but it forms a proper crossing with each red edge in this subtree. The third term is the largest weight proper crossing in which the red edge is in $\mathcal{T}(z)$ for some

$z \in \mathit{lfringe}(v_j)$; the green edge is not in this subtree and it does not form a proper crossing with some red edge in this subtree.

Processing red edge $e = (u_i, v_j)$. The $m()$ and $x()$ values remain unchanged in $\mathcal{T}$. Next, note that no green edge already in $\mathcal{T}$ can form a proper crossing with e. This implies that the $y()$ and $g()$ values for $z \in anc(v_j)$ need to be modified.

Consider $y(z)$ first, $z \in anc(v_j)$. A green edge in $\mathcal{T}$ which formed a proper crossing with all red edges in $\mathcal{T}(z)$ does not do so any more. So $y(z)$ is set to $\max\{y(z), (\max_{z' \in anc(z)} g(z')) + r(z)\}$.

Consider $g(z)$ next, $z \in anc(v_j)$. $g(z)$ is set to ϕ. Before this is done, $g(y)$ is updated to $\max_{y' \in anc(y)} g(y')$ for each $y \in \mathit{lfringe}(v_j)$ and $y \in \mathit{rfringe}(v_j)$. The invariant on $g()$ is easily seen to be maintained.

Finally, $r(z)$ is set to $\max\{r(z), wt(e)\}$ for each z, $z \in anc(v_j)$.

Processing green edge $e = (u_i, v_j)$. Note that e can form a proper crossing with only those red edges in $\mathcal{T}$ which are in $\mathcal{T}(z)$, $z \in \mathit{rfringe}(v_j)$; further, e forms a proper crossing with each such red edge. Therefore, $g(z)$ is set to

$$\max\left\{ \max_{z' \in anc(z)} g(z'), wt(e) \right\}$$

for each $z \in \mathit{rfringe}(v_j)$.

For each $z \in anc(v_j)$, $x(z)$ is then set to the larger of the current value and $\max(wt(e) + r(z'))$, the maximum being taken over all vertices $z' \in \mathit{rfringe}(v_j)$ which are descendants of z. Also note that $\max_{z \in \mathit{rfringe}(v_j)}(wt(e) + r(z))$ gives the largest weight proper crossing containing e.

Case 2. $d_x(u_i) = k > 1$. Suppose u_i is adjacent to $v_{j_1}, v_{j_2}, \ldots, v_{j_k}$, in bottom-to-top order. Then these vertices are searched for in sequence in $\mathcal{T}$. This takes $O(k \log \frac{nsav(x)}{k})$ time by the procedure mentioned earlier, i.e., first search for v_{j_1}, starting at the root, and then search for v_{j_2}, starting at v_{j_1} in the obvious way, and so on. In the above process, all vertices in the set $\{z | z \in (anc(v_{j_1}) \cup anc(v_{j_2}) \cup \cdots \cup anc(v_{j_k}))\}$ are traversed. Again, as in Case 1, only information at vertices in the above set and at children of vertices in the above set needs to be read and updated. This takes time proportional to the size of the above set, which, in turn, is $O(k \log \frac{nsav(x)}{k})$.

Processing $R(x)$. It remains to show how, for each $v_j \in R(x)$, the largest weight agreement matching containing only edges incident on or below v_j in $R(x)$ is computed.

For each $v_j \in R(x)$, we find the largest weight agreement matching with some white edge incident upon v_j as the dominant edge and the largest weight proper crossing containing some red edge incident on v_j. This information clearly suffices. The first of the above two is given simply by $m(v_j)$. The second is given by $\max\{y(z), \max_{z' \in anc(v_j)} g(z') + r(v_j)\}$. Over all $v_j \in R(x)$, the computation of the above two values can be accomplished in a single pass of $\mathcal{T}$ in $O(|R(x)|)$ time.

8. Computing induced subtrees. We show how to preprocess a tree in $O(|T|)$ time so that given any subset L of its leaves in order, the subtree induced by L can be computed in $O(|L|)$ time. The construction is a generalization of the proof of Lemma 5.2 in [FT95].

T is preprocessed for LCA queries in $O(|T|)$ time. This enables the computation of the LCA of any two leaves of T in constant time [HT84]. The distance of each vertex from the root of T is also computed; call this quantity the *depth* of a vertex.

Given the ordered set of leaves $L = l_1, l_2, \ldots, l_{|L|}$, the following steps are executed. First, the LCA l'_i of each pair of consecutive leaves l_i, l_{i+1}, $1 \leq i \leq |L| - 1$, is found; the l'_is will be the internal vertices in the subtree induced by L. Next, the edges between vertices are set up as follows.

For each vertex v in the sequence $l_1, l'_1, l_2, l'_2, \ldots, l_{|L|-1}, l'_{|L|-1}, l_{|L|}$, two vertices v_{left} and v_{right} are computed. v_{left} is the nearest vertex to the left of v, if any, which has depth strictly less than v. v_{right} is defined analogously. This computation is easily accomplished in $O(|L|)$ time. Finally, edges are put between v and one of v_{left}, v_{right}—whichever has greater depth. If exactly one of v_{left}, v_{right} is defined (this will happen only for vertices on the paths from the root to the leftmost and rightmost leaves in the induced subtree), then an edge is put between v and the vertex which is defined. The root of the induced tree will be the unique vertex for which both v_{left} and v_{right} are undefined; no edges need be put in this case.

Step 2 of the main algorithm. Step 2 (see section 3) requires finding the induced trees S_i for each M_i, $1 \leq i \leq p-1$, in $O(\sum_i^{p-1} m_i)$ time. This is done in two steps. (Essentially, this procedure is described in [FT95].) First, the leaves of each M_i are sorted by the order in which their twins occur in T_2. This is done by bucket sorting all the leaves of T_1 by the order in which their twins occur in T_2, and then bucket sorting them (in a stable way) by the order in which the trees M_i to which they belong occur in T_1. This takes $O(\sum_i^{p-1} m_i)$ time.

Next, for each M_i, $1 \leq i \leq p-1$, the subtree of T_2 induced by the leaves of M_i is found using the algorithm described above in $O(m_i)$ time. The total time taken is $O(\sum_i^{p-1} m_i)$.

9. The weighted search tree. We will now complete our solution to the *MAST* problem by describing the weighted search trees from section 7. Recall that we are given vertices $v_1, v_2, \ldots, v_{|R(x)|}$, such that vertex v_j has weight $w(v_j) = n_j + \frac{|T(x)|}{nsav(x)}$ if it has an incident nonsingleton edge, and weight $w(v_j) = n_j$ otherwise. The sum of the vertex weights is bounded by $2|T(x)|$.

Theorem 9.1 from [Fre75, Meh77] shows that the weight balanced tree $\mathcal{T}$ can be constructed in $O(|R(x)|)$ time.

THEOREM 9.1. *Given weights $w_1, \ldots, w_n$ with sum W, a binary tree such that the depth of the ith leaf is $O(1 + \log(W/w_i))$ can be constructed in $O(n)$ time. In this tree, the total weight of all leaves in the subtree rooted at any node z will be at most half of the corresponding weight for the subtree rooted at the grandparent of z.*

It follows from Theorem 9.1 that the time to search for v_j in $\mathcal{T}$ is $O(1+\log \frac{|2T(x)|}{w(v_j)}) = O(1 + \log \frac{|T(x)|}{n_j})$.

Next consider the case when an ordered subset $\{v_{j_1}, \ldots, v_{j_k}\}$ of vertices is given, each having an incident nonsingleton edge. The algorithm to search for these vertices is to first start from the root and search for v_{j_1}, then start from v_{j_1} and search for v_{j_2}, then start from v_{j_2} and search for v_{j_3}, and so on. Each search is performed in the obvious way. For technical reasons, we return to the root at the end.

Each edge in $\mathcal{T}$ is traversed at most twice during the above search, once in each direction.

Consider the topological subtree formed by the traversed edges. It has k leaves and $k-1$ internal nodes with two children. These nodes form a tree, with each edge

in the tree corresponding to a path in the search tree. We associate with a node in the topological subtree the path in the search tree corresponding to the edge from the node to its parent in the topological subtree. The associated path for the root node of the topological subtree is the path from this node to the root of the search tree. We will give an upper bound on the total number of vertices on these paths excluding their endpoints. To do this, we give a lower bound on the total weight of the "off-path" subtrees for each path. (An "off-path" subtree for a node is simply the subtree which does not contain the continuation of the path.)

Let l be the number of internal vertices on one such path associated with node v. By Theorem 9.1, the sum of the weights of every second root of the off-path subtrees is at least $(2w(v) - w(v)) + (4w(v) - 2w(v)) + \cdots + (2^{\lfloor l/2 \rfloor} w(v) - 2^{\lfloor l/2 \rfloor - 1} w(v)) = (2^{\lfloor l/2 \rfloor} - 1)w(v)$. But $w(v) \geq \frac{|T(x)|}{nsav(x)}$ and the total weight is at most $2T(x)$. Simple calculus shows that the sum of these lower bounds on the path lengths is maximized when the terms $w(v)$ are all at their minimum value and the path weights are all equal at $2T(x)/(2k-1)$. This gives path lengths of $O(\log \frac{nsav(x)}{2k-1})$ and hence a total path length of $O(k \log \frac{nsav(x)}{k})$.

10. Concluding remarks. We can generalize our technique to higher degree bounds $d > 2$ by combining it with techniques from [FT95, section 2] for unbounded degrees. This appears to yield an algorithm with running time $O(\min\{n\sqrt{d} \log^2 n, nd \log n \log d\})$. We conjecture, however, that there is an algorithm with running time $O(n\sqrt{d} \log n)$.

Acknowledgments. We would like to thank the referees for some very thorough comments.

REFERENCES

[AK97] A. Amir and D. Keselman, *Maximum agreement subtree in a set of evolutionary trees,* SIAM J. Comput., 26 (1997), pp. 1656–1669.

[CR96] R. Cole and R. Hariharan, *An $O(n \log n)$ algorithm for the maximum agreement subtree problem for binary trees,* in Proceedings of the 7th ACM–SIAM Symposium on Discrete Algorithms, ACM, New York, SIAM, Philadelphia, 1996, pp. 323–332.

[FPT95a] M. Farach, T. Przytycka, and M. Thorup, *The maximum agreement subtree problem for binary trees,* in Proceedings of the 2nd European Symposium on Algorithms, Corfu, Greece, 1995, Lecture Notes in Comput. Sci. 979, Springer-Verlag, New York, 1995, pp. 381–393.

[FPT95b] M. Farach, T. Przytycka, and M. Thorup, *Agreement of many bounded degree evolutionary trees,* Inform. Process. Lett., 55 (1995), pp. 297–301.

[FT95] M. Farach and M. Thorup, *Fast comparison of evolutionary trees,* Inform. and Comput., 123 (1995), pp. 29–37.

[FT97] M. Farach and M. Thorup, *Sparse dynamic programming for evolutionary-tree comparison,* SIAM J. Comput., 26 (1997), pp. 210–230.

[FG85] C. R. Finden and A. D. Gordon, *Obtaining common pruned trees,* J. Classification, 2 (1985), pp. 255–276.

[Fre75] M. L. Fredman, *Two applications of a probabilistic search technique: Sorting $X + Y$ and building balanced search trees,* in Proceedings of the 7th ACM Symposium on the Theory of Computing, ACM, New York, 1975, pp. 240–244.

[GY95] R. Grishman and R. Yangarber, *private communication*, New York University, New York, 1995.

[HT84] D. Harel and R. E. Tarjan, *Fast algorithms for finding nearest common ancestors,* SIAM J. Comput., 13 (1984), pp. 338–355.

[Ka95] M.-Y. Kao, *Tree contractions and evolutionary trees,* SIAM J. Comput., 27 (1998), pp. 1592–1616.

[KKM95] E. Kubicka, G. Kubicki, and F. R. McMorris, *An algorithm to find agreement subtrees,* J. Classification, 12 (1995), pp. 91–100.

[Meh77] K. Mehlhorn, *A best possible bound for the weighted path length of binary search trees,* SIAM J. Computing, 6 (1977), pp. 235–239.

[SW93] M. Steel and T. Warnow, *Kaikoura tree theorems: Computing the maximum agreement subtree,* Inform. Process. Lett., 48 (1993), pp. 77–82.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1405–1426

THE WARM-UP ALGORITHM: A LAGRANGIAN CONSTRUCTION OF LENGTH RESTRICTED HUFFMAN CODES*

RUY LUIZ MILIDIÚ† AND EDUARDO SANY LABER‡

Abstract. Given an alphabet $\{a_1, \ldots, a_n\}$ with the corresponding list of weights $[w_1, \ldots, w_n]$, and a number $L \geq \lceil \log n \rceil$, we introduce the WARM-UP algorithm, a Lagrangian algorithm for constructing suboptimal length restricted prefix codes. Two implementations of the algorithm are proposed. The first one has time complexity $O(n \log n + n \log \overline{w})$, where $\overline{w}$ is the highest presented weight. The second one runs in $O(nL \log(n/L))$ time. The number of additional bits per symbol generated by WARM-UP when comparing to Huffman encoding is not greater than $1/\psi^{L-\lceil \log(n+\lceil \log n \rceil - L) \rceil - 2}$. Even though the algorithm is approximated it presents an optimal behavior for practical settings.

An important feature of the proposed algorithm is its implementation simplicity. The algorithm is basically a selected sequence of Huffman tree constructions for modified weights. The approach gives some new insights on the problem.

Key words. prefix codes, Huffman trees, Lagrangian duality

AMS subject classifications. 94A45, 94A45, 90C11

PII. S009753979731981X

1. Introduction.

An important problem in the field of coding and information theory is the variable length code problem [10]. A particular case of relevant interest is the binary prefix code problem. Given an alphabet $\Sigma = \{a_1, \ldots, a_n\}$ and a corresponding list of weights $[w_1, \ldots, w_n]$, the problem is to find a prefix code for Σ that minimizes the *weighted length* of a code string, defined to be $\sum_{i=1}^{n} w_i l_i$, where l_i is the length of the code assigned to a_i. This problem is equivalent to the following problem: given a list of weights $[w_1, \ldots, w_n]$, find a tree T that minimizes the *weighted path length* $\sum_{i=1}^{n} w_i l_i$, where l_i is the height of the ith leaf of T. T must be chosen among all full binary trees[1] with n leaves. The equivalence is due to the fact that every binary prefix code can be represented by a full binary tree [4]. If the list of weights is sorted, this problem can be solved in $O(n)$ by one of the efficient implementations of Huffman's algorithm [10, 26, 21]. Any tree constructed by Huffman's algorithm is called a Huffman tree.

In this paper we consider the binary prefix code problem with restricted maximal length, that is, for a fixed $L \geq \lceil \log n \rceil$, we must minimize $\sum_{i=1}^{n} w_i l_i$ constrained to $l_i \leq L$ for $i = 1, \ldots, n$. Gilbert [8] recommends formulating this problem when the weights w_i are inaccurately known. Choueka, Klein, and Perl [3] suggest the use of length restricted codes to reduce the external path length $\sum_{i=1}^{n} l_i$. The objective is to allow space efficient decoding of optimal prefix codes without bit-manipulation. Zobel and Moffat [29] describe the use of word-based Huffman codes for compression of large textual databases. The application allows the maximum of 32 bits for each codeword. For the cases that exceed this limitation, it is recommended to use length restricted codes.

*Received by the editors April 16, 1997; accepted for publication (in revised form) July 13, 2000; published electronically November 8, 2000.

http://www.siam.org/journals/sicomp/30-5/31981.html

†Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil (milidiu@inf.puc-rio.br).

‡Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil (laber@cos.ufrj.br).

[1]In a full binary tree each internal node has exactly two children.

Some methods can be found in the literature to solve the binary prefix code problem with restricted maximal length [9, 27, 12, 5, 1, 23]. The first polynomial algorithm is due to Garey [7]. The algorithm is based on dynamic programming, and it has an $O(n^2L)$ complexity for time and space. Larmore and Hirschberg [12] presented the Package-Merge algorithm. This is an $O(nL)$ algorithm that requires linear space. The authors reduce the original problem to a coin collector's problem, using a nodeset representation of a binary tree. Turpin and Moffat [25] discuss some aspects about the implementation of the Package-Merge algorithm. Aggarwal, Schieber, and Tokuyama [1] have used Megiddo's parametric search paradigm [14] to obtain an $O(n\sqrt{L \log n} + n \log n)$ time algorithm. Currently, the fastest strongly polynomial time algorithm for the problem is due to Schieber [23]. This algorithm also utilizes a parametric search. However, it runs in $O(n2^{O(\sqrt{\log L \log\log n})})$ time and requires $O(n)$ space. These last two algorithms are based on a reduction of the binary prefix code problem to the concave least weight subsequence problem [13].

Some approximative algorithms have also been proposed in the literature [5, 15]. Milidiú and Laber proposed the build, remove, condense, and insert (BRCI) algorithm [15, 17]. This algorithm is $1/\psi^{L-\lceil \log(n+\lceil \log n \rceil - L)\rceil - 1}$-approximative and constructs prefix codes with restricted maximum length L in $O(n)$ time if the input weights are already sorted. The algorithm requires $O(n)$ space. BRCI first constructs a Huffman tree, and next it obtains a code tree with restricted maximal height L. The second tree is obtained by removing some leaves of the Huffman tree and condensing them into a complete binary tree. The strategy employed by the ROT algorithm is similar [5]. However, it does not provide a good theoretical approximation for the most important case, where $L = O(\log n)$.

In this paper we present an approximative algorithm that constructs length restricted prefix codes. The algorithm is called WARM-UP. It is based on Lagrangian relaxation [2], a powerful technique used to solve some important problems in *combinatorial optimization*. A very simple implementation of WARM-UP runs in $O(n \log n + n \log \overline{w})$ time, where n is the number of weights and $\overline{w}$ is the highest weight. A more elaborated implementation that uses Megiddo's parametric search runs in $O(nL \log(n/L))$ time. The algorithm requires $O(n)$ space and allows a very space-economical implementation [18]. We show that the number of additional bits per symbol generated by a WARM-UP code rather than a Huffman code is not greater than $1/\psi^{L-\lceil \log(n+\lceil \log n \rceil - L)\rceil - 2}$.

The first parametric search proposed in [1] is similar to the WARM-UP approach introduced here. Nevertheless, from a Lagrangian point of view, the former approach uses only one Lagrange multiplier, whereas the later uses a varying number of nonzero Lagrange multipliers throughout the WARM-UP process. Furthermore, Aggarwal's algorithm solves a sequence of concave least weight subsequence problems, while our algorithm constructs a sequence of Huffman trees. Due to the simplicity of Huffman's procedure, the WARM-UP algorithm has a small constant factor in its $O(n \log n + n \log \overline{w})$ time complexity. Furthermore, WARM-UP can take advantage of improvements in Huffman's procedure, such as the recent ones proposed in [22, 19].

WARM-UP is quite different from the other approximative algorithms presented in the literature. First, WARM-UP is able to recognize when it gets an optimal code during its execution, while the others are not. This capability is provided by the Lagrangian duality theory. Second, WARM-UP constructs optimal codes in almost all practical settings. This behavior does not happen with the algorithms proposed in [5, 15]. Although this fact is based on empirical results, it can be theoretically

explained somehow. Actually, the codes obtained by the WARM-UP algorithm are optimal codes either for the length restricted problem or for a related problem, where some codeword lengths cannot be longer than L and some others cannot be longer than $L-1$. This fact gives some insight on why WARM-UP is able to construct optimal codes so frequently. Hence, WARM-UP allows as simple implementations as the other approximate algorithms, and it is also able to produce length restricted codes with compression power similar to those produced by the exact algorithms.

The paper is organized as follows. In section 2, we give an integer programming formulation to the problem and develop the theoretical background needed to justify our algorithm. In section 3, we address the special case of constructing a Huffman tree with height exactly equal to L, given that the Huffman tree with maximum height has height greater than L, and the Huffman tree with minimum height has height smaller than L. In section 4, we introduce the WARM-UP algorithm. In section 5, we present a strong polynomial implementation for WARM-UP. In section 6, we analyze the approximation of the algorithm. In section 7, we present some experimental results. Finally, in section 8, we comment on our findings.

2. The Lagrangian approach. Our approach is based on an integer programming formulation to the problem. Given a sequence of n integer weights $w_1 \leq \cdots \leq w_n$ and a length $L \geq \lceil \log n \rceil$, the problem is to find a sequence of n integers $l_1, \ldots, l_n$ that minimizes

$$\sum_{i=1}^{n} w_i.l_i,$$

constrained to

$$\begin{cases} l_i \leq L, \\ \sum_{i=1}^{n} 2^{-l_i} = 1, & i = 1, \ldots, n. \\ l_i \in N, \end{cases} \tag{2.1}$$

We call this problem P_L. The restriction $\sum_{i=1}^{n} 2^{-l_i} = 1$ is satisfied if and only if $l_1, \ldots, l_n$ are the heights of the leaves of a full binary tree with n leaves [4]. In mathematical programming terms, we say that P_L is a primal problem [20].

Since $w_1 \leq \cdots \leq w_n$, there is a solution where $l_1 \geq \cdots \geq l_n$. Throughout this work we use this fact. For convenience, we use S_n to denote the points with integer coordinates that satisfy $\sum_{i=1}^{n} 2^{-l_i} = 1$, and we use $h(T)$ to denote the height of a tree T.

2.1. The Lagrangian relaxation. Let $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_n)$, with $\lambda_i \geq 0$ for $i = 1, \ldots, n$. The Lagrangian function associated to problem P_L is given by

$$L(\boldsymbol{l}, \boldsymbol{\lambda}) = \sum_{i=1}^{n} w_i.l_i + \sum_{i=1}^{n} \lambda_i.(l_i - L). \tag{2.2}$$

The values λ_i are called Lagrangian multipliers. A point $(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$ is a saddle point of the Lagrangian function $L(\boldsymbol{l}, \boldsymbol{\lambda})$ if and only if

(a) $L(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}}) \leq L(\boldsymbol{l}, \overline{\boldsymbol{\lambda}})$ for all $\boldsymbol{l} \in S_n$,
(b) $\overline{l_i} \leq L$ for $i = 1, \ldots, n$,
(c) $\overline{\boldsymbol{\lambda}}_i.(\overline{l_i} - L) = 0$ for $i = 1, \ldots, n$.

It is a well-known result [20] that if $(\bar{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$ is a saddle point of the Lagrangian function $L(\boldsymbol{l}, \boldsymbol{\lambda})$, then $\bar{\boldsymbol{l}}$ is a global optimum to the primal problem, in this case to problem P_L.

Now, we can establish a sufficient condition for the sequence of integers $l_1, \ldots, l_n$ to be a solution to problem P_L.

THEOREM 2.1. *Let* $\overline{\boldsymbol{\lambda}} = (\overline{\lambda_1}, \ldots, \overline{\lambda_n})$, *with* $\overline{\lambda_i} \geq 0$ *for* $i = 1, \ldots, n$. *Let also* $\bar{\boldsymbol{l}} = (\bar{l_1}, \ldots, \bar{l_n})$ *be the heights of the leaves of a Huffman tree for the modified list of weights given by* $w_1 + \overline{\lambda_1}, \ldots, w_n + \overline{\lambda_n}$. *If*

$$\begin{cases} \bar{l}_i \leq L, \\ \bar{l}_i = L \text{ when } \overline{\lambda_i} \neq 0 \end{cases}$$

for $i = 1, \ldots, n$, *then* $\bar{\boldsymbol{l}}$ *solves problem* P_L.

Proof. We show that $(\bar{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$ is a saddle point of the Lagrangian function $L(\boldsymbol{l}, \boldsymbol{\lambda})$ by proving conditions (a)–(c).

(a) Since $\bar{\boldsymbol{l}}$ corresponds to the heights of the leaves of a Huffman tree for the weights $w_1 + \overline{\lambda_1}, \ldots, w_n + \overline{\lambda_n}$, it follows that

$$\sum_{i=1}^{n} (w_i + \overline{\lambda}_i).\bar{l}_i \leq \sum_{i=1}^{n} (w_i + \overline{\lambda}_i).l_i \quad \text{for all } \boldsymbol{l} \in S_n.$$

Subtracting the expression $\sum_{i=1}^{n} \overline{\lambda}_i.L$ from both sides of the inequality above, we obtain that

$$\sum_{i=1}^{n} w_i.\bar{l}_i + \sum_{i=1}^{n} \overline{\lambda}_i(\bar{l}_i - L) \leq \sum_{i=1}^{n} w_i.l_i + \sum_{i=1}^{n} \overline{\lambda}_i(l_i - L),$$

and therefore,

$$L(\bar{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}}) \leq L(\boldsymbol{l}, \overline{\boldsymbol{\lambda}}) \text{ for all } \boldsymbol{l} \in S_n.$$

Conditions (b) and (c) follow immediately from the theorem hypothesis. This result shows that $\bar{\boldsymbol{l}}$ is a global optimum to problem P_L. ∎

2.2. Finding the multipliers. Theorem 2.1 suggests a simple way to solve problem P_L. Find multipliers $\lambda_1, \ldots, \lambda_n$ such that the Huffman tree for the modified weights $w_1 + \lambda_1, \ldots, w_n + \lambda_n$ have height equal to L, and all the leaves associated to the effectively modified weights $w_i + \lambda_i$, with $\lambda_i > 0$, are arranged at height L. This Huffman tree is an optimal prefix code tree with height restriction L.

As an example, let us consider the list of weights $W = [1, 1, 2, 3, 5, 8, 13]$. Figure 1 (a) shows a Huffman tree for this list. Suppose that we add 0.5 to the two lowest weights of the given list. The tree in Figure 1 (b) is a Huffman tree for this modified list. Since all the leaves with modified weights are at height 4, then the tree in Figure 1 (b) is an optimal code tree with restricted maximal height 4 for the original list of weights W.

A natural question appears: *how to choose these multipliers?*

Our approach to choose these multipliers is to select a value x, with $w_1 < x < w_n$, and set

$$\lambda_i = \max\{0, x - w_i\}.$$

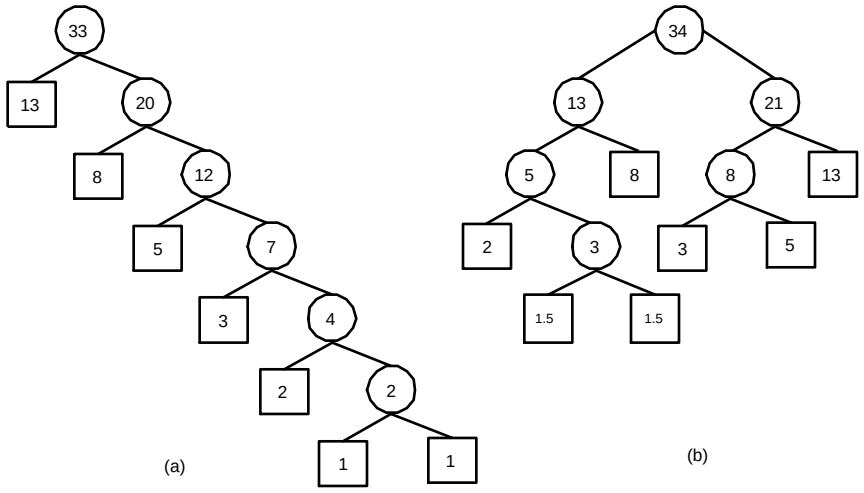


FIG. 1. (a) *A Huffman tree for the list of weights* $[1, 1, 2, 3, 5, 8, 13]$. (b) *An optimal tree with maximal restricted height* 4 *for the same list.*

Next, we use a test to verify if the chosen set of multipliers, defined by x, is adequate. Let $\boldsymbol{w}^x$ be the new list of weights obtained through this perturbation. Observe that $\boldsymbol{w}^x = (\max\{w_1, x\}, \ldots, \max\{w_n, x\})$. Let T_x be a Huffman tree for the list $\boldsymbol{w}^x$. We say that the leaves of T_x with weight x are the *warmed leaves.* A tree T_x has a useful property, given by the following lemma.

LEMMA 2.2. *If $h(T_x) = L$, then all the warmed leaves of T_x are either at height L or $L-1$ in T_x.*

Proof. Since x is the smallest weight of the list $\boldsymbol{w}^x$ and $h(T_x) = L$, then there is a warmed leaf arranged at height L. On the other hand, leaves with equal weights are always arranged in consecutive heights in a Huffman tree [28]. Hence, all the warmed leaves are either at height L or $L-1$ in T_x. □

If $h(T_x) = L$, then we are close to the conditions of Theorem 2.1. Actually, we can prove that a tree T_x with height L either solves P_L or a quite related problem.

THEOREM 2.3. *Let T_x be a tree such that $h(T_x) = L$, and let n_L and n_{L-1} be, respectively, the number of leaves at heights L and $L-1$ in T_x. Moreover, let $\overline{\boldsymbol{l}} = (\overline{l_1} \geq \cdots \geq \overline{l_n})$ be the heights of the leaves of T_x. We have the following.*

(a) *If $n_{L-1} = 0$, then $\overline{\boldsymbol{l}}$ solves the problem P_L.*

(b) *If $n_{L-1} > 0$, then $\overline{\boldsymbol{l}}$ solves the problem P_L with the additional list of constraints*

$$l_i \leq L - 1 \text{ for } i = n_L + 1, \ldots, n_L + n_{L-1}.$$

Proof. (a) Lemma 2.2 ensures that all warmed leaves can only be at heights L and $L-1$ in a T_x tree with height L. Since $n_{L-1} = 0$, it follows that all the warmed leaves are at height L. Hence, it follows from Theorem 2.1 that $\overline{\boldsymbol{l}}$ solves the problem P_L.

(b) The proof is analogous to that of Theorem 2.1. It suffices to prove that $(\overline{\boldsymbol{l}}, \overline{\boldsymbol{\lambda}})$, where $\overline{\lambda_i} = \max\{0, x - w_i\}$ for $i = 1, \ldots, n$, is a saddle point of the Lagrangian function associated to P_L with the additional list of constraints

$$l_i \leq L - 1 \text{ for } i = n_L + 1, \ldots, n_L + n_{L-1}. \qquad \square$$

We have given some insight explaining why a tree T_x with height L induces very good codes. One question arises: *how to select x?*

Let us define T_x^- as a Huffman tree with minimum height for the list $\boldsymbol{w}^x$, and T_x^+ as a Huffman tree with maximum height for the same list. Schwartz [24] shows how to construct both of these trees. The following theorem shows that if we increase the value of the parameter x, then the heights of the warmed leaves of the tree T_x cannot increase. In particular, the height of T_x cannot increase. This monotonicity suggests that we can determine an adequate value for x through a convenient binary search.

THEOREM 2.4. *Let s, t be two real numbers, with $0 < s < t$. Furthermore, let $\overline{\boldsymbol{l}} = (\overline{l_1} \geq \cdots \geq \overline{l_n})$ and $\boldsymbol{l}' = (l'_1 \geq \cdots \geq l'_n)$ be, respectively, the heights of the leaves of the trees T_s^- and T_t^+. Then*

$$l'_i \leq \overline{l_i}$$

for $i = 1, \ldots, k$, where k is the largest value such that $\max\{s, w_k\} = s$.

Proof. The optimality of $\overline{\boldsymbol{l}}$ and $\boldsymbol{l}'$ implies that

$$\sum_{i=1}^{n} \max\{w_i, s\}.(l'_i - \overline{l_i}) \geq 0, \tag{2.3}$$

$$\sum_{i=1}^{n} \max\{w_i, t\}.(\overline{l_i} - l'_i) \geq 0. \tag{2.4}$$

Let k' be the greatest integer value such that $\max\{t, w_k\} = t$. Adding inequalities (2.3) and (2.4), we obtain that

$$(t - s).\sum_{i=1}^{k}(\overline{l_i} - l'_i) + \sum_{i=k+1}^{k'} (t - w_i)(\overline{l}_i - l'_i) \geq 0. \tag{2.5}$$

Since we have k' equal weights in the tree T_t^+, it follows that $l'_1 \geq l'_i \geq l'_1 - 1$ for $i = 1, \ldots, k'$. Similarly, we have that $\overline{l_1} \geq \overline{l_i} \geq \overline{l_1} - 1$ for $i = 1, \ldots, k$.

Let us assume that $l'_i > \overline{l_i}$ for some $i \leq k$. We can show that this assumption implies that $l'_j \geq l_j$ for $j = 1, \ldots, k'$. In effect, if $j > i$, we have that $l'_j \geq l'_i - 1 \geq l_i \geq l_j$. On the other hand, if $j < i$, then $l'_j \geq l'_i \geq \overline{l_i} + 1 \geq \overline{l_j}$. Since $l'_j \geq \overline{l_j}$ for $j = 1, \ldots, k'$ and $l'_i > l_i$, then the inequality (2.5) is not satisfied. Since this contradiction was generated by the assumption that $l'_i > \overline{l_i}$ for some $i \leq k$, then we have that $l'_i \leq \overline{l_i}$ for $i = 1, \ldots, k$. ∎

COROLLARY 2.5 (WARM-UP theorem). *Let s, t be two real numbers with $0 < s < t$. Then we have that $h(T_t^+) \leq h(T_s^-)$.*

Proof. It follows immediately from Theorem 2.4, since $l'_1 = h(T_t^+)$ and $\overline{l_1} = h(T_s^-)$. ∎

We note that there may exist more than one value of x such that $h(T_x) = L$. Actually, this is the usual case. Hence, one could ask which value of x generates the best tree? The next theorem shows that $x^* = \min\{x | h(T_x^-) \leq L\}$ is the choice that produces the best tree.

THEOREM 2.6. *Let x^* be the smallest value of x such that $h(T_x^-) \leq L$, and let y be such that $h(T_y) \leq L$. Moreover, let $\overline{\boldsymbol{l}} = (\overline{l_1}, \ldots, \overline{l_n})$ and $\boldsymbol{l}' = (l'_1, \ldots, l'_n)$ be, respectively, the heights of the leaves of T_{x^*} and T_y. Hence*

$$\sum_{i=1}^{n} w_i \overline{l_i} \leq \sum_{i=1}^{n} w_i l'_i.$$

Proof. Let $\overline{k} = \max\{k | w_k < x^*\}$. Let $\overline{P}$ be the problem P_L augmented by the list of constraints $l_i \leq \overline{l_i}$ for $i = 1, \ldots, \overline{k}$. By using a saddle point argument one can show that $\overline{\boldsymbol{l}}$ solves the problem $\overline{P}$. Similarly, $\boldsymbol{l}'$ solves the problem P_L augmented by the list of constraints $l_i \leq l'_i$ for $i = 1, \ldots, k'$, where $k' = \max\{k | w_k < y\}$. We call this second problem P'.

Since $x^* < y$, it follows from Lemma 2.4 that $\overline{l_i} \geq l'_i$ for $i = 1, \ldots, \overline{k}$. Moreover, $\overline{k} \leq k'$. Hence, $\overline{P}$ is a relaxation of P', and as a consequence,

$$\sum_{i=1}^{n} w_i \overline{l_i} \leq \sum_{i=1}^{n} w_i l'_i. \qquad \square$$

2.3. Well-numbered trees. The WARM-UP theorem allows us to search for an appropriate value of x by performing a binary search on the interval (w_1, w_n). If for a given x, $h(T_x^-) > L$, then we must increase x. On the other hand, if $h(T_x^+) < L$, we must decrease x. The special case where $h(T_x^-) < L$ and $h(T_x^+) > L$ is addressed in the next section.

Now, we show that there exists x such that $h(T_x) = L$. In order to show this fact, we define the concept of *well-numbered trees.*

DEFINITION 2.7. *A weighted full binary tree is well numbered if and only if the following hold.*

(a) *To each node is assigned a different integer number i, with $1 \leq i \leq 2n-1$.*

(b) *A parent node weight is equal to the sum of the weights of its two children nodes.*

(c) *If $i > j$, then the weight of the node with number i is not smaller than the weight of the node with number j.*

(d) *Let $i > j$. If the node with number i is son of a node with number i', and the node with number j is son of a node with number j', then $i' \geq j'$.*

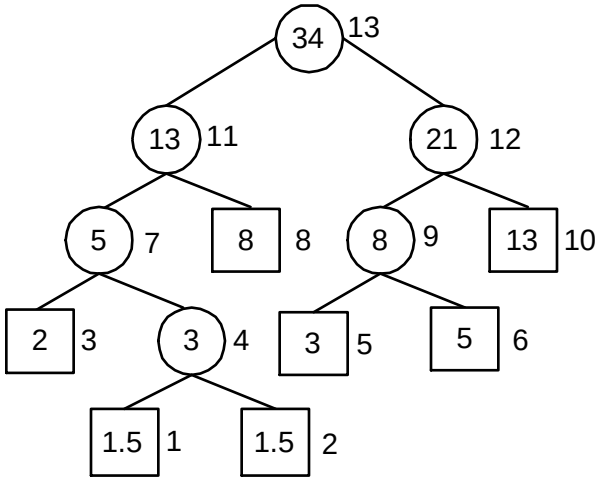


FIG. 2. *A well-numbered tree.*

The tree in Figure 2 is well numbered. Each number is drawn at the right side of the node. Now, we state three key properties concerning the well-numbered trees.

PROPOSITION 2.8. *Let $y_1, \ldots, y_{2n-1}$ be the nodes of a well-numbered tree T. Furthermore, let y_{i_1} and y_{i_2}, with $i_2 > i_1$, be the two children of node y_i, and let y_{i_3}*

and y_{i_4}, with $i_4 > i_3$, be the two children of node y_{i+1}. Under these assumptions, we have that $i_2 = i_1 + 1, i_3 = i_2 + 1$, and $i_4 = i_3 + 1$.

Proof. First, we prove that $i_2 = i_1 + 1$. Assume that $i_2 > i_1 + 1$. Let z be the node with number $i_1 + 1$. It follows that the parent of z has a number not smaller than the parent of y_{i_1} and a number not greater than the parent of y_{i_2}. Hence, the parent of z has number i, which contradicts the fact that each internal node has only two children. The proof that $i_4 = i_3 + 1$ is analogous and the proof that $i_3 = i_2 + 1$ follows by the same arguments already used. □

PROPOSITION 2.9. *Let $y_1, \ldots, y_{2n-1}$ be the nodes of a well-numbered tree T with corresponding heights $l_1, \ldots, l_{2n-1}$. Then, $0 \leq l_i - l_{i+1} \leq 1$ for $i = 1, \ldots, 2n-2$.*

Proof. First, let us prove that $l_{i+1} \leq l_i$ for $i = 1, \ldots, 2n-2$. It follows from properties (b) and (c) that y_{2n-1} is the heaviest node and hence the root of T. Hence, the result holds for $i = 2n-2$. Let us assume by induction that the result is valid for $i > p$, and then we prove it for $i = p$. Observe that the parent of y_p has a number smaller than or equal to the number of the parent of y_{p+1}. It follows by induction that the parent of y_p has a height not lower than the height of the parent of y_{p+1}. Since a parent's height is exactly one unit lower than the height of its children, the result holds for $i = p$.

Now, suppose that $l_i > l_{i+1} + 1$ for some i. In this case, the parent $y_{i'}$ of y_i has height $l_{i'} = l_i - 1$. Therefore, $i' \geq i + 1$ and $l_{i'} = l_i - 1 > l_{i+1}$, which contradicts the inequality $l_{i+1} \leq l_i$ for $i = 1, \ldots, 2n-2$, that we have already proved. Hence, $l_i \leq l_{i+1} + 1$. □

PROPOSITION 2.10. *If T is well numbered, then T is a Huffman tree.*

Proof. It follows from Proposition 2.8 that the tree T has the *sibling property*, defined by Gallager [6]. Therefore, T is a Huffman tree. □

2.4. The existence of a suitable x. Now, we are ready to prove that there is a value x, such that $h(T_x) = L$. This proof gives an insight about the effects in the tree generated by changing the parameter x.

THEOREM 2.11. *If $h(T^-_{w_1}) > L \geq \lceil \log n \rceil$, then there is a rational number $x=p/q$, with $q \leq n$ and $w_1 < x < w_n$, such that $h(T_x) = L$.*

Proof. Observe that $T^-_{w_1}$ is the Huffman tree for the original list of weights, since $\max\{w_1, w_i\} = w_i$ for $i = 1, \ldots, n$. Let us assume that $T^-_{w_1}$ is well numbered. In section 3 we show how to construct such a tree. We use y_i to denote the node with number i, ϕ_i to denote the weight of node y_i, and n_i to denote the number of descendent leaves of node y_i with weight x. We define $n_i = 1$ when y_i is a leaf with weight equal to x. We also define δ_i by $\delta_i = (\phi_{i+1} - \phi_i)/(n_i - n_{i+1})$.

The well-numbered tree in Figure 2 is a T_x tree, with $x = 1.5$. For this tree, we have $n_1 = n_2 = 1$, $n_4 = n_7 = n_{11} = n_{13} = 2$, and the others nodes have $n_i = 0$. We also have that $\phi_1 = 0$, $\phi_2 = 0.5$, $\phi_4 = 0$, and so on. Now, we prove that one of the trees obtained by the following algorithm is a T_x tree with $h(T_x) = L$.

```
x ← w_1
T ← T_x^-
Repeat
  Let I = {i | i = arg min {δ_i | n_i > n_{i+1}}}
  Define i* to be the smallest integer in I
  For each leaf y_j of T with φ_j = x do
      φ_j ← x + δ_{i*}
  Update all internal node weights to assure that a parent weight
  is equal to the sum of its children's weights.
```

`In` T `interchange`[2] `the subtrees with roots` y_{i^*} `and` y_{i^*+1}
`Assign number` i^* `to` y_{i^*+1} `and number` i^*+1 `to` y_{i^*}
$x \leftarrow x + \delta_{i^*}$
`Until` $I = \emptyset$

We divide the algorithm correctness proof into six parts.

(Part a) *All trees constructed by the algorithm are well numbered.*

Items (a) and (b) of Definition 2.7 are clearly respected by all trees T. Hence, we need to prove that items (c) and (d) are also respected.

For the initial tree, the result follows from the hypothesis under $T^-_{w_1}$. When the algorithm adds δ_{i^*} to the leaves with weight x, and after updating the internal node weights, we still have $\phi_j \leq \phi_{j+1}$ for $j = 1, \ldots, 2n-2$, since δ_{i^*} is the minimum value necessary to produce a tie between the weights of two nodes with consecutive numbers. Therefore, items (c) and (d) of Definition 2.7 hold. On the other hand, interchanging the two chosen subtrees and also their corresponding root node numbers will also ensure items (c) and (d).

(Part b) *In each cycle, the tree T is a T_x tree for the current value of x.*

It follows from (Part a) and Proposition 2.8 that each tree T is a Huffman tree. Thus, we must prove that the weights of the leaves of T are given by $\max\{w_1, x\}, \ldots, \max\{w_n, x\}$ for the current value of x. The leaves of the initial tree have these weights. Let T^m and x^m be, respectively, the tree T at the beginning of the mth cycle and the value of x at the beginning of this same cycle. We assume that the weights of the leaves of T^m are given by $\max\{w_1, x^m\}, \ldots, \max\{w_n, x^m\}$, and then we prove that the weights of the leaves of T^{m+1} are given by $\max\{w_1, x^{m+1}\}, \ldots, \max\{w_n, x^{m+1}\}$.

We can suppose that $w_i \leq x^m < w_{i+1}$. If $x^{m+1} \leq w_{i+1}$, then the result clearly holds. On the other hand, if $x^{m+1} > w_{i+1}$, the result doesn't hold, since T^{m+1} would have a leaf with weight $w_{i+1} < x^{m+1}$. However, if $x^{m+1} > w_{i+1}$, then the integer i^* chosen during the mth cycle is such that $\delta_{i^*} = (x^{m+1} - x^m) > (w_{i+1} - x^m)$. Let y_j be the node of tree T^m such that $\phi_{j+1} = w_{i+1}$. Since $\phi_j \geq x_m$, we have that $\delta_j \leq w_{i+1} - x^m$. It follows that i^* would not belong to I, and, as a consequence, the case $x^{m+1} > w_{i+1}$ is not possible.

(Part c) *Node y_{i^*+1} is always a leaf.*

First, let us assume that y_{i^*} is a leaf and that y_{i^*+1} is an internal node. In this case, $n_{i^*} = 1$; otherwise, i^* would not have been selected because $n_{i^*} = 0 \leq n_{i^*+1}$. As a consequence we get that $\phi_{i^*} = x$. The two children of y_{i^*+1} have numbers no greater than i^*. Therefore, these children must have weights x; otherwise, we contradict either item (c) of Definition 2.7 or the result of (Part b). Hence, $n_{i^*} = 1 < 2 = n_{i^*+1}$ and i^* would not belong to set I.

Now, let us assume that y_{i^*} and y_{i^*+1} are both internal nodes. Let $y_{i'}$, $y_{i'+1}$ be the sons of y_{i^*}, and let $y_{i'+2}$, $y_{i'+3}$ be the sons of y_{i^*+1}. It follows from the definition of δ_i that $\phi_{i'} + \phi_{i'+1} + (n_{i'} + n_{i'+1}).\delta_{i^*} = \phi_{i^*} + \delta_{i^*}.n_{i^*} = \phi_{i^*+1} + \delta_{i^*}.n_{i^*+1} = \phi_{i'+2} + \phi_{i'+3} + (n_{i'+2} + n_{i'+3}).\delta_{i^*}$. We also have that $\phi_{i'} + \delta_{i^*}.n_{i'} \leq \phi_{i'+1} + \delta_{i^*}.n_{i'+1} \leq \phi_{i'+2} + \delta_{i^*}.n_{i'+2} \leq \phi_{i'+3} + \delta_{i^*}.n_{i'+3}$; otherwise, there would be a node y_j with $n_j > n_{j+1}$ such that $\delta_j < \delta_{i^*}$. To satisfy the two previous expressions, we must have that $\phi_{i'} + \delta_{i^*}.n_{i'} = \phi_{i'+1} + \delta_{i^*}.n_{i'+1} = \phi_{i'+2} + \delta_{i^*}.n_{i'+2} = \phi_{i'+3} + \delta_{i^*}.n_{i'+3}$. Since $n_{i^*} > n_{i^*+1}$, $n_{i^*} = n_{i'} + n_{i'+1}$, and $n_{i^*+1} = n_{i'+2} + n_{i'+3}$, then either $n_{i'} > n_{i'+1}$ or $n_{i'+1} > n_{i'+2}$ or $n_{i'+2} > n_{i'+3}$. Hence, there is an index j with $i' \leq j < i'+3 \leq i$, such that $n_j > n_{j+1}$ and $\delta_{i^*} = \delta_j$. This contradiction establishes the result.

[2]When two subtrees are interchanged, the second subtree becomes a child of the root parent of the first subtree and vice-versa.

(Part d) *Let M be the sum of the numbers assigned to the leaves of T. The five statements below are true.*

(i) $(n^2 - n)/2 \leq M \leq (3n^2 - n)/2$.

(ii) *If the algorithm halts, then $M = (n^2 - n)/2$.*

(iii) *After each cycle the value of M either remains the same or decreases by exactly one unit.*

(iv) *In no more than $n^2 + n$ cycles, the value of M satisfies $M = (n^2 - n)/2$.*

(v) *If $M = (n^2 - n)/2$, then $h(T) = \lceil \log n \rceil$.*

Demonstrations. (i) There are n leaves in T, each leaf assigned a different number. Thus, $M \geq \sum_{j=1}^{n} j = (n^2 - n)/2$ and $M \leq \sum_{j=n}^{2n-1} j = (3n^2 - n)/2$.

(ii) In order to prove this item, we just need to show that $I \neq \emptyset$ whenever $M > (n^2 - n)/2$. If the sum of the numbers assigned to the leaves of T is greater than $n(n-1)/2$, then there is a leaf with number k, such that $k > n$. Since each number in the set $\{1, \ldots, 2n-1\}$ is a assigned to a node, then there is a node with number smaller than n.

It follows from item (d) of Definition 2.7 that the parent of y_1 is the internal node with minimum number. Let m be the number of y_1's parent. We have that $m < n < k$ and $n_m \geq 1$. Since $\phi_k \geq \phi_m > x$ and y_k is a leaf, then $n_k = 0$. As $n_m > n_k$, there exists an index i with $m \leq i < k$, such that $n_i > n_{i+1}$. Hence, the set I is not empty.

(iii) It follows from (Part c) that y_{i^*+1} is a leaf. If y_{i^*} and y_{i^*+1} are leaves, then the value of M remains the same. On the other hand, if y_{i^*} is an internal node, then the value of M decreases by exactly one unit, since we interchange the numbers.

(iv) First, we show that in no more than n cycles y_{i^*} is a leaf. If y_{i^*} is a leaf, then $\phi_{i^*} = x$; otherwise, $n_{i^*} = 0 \leq n_{i^*+1}$. Furthermore, $\phi_{i^*+1} > x$; otherwise $n_{i^*+1} = 1 = n_{i^*}$. In this case the value of x is updated to $x + \phi_{i^*+1} - \phi_{i^*} = \phi_{i^*+1}$. Hence, the number of leaves with weight x is increased by one unit. Therefore, this case can happen in no more than n cycles, otherwise we would obtain a tree with more than n leaves.

When y_{i^*} is an internal node, the value of M decreases by one unit. Since $(n^2 - n)/2 \leq M \leq (3n^2 - n)/2$, this second case can happen no more than $(3n^2 - n)/2 - (n^2 - n)/2 = n^2$. Therefore, in no more than $n^2 + n$ the value of M satisfies $M = (n^2 - n)/2$.

(v) If $M = (n^2 - n)/2$, then the numbers of all the leaves are not greater than n. Let l_j be the height of a generic leaf y_j. We have that the parent of y_1 has height $l_1 - 1$ and a number greater than n. It follows from Proposition 2.9 that the height of any leaf y_j satisfies $l_1 - 1 \leq l_j \leq l_1$. Therefore, all the leaves of T are arranged in consecutive heights. Hence, $h(T) = \lceil \log n \rceil$.

(Part e) *On each cycle the height of T either remains the same or decreases by one unit.*

If y_{i^*} has the same height as y_{i^*+1}, then the interchange does not modify $h(T)$. On the other hand, if the height of y_{i^*+1} is smaller than that of y_{i^*} by one unit, then the interchange either decreases $h(T)$ by one unit or does not modify $h(T)$, because y_{i^*+1} is a leaf.

(Part f) *The variable x assumes only rational values of the form p/q with $p, q \in N$ and $1 \leq q \leq n$ during the algorithm execution.*

Let x_m be the value of x at the beginning of the mth cycle. The first value of x is w_1, so x_1 is an integer. We observe that in the next assignment x assumes the value $x_m + ((\phi_{i^*+1} - \phi_{i^*})/n_{i^*}) = x_m + (\phi_{i^*+1} - A - x_m.n_{i^*})/n_{i^*} = (\phi_{i^*+1} - A)/n_{i^*}$, where A is the sum of the weights of the leaves that descend from y_{i^*} with corresponding weight values different from x_m. Since $\phi_{i^*+1} - A$ and n_{i^*} are positive integers and

$1 \leq n_{i^*} \leq n$, we get that x assumes only values in the form p/q with $p, q \in$N and q $\leq$ n.

Algorithm Correctness. It follows from (Part d) that the algorithm can only stop if $M = (n^2 - n)/2$. On the other hand, if $M = (n^2 - n)/2$, then $h(T) = \lceil \log n \rceil$. Since the first tree has height greater than L and after each cycle the height of T does not decrease by more than one unit, then the algorithm must obtain a tree T_x with height exactly L in no more than $n^2 + n$. Furthermore, (Part f) ensures that x is a rational number p/q with $q \leq n$. □

This result allows us to stop searching for a value x when the length of the search interval is smaller than $1/n^2$, since there is only one rational with denominator less than n in such an interval, and it can be easily determined.

3. The procedure Ties. Before describing our algorithm we must first consider a special case, where

$$h(T_x^-) < L \text{ and } h(T_x^+) > L.$$

For the same list of weights it is possible that more than one Huffman tree exists. This happens because when constructing the tree, Huffman's algorithm faces alternative choices on how to choose a node when there is more than one node with minimal weight.

Searching for an adequate value for x, we can face a situation where $h(T_x^-) < L$ and $h(T_x^+) > L$. If we increase x, then the new tree gets a height smaller than L. On the other hand, if we decrease the value of x, then we obtain a tree with height greater than L. Corollary 2.5 guarantees this fact. Since we have already shown that there is always a value x such that $h(T_x) = L$, then the current value of x is the one we are looking for. In this case, we can construct a Huffman tree with height exactly L for the current list of weights.

The trees in Figure 3 are T_3 trees for the weights $1, 2, 4, 6, 10, 16, 26$. The left tree is a T_3^+ tree and its height is 6, while the right tree is a T_3^- tree and its height is 4. If we have been trying to find a T_x tree with height 5, we would face the special case addressed in this section.

Let us consider the well-known implementation [26] of Huffman's algorithm that uses a stack S to store the leaves of the tree, and a queue Q to store the internal nodes. At the start, Q is empty and S contains the leaves of the tree sorted by weights, such that the leaf with lower weight is located at the top of S.

During the main step of this implementation, the weight of the leaf at the top of S is compared to the weight of the internal node at the head of Q. The node with smaller weight is selected. After the selection of two nodes, a new internal node is created. This new node is the parent of the selected nodes, and it is inserted at the tail of Q. Observe that whenever a node is selected, it is removed either from the top of S or from the head of Q. The main step is executed until S is empty and Q contains only one node.

We say that a tie occurs whenever a leaf at the top of S has the same weight as an internal node at the head of Q. If we choose a leaf whenever a tie occurs, we obtain a Huffman tree with minimum height. On the other hand, if we always choose an internal node, we get a Huffman tree with maximum height [24].

3.1. Procedure Ties description. Let T^- be a Huffman tree with minimum height, and let T^+ be a Huffman tree with maximum height. Both of these trees have the same list of weights. The procedure Ties constructs a Huffman tree with height L when $h(T^-) < L$ and $h(T^+) > L$.

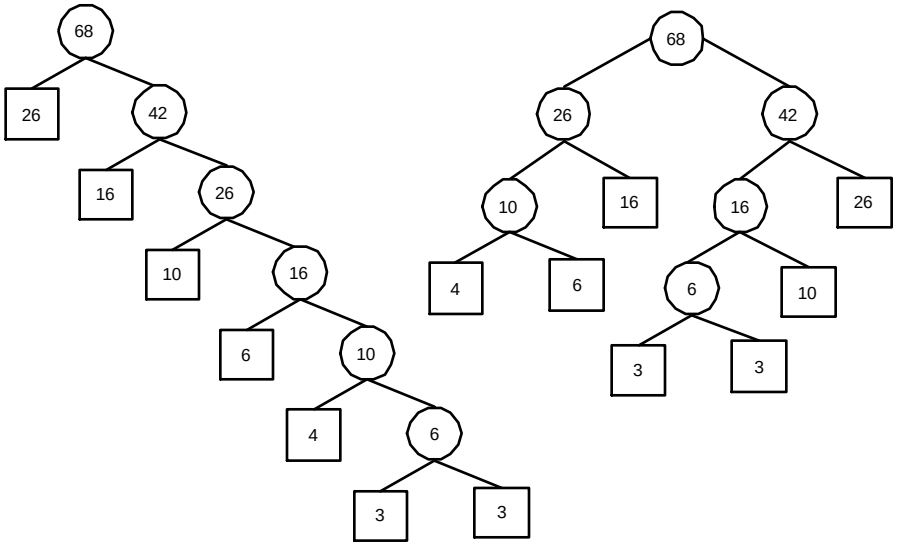


FIG. 3. *A T_3^+ tree and a T_3^- tree for the weights* $1, 2, 4, 6, 10, 16, 26$.

The procedure performs a binary search on the integers $1, \ldots, 2n-2$. For each selected integer k, it constructs a Huffman tree T^k using the implementation described above with an additional refinement: *choose a leaf on each one of the first* `k` *ties and then choose only internal nodes on all subsequent ties.* If it obtains the tree with height L, then the procedure ends. If the obtained tree has height lower than L, then the binary search proceeds with a decreased value of k. Otherwise, the binary search proceeds with an increased value of k.

3.2. Ties correctness. Let us suppose that procedure Ties produces also a numbering of the nodes as follows. Number 1 is assigned to the first node removed either from the top of S or from the head of Q. Number 2 is assigned to the second removed node, and so on. It is easy to see that the constructed tree is well numbered.

Now, let y_i denote the node with number i in T^k, with corresponding weight and height given by ϕ_i and l_i, respectively. Let us also assume that p is the number of the internal node selected when the $(k+1)$th tie occurs, and that q is the number of the first leaf selected after node y_p. Observe that $\phi_p = \phi_{p+1} = \cdots = \phi_q$ since the $(k+1)$th tie was between nodes y_p and y_q and T^k is well numbered. Furthermore, it follows from Proposition 2.9 that $l_p \geq l_{p+1} \cdots \geq l_q \geq l_p - 1$. Under these assumptions, the following steps are performed to obtain T^{k+1} from T^k.

```
For i = q down to p + 1
  Interchange the subtree rooted at y_i with the one rooted at y_{i-1}
  Assign number i to node y_{i-1}, and number i - 1 to node y_i
Next
```

Two cases can occur when the procedure interchanges the subtrees. If the leaf y_i has the same height as the internal node y_{i-1}, then the tree height is not affected. Otherwise, y_i has a height smaller than that of y_{i-1} by exactly one unit. In this second case, the interchange either reduces the tree height by one unit or keeps it the same. Since $l_p \geq \cdots \geq l_q \geq l_p - 1$, the second case can only happen once. Therefore, we have that $h(T^{k+1}) \leq h(T^k) \leq h(T^{k+1}) + 1$.

When $h(T^{2n-1}) < L$ and $h(T^0) > L$, one can find a value k such that $h(T^k) = L$, by performing a binary search on $1, \ldots, 2n$. Hence, the procedure Ties can be used to construct a T_x tree with $h(T_x) = L$, since $h(T_x^-) < L$ and $h(T_x^+) > L$.

3.3. Ties complexity. The binary search tries no more than $\lceil \log(2n-2) \rceil$ values for k. As we spend $O(n)$ to construct an optimal tree if the frequencies are already sorted, we have that the procedure spends an $O(n \log n)$ time in the worst case.

4. The WARM-UP algorithm. In this section, we describe the WARM-UP algorithm. A pseudocode for it is given in Figure 4. In Phase 0, the algorithm constructs a Huffman tree for the given list of weights. If this tree has a height not greater than L, then the algorithm halts since this tree is an optimal code tree with restricted maximal height L. Otherwise, the algorithm goes to Phase 1.

In Phase 1, WARM-UP executes a binary search, in the interval $(w_1, w_n]$, looking for a value x such that all the leaves with weight x in the tree T_x are at level L. If this happens, the algorithm halts, since this tree is known to be optimal from Theorem 2.1. In the pseudocode, "Is Optimal" accounts for the checking of this condition. If the current tree is not optimal, then the algorithm compares its height against L and uses this information to update the interval search limits and the value of x.

The binary search also stops when the length of the searching interval is smaller than $1/n^2$. Whenever this happens, the algorithm finds the unique rational number p/q with $q \leq n$ in the last searching interval. Setting $x = p/q$, the algorithm builds a T_x tree with height L. This last fact is assured by Theorem 2.11.

4.1. Complexity. In Phase 0, the algorithm spends an $O(n)$ time to construct the Huffman tree since the weights are already sorted.

In Phase 1, the algorithm can spend an $O(n \log(n^2.w_n))$ time in the worst case. That is true because in every cycle the length of the search interval is divided by 2. Hence, in no more than $\lceil \log n^2.(w_n - w_1) \rceil$ steps we have $\texttt{sup} - \texttt{inf} \leq 1/n^2$. Moreover, during each cycle we spend an $O(n)$ time to determine both a Huffman tree with minimum height and a Huffman tree with maximum height for the current list of weights. In Phase 1, the algorithm can also execute one call to procedure Ties. However, that does not affect the time complexity, since it represents an additional effort bounded by $O(n \log n)$.

If the algorithm reaches Phase 2, it spends an $O(n)$ time to find the adequate rational number p/q. Therefore, we conclude that the WARM-UP algorithm has an $O(n \log n + n \log w_n)$ worst case time complexity.

By using Moffat's runlength approach [22], the WARM-UP time complexity can be easily reduced to $O(n \log n + r \log(n/r) \log w_n)$, where r is the number of distinct weights in the given list of weights.

5. A strong polynomial time implementation. We can use Megiddo's parametric search [14] to devise a strong polynomial time implementation for WARM-UP. Instead of performing a binary search in the interval $(w_1, w_n]$, we use a parametric search to find a tree T_{x^*} with height L, where $x^* = \min\{x | h(T_x^-) \leq L\}$. Theorem 2.6 guarantees that this is the best choice for x.

The new search scheme is divided into two phases: the binary search phase and the parametric search phase. In the first phase, the algorithm performs a binary search in the list of weights $[w_1, \ldots, w_n]$. For each value of x, the algorithm checks if the height of T_x^- is greater than L or not. If the height is greater, then the value of x is increased; otherwise, it is decreased. The value of x^* is known to be in some

Phase 0: Trying a Huffman tree

If $h(T^-_{w_1}) \leq L$ **then** Return$(T^-_{w_1})$

Phase 1: Binary search

$inf \leftarrow w_1$; $sup \leftarrow w_n$; $x \leftarrow (inf + sup)/2$
While sup-inf $> 1/n^2$ **do**
  If T^-_x Is Optimal **then** Return(T^-_x)
  If $h(T^-_x) > L$ **then** $inf \leftarrow x$
  Else
    If T^+_x Is Optimal **then** Return(T^+_x)
    If $(h(T^+_x) > L$ and $h(T^-_x) < L)$ **then** Return(Ties(x))
    $sup \leftarrow x$
  End If
  $x \leftarrow (inf + sup)/2$
End While
If $h(T^+_{sup}) = L$ **then** Return(T^+_{sup})

Phase 2: Search for p/q

$q \leftarrow 1$; $p \leftarrow \lceil$q.inf$\rceil$
While ($p =$ q.inf or $p \geq$ q.sup) **do**
  $q \leftarrow q + 1$
  $p \leftarrow \lceil$q.inf$\rceil$
End While
  Return(Ties(p/q))

FIG. 4. *The WARM-UP Algorithm.*

interval $(w_{k-1}, w_k]$, where k is an integer, with $1 < k \leq n$. The objective of the first phase is to find out the value of k.

In the second phase, a Huffman tree is constructed for the list of warmed up weights $[w^*_1, \ldots, w^*_n]$, where $w^*_i = \max\{w_i, x^*\}$. At the beginning, the only information about x^* is that it belongs to the interval (inf, sup], where inf $= w_{k-1}$ and sup $= w_k$. In this phase, we construct the required Huffman tree T_{x^*}, using only an interval estimate of x^*. Phase 2 is based on a serialization of the parallel algorithm for constructing Huffman codes, proposed in [16]. Since the value of x^* is not known during the algorithm execution, it is not possible to directly compare weights that depend on x^*. Before explaining the second phase, we show how to compare such weights.

5.1. Comparing weights. Let s be a leaf, and let t be an internal node. Let us assume that the weight of s is a real number $w(s)$ and the weight of t is given by $w(t, x^*)$, a linear function of x^*, defined by $a_t x^* + b_t$, where a_t and b_t are known positive coefficients. If $a_t = 0$, then the comparison between $w(s)$ and $w(t, x^*)$ is immediate since $w(t, x^*) = b_t$. On the other hand, if $a_t > 0$, then the comparison is more elaborate since the value of $w(t, x^*)$ is not known with precision. The only information about x^* is that it belongs to a certain interval (inf, sup]. In order to

perform this parametric comparison, we need to use the concept of critical value (intersection point) [14] that we redefine for our purposes.

DEFINITION 5.1. *Let (s,t) be an ordered pair, where s is a leaf and t is an internal node. Furthermore, let $w(s)$ be the weight of s, and let $w(t,x^*) = a_t x^* + b_t$ be the weight of t. If $a_t > 0$, then the critical value γ associated to the pair (s,t) is the unique real number that solves the equation $w(s) = w(t,x^*)$; that is, $\gamma = (w(s) - b_t)/a_t$.*

As an example, if $w(s) = 17$ and $w(t,x^*) = 3x^* + 2$, then the critical value γ associated to the pair (s,t) is 5. In order to compare the weights of s and t, the Parametric WARM-UP uses the function **Compare** that we describe below.

Compare determines if either $w(s) < w(t,x^*)$ or $w(s) \geq w(t,x^*)$. In order to decide if either $w(s) < w(t,x^*)$ or $w(s) \geq w(t,x^*)$, it compares if either $\gamma < x^*$ or $\gamma \geq x^*$. If $\gamma < x^*$, then $w(s) = w(t,\gamma) < w(t,x^*)$; otherwise, $w(s) = w(t,\gamma) \geq w(t,x^*)$. If either $\gamma \leq \inf$ or $\gamma > \sup$, the comparison is straightforward. If $\inf < \gamma \leq \sup$, the function compares $h(T_\gamma^-)$ against L. If $h(T_\gamma^-) > L$, then $\gamma < x^*$; otherwise, $\gamma \geq x^*$. This last statement follows from the definition of x^* and from Corollary 2.5. The time complexity of **Compare** is $O(n)$, since it may construct a Huffman tree T_γ^-.

Compare also updates the interval (inf, sup] whenever the value of γ is in the interval (inf, sup]. Actually, if $\gamma < x^*$, the lower limit inf takes the value of γ; otherwise, the upper limit sup takes the value of γ.

5.2. The second phase. At the second phase, the algorithm maintains a list S of leaves and a queue Q of internal nodes, both sorted by the nondecreasing order of their weights. In some situations Q may contain one leaf. At the beginning Q is empty and S contains all the leaves. At each cycle, the algorithm performs the following steps.

Step 1. Compute the sum α of the two current smallest weights among the weights of the nodes in Q and S.

Step 2. Do an exponential search to determine the leaves with weights smaller than α in the list S.

Step 3. Let S' be the sublist of S with weights smaller than α. $U \leftarrow \text{Merge}(S', Q)$, $Q \leftarrow$ null, and $S \leftarrow S - S'$.

Step 4. If $|U|$ is odd, then remove the last node from U and insert it in Q.

Step 5. For $i = 1$ to $|U|/2$ do: create a parent p_i for the pair of nodes u_{2i-1} and u_{2i} in U; insert p_i in Q.

The algorithm ends if we get a T_x tree with all warmed leaves at height L during a call to **Compare**. Otherwise, it ends when the number of cycles reaches L.

At the end of the second phase the algorithm verifies if the height of T_{sup}^+ is equal to L. If that happens, it outputs the heights of the leaves of T_{sup}^+. Otherwise, the procedure **Ties** is called to construct a tree T_{sup} with height exactly L, and the algorithm outputs the leaf levels of this tree.

5.3. The merging step. All the operations but the merge are straightforward. Hence, we describe it now. At step 3, we merge S' and Q into an auxiliary list U. We can divide Q into two ordered lists. The first list Q' contains the nodes of Q whose weights depend on x^*. We observe that we do not allow nodes with equal weights in Q'. If two nodes have equal weights, we insert in Q' the one that appears first in Q. The second list Q'' is the list of the remaining nodes. For example, if $w(Q) = [2x^*, 2x^*, 33, 2x^*+7, x^*+30, x^*+30, 46, 64]$, then $w(Q') = [2x^*, 2x^*+7, x^*+30]$ and $w(Q'') = [2x^*, 33, x^*+30, 46, 64]$. In this case, $w()$ is a function that returns the node weights.

Let U' be the list obtained by merging S' and Q'. After obtaining U', the merge between Q'' and U' becomes straightforward. Hence, we should be concerned on merging S' and Q'. This merge is achieved through eight successive substeps that we describe below.

At step 3.1, we choose a suitable parameter p and then we consider the list $S'(p)$ of the nodes that appear at positions $p, 2p, \ldots, \lfloor S'(p)/p \rfloor p$ in the list S'. We observe that these positions split the list S' into $\lceil |S'|/p \rceil$ contiguous sublists of approximately equal sizes. Let $S'_1, \ldots, S'_{\lceil |S|/p \rceil}$ be these sublists. At step 3.1, we determine the critical values associated to all pairs in the cartesian product $S'(p) \times Q'$. For example, if $w(S'(p)) = [22, 26, 31]$ and $w(Q') = [2x^*, 2x^* + 7]$, then the critical values are $[11, 15/2, 13, 19/2, 31/2, 12]$.

At step 3.2, we sort the critical values obtained at step 3.1. Let $\gamma_1, \ldots, \gamma_{|Q'||S'(p)|}$ be the sorted list. At step 3.3, we perform a binary search on the sorted list to obtain an index j such that $\gamma_j < x^* \leq \gamma_{j+1}$. At step 3.4, we merge the lists $S'(p)$ and Q'. We use the fact that each critical value γ associated to a pair of nodes to be compared is such that either $\gamma \leq \gamma_j < x^*$ or $\gamma \geq \gamma_{j+1} \geq x^*$. Hence, all comparisons are made in $O(1)$.

Let q'_i be an element of Q'. We define $f(i)$ as the position of the smallest element of $S'(p)$ that is greater than or equal to q'_i. At the end of the step 3.4, the values of $f(i)$ are known. Hence, in order to know the position of q'_i in the list S', we need only to compare q'_i against the elements of the sublist $S'_{f(i)}$. At step 3.5, for $i = 1, \ldots, |Q'|$, we evaluate the critical values associated to all pairs of the cartesian product $[q'_i] \times S'_{f(i)}$. Let Γ be the list of all the critical values so obtained. We observe that $|\Gamma| \leq p|Q'|$ since $|S'_{f(i)}| \leq p$.

At step 3.6, we sort Γ obtaining a sorted list $\gamma_1, \ldots, \gamma_{|\Gamma|}$. At step 3.7, we perform a binary search on the sorted list of critical values to obtain and index j such that $\gamma_j < x^* \leq \gamma_{j+1}$.

At step 3.8, we merge S' and Q' into an auxiliary list U'. For that, we compare q'_i for $i = 1, \ldots, |Q'|$ against the elements of $S_{f(i)}$. The comparisons are made in $O(1)$ since all the critical values γ are such that $\gamma \leq \gamma_j < x^*$ or $\gamma \geq \gamma_{j+1} \geq x^*$. Finally, at step 3.9, we merge U' and Q'', obtaining an ordered list U.

5.3.1. Analysis of the merging step. Let us analyze the complexity of step 3. The most expensive steps are steps 3.2, 3.3, 3.6, and 3.7 since all the other steps are $O(n)$. Step 3.2 can be implemented in $O(|Q'||S'|/p \log(|Q'||S'|/p))$ time, by using an $O(n \log n)$ time sorting algorithm. Step 3.3 runs in $O(n \log(|Q'||S'|/p)))$ time. Furthermore, step 3.6 runs in $O(|Q'|p \log(|Q'|p))$ time, and step 3.7 runs in $O(n \log(|Q'|p))$ time.

If we set $p = \lfloor \sqrt{S'} \rfloor$, the complexity becomes

$$O(\max\{|\sqrt{S'}||Q'| \log(|\sqrt{S'}||Q'|), n \log(|\sqrt{S'}||Q'|)\}).$$

However, we can show that $|Q'| = O(\log n)$. Assuming that this is true, the complexity becomes $O(n \log(|\sqrt{S'}||Q'|))$, since steps 3.2 and 3.6 become $O(\sqrt{n} \log^2 n) = o(n)$. In addition, the space complexity is $O(n)$. The critical values require $O(|Q'|\sqrt{S'}) = O(\sqrt{n} \log n)$ space, and the other information requires $O(n)$ space. Hence, we can state the following theorem.

THEOREM 5.2. *Let $k(i)$ and $q(i)$ be, respectively, the number of leaves in S' and the number of nodes in Q at the beginning of the ith cycle of the parametric WARM-UP. Then, at the ith cycle, the merge operation runs in $O(n \log k(i) + n \log q(i))$ time, requiring $O(n)$ space.*

In order to complete the analysis, we show that $|Q'| = O(\log n)$.

PROPOSITION 5.3. *At every cycle of the parametric WARM-UP we have that* $|Q'| \leq 2\lceil \log n \rceil + 2$.

Proof. Let $Q(j) = [q_1(j), \ldots, q_{|Q|}(j)]$ and $Q'(j) = [q'_1(j), \ldots, q'_{|Q'|}(j)]$ be, respectively, the lists Q and Q' at the beginning of the jth cycle. Furthermore, let $n_i(j)$ be the number of nodes in the list $Q(j)$ with weight equal to $w(q'_i(j))$ at the beginning of the jth cycle. The list $N(j)$ is given by $N = [n_1(j), \ldots, n_{|Q'|}(j)]$. We use $\max(N(j))$ to denote the maximum value of $N(j)$. For example, if $w(Q(3)) = [2x^*, 33, 2x^*+7, 2x^*+7, 2x^*+7, x^*+30, 46, 64]$, then $w(Q'(3)) = [2x^*, 2x^*+7, x^*+30]$, $N(3) = [1, 3, 1]$, and $\max(N(3)) = 3$. For the sake of simplicity, during this proof, we use the term x^*-node to denote a node whose weight depends on x^*.

Since $|Q'(j)| = |N(j)|$, our approach is to bound $|N(j)|$. First, we consider the list $N(2)$ since $N(1)$ is empty. Let $k-1$ be the number of leaves with weight x^* at the beginning of the second phase of the parametric WARM-UP. At the beginning of the second cycle the list Q has $\lfloor (k-1)/2 \rfloor$ nodes with weights equal to $2x^*$, since the leaves with weight x^* are melded at the first cycle. If $k-1$ is odd, Q may also contain a node with weight equal to x^* or $x^* + w_k$. Hence, at the beginning of the second cycle, we have that either $N(2) = [\lfloor (k-1)/2 \rfloor, 1]$, or $N(2) = [1, \lfloor (k-1)/2 \rfloor]$, or $N(2) = [(k-1)/2]$. In order to prove that $|Q'| \leq 2\lceil \log n \rceil + 2$, it suffices to prove the three facts below.

(a) $\max(N(j+1)) \leq \lceil \max(N(j))/2 \rceil$.

(b) If $\max(N(j)) = 1$, then $|N(j+1)| \leq |N(j)|$.

(c) $|N(j+1)| \leq |N(j)| + 2$.

In effect, since $\max(N(2)) \leq \lceil (k-1)/2 \rceil$, it follows by (a) that after $\lceil \log(k-1)/2 \rceil$ cycles we have $\max(N(j)) = 1$. Hence, it follows by (b) and (c) that $|N(2)|$ can increase by at most $2\lceil \log(k-1)/2 \rceil$ units. Hence, $|Q'(j)| = |N(j)| \leq 2 + 2\lceil \log(k-1)/2 \rceil \leq 2 + 2\lceil \log(n) \rceil$ for all j.

Before proving that (a), (b), and (c) are correct, we should examine a special case. Let us assume that the list U obtained through the merge between $Q(j)$ and S' has an odd number of nodes. In addition, we assume that a x^*-node (say y) occupies the last position of U. In this case, step 4 of the parametric WARM-UP includes y at the beginning of the list $Q(j+1)$, and, as a result, y is not melded at the jth cycle. We use the term $Q(j)$-special to denote a node of $Q(j)$ with this unusual behavior. Furthermore, we define the parent of a $Q(j)$-special node as the node $q_1(j+1)$. This definition is useful because it forces every node in $Q(j)$ to have a representative in $Q(j+1)$. It can be proved by induction that the parent of a $Q(j)$-special node has a weight different from the weight of every other x^*-node in $Q(j+1)$.

Now, we prove that (a), (b), and (c) are correct. First, we prove (a). If $\max(N(j+1)) = 1$, then the result is obvious since $\max(N(j)) \geq 1$. Let $t > 1$. We assume that t x^*-nodes of $Q(j+1)$ have weights equal to $ax^* + b$. We observe that none of these nodes is a $Q(j)$-special parent, and, as a consequence, each of them has two children that belong to $Q(j)$. Moreover, all their $2t$ children must have equal weights. This is because the list U is sorted. Since at least one of the children of a x^*-node must be also a x^*-node, then those $2t$ children are x^*-nodes with weights equal to $a/2x^* + b/2$. Letting $t = \max(N(j+1))$, we establish the correctness of (a).

Now, we prove (b). The condition $\max(N(j)) = 1$ implies that all x^*-nodes of $Q(j)$ have different weights. Hence, $|N(j)|$ is given by the number of x^*-nodes in $Q(j)$. Since the number of x^*-nodes in $Q(j+1)$ is not greater than that of $Q(j)$, we obtain that $|N(j+1)| \leq |N(j)|$.

Now, we prove (c). Let $q_i(j)$ be an x^*-node, and let us assume that there are t nodes in $Q(j)$ with weights equal to that of $q_i(j)$. Under these conditions, we say that $q_i(j)$ contributes to $|N(j)|$ with $1/t$ units. The value of $|N(j)|$ is given by the sum of the contributions of the nodes in $Q(j)$. Furthermore, the value of $|N(j+1)|$ is given by the sum of the contributions of the parents of the x^*-nodes that belong to $Q(j)$. Hence, in order to give an upper bound on $|N(j+1)| - |N(j)|$, we must consider the nodes of $Q(j)$ that contribute with less than one unit. Let us assume that $Q(j)$ has t nodes, $q_{i_1}(j), \ldots, q_{i_t}(j)$, with weights equal to $ax^* + b$. All these nodes, together, contribute to $|N(j)|$ with one unit. In addition, these nodes are disposed at consecutive positions at the list U, and, as a consequence, they generate either $\lfloor t/2 \rfloor$ or $\lfloor t/2 \rfloor - 1$ parents with weights equal to $2(ax^* + b)$. If $q_{i_1}(j)$ is melded with a node whose weight does not depend on x^*, then its parent will have a weight different from the weight of every other x^*-node in $Q(j+1)$. In this case, its parent contributes to $|N(j)|$ with one unit. This may also occur either if $q_{i_t}(j)$ is melded with a node whose weight does not depend on x^* or if $q_{i_t}(j)$ is $Q(j)$-special. Hence $|N(j+1) - N(j)|$ can increase by at most two units from one cycle to another, which establishes the correctness of (a). We observe that the correctness of (a) is based on the fact that at most one element of $N(j)$ is greater than 1. This result can be proved by induction on the number of cycles, using the same kind of arguments presented throughout this proof. ∎

5.4. Algorithm analysis. We can state the following theorem concerning the behavior of the parametric WARM-UP.

THEOREM 5.4. *The parametric WARM-UP spends $O(nL\log(n/L))$ time to construct a tree T_{x^*} with height L, where x^* is the smallest value of x such that $h(T_x^-) \leq L$.*

Proof. The correctness of the algorithm follows from the properties of **Compare**. The decisions taken by **Compare** are equal to the ones taken by Huffman's algorithm when constructing a maximum height Huffman tree for the list of weights $[w_1^*, \ldots, w_n^*]$, where $w_i^* = \max\{x^*, w_i\}$.

Now, we consider its time complexity. The most expensive steps are steps 2 and 3. The exponential search takes $O(n \log k(i))$ time, where $k(i)$ is the number of leaves with weights smaller than α at the ith cycle. On the other hand, it follows from Theorem 5.2 that the merge operation takes $O(n \log k(i) + n \log q(i))$ time, where $q(i)$ is the number of nodes in Q at the ith cycle. Since the algorithm executes at most L cycles, the time complexity is upper-bounded by

$$\sum_{i=1}^{L} O(n \log k(i) + n \log q(i)). \tag{5.1}$$

Since $\sum_{i=1}^{L} k(i) = n$ and $\sum_{i=1}^{L} q(i) < 2n$, then an upper bound for the time complexity can be obtained by maximizing (5.1) as a function of $k(i)$ and $q(i)$, constrained to $\sum_{i=1}^{L} k(i) = n$ and $\sum_{i=1}^{L} q(i) < 2n$.

By Jensen's inequality, the maximum occurs for $k(i) = n/L$ and $k(i) = 2n/L$. Hence, the parametric WARM-UP runs in $O(nL\log(n/L))$ time. ∎

6. Approximation. In the previous sections we proposed a method to construct a prefix code tree T_x with height L. This tree is not necessarily an optimal prefix code tree with restricted maximal height L. Now, we give an upper bound for the

error of the code induced by this tree. Let us define this error ϵ by

$$\epsilon = \sum_{i=1}^{n} p_i l_i - \sum_{i=1}^{n} p_i \bar{l}_i,$$

where $\bar{l}_i$ is the height of ith leaf of an unrestricted Huffman tree, l_i is the height of ith leaf of the tree T_x constructed by the WARM-UP algorithm, and p_i is the probability of symbol a_i that is given by $w_i/(\sum_{j=1}^{n} w_j)$. The value $\sum_{i=1}^{n} p_i l_i$ is the *average code length*, and it represents the average number of bits used per symbol. Now, we state a theorem that gives an upper bound on ϵ.

THEOREM 6.1. *Let L be an integer such that $L > \lceil \log n \rceil + 1$. The average length difference ϵ between the code induced by a tree T_x with height L and an unrestricted Huffman code is such that*

$$\epsilon < 1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 2},$$

where ψ is the golden ratio 1.618.

Proof. Let A_W, A_H, and A_L be, respectively, the average length of a code induced by a tree T_x with height L, the average length of a Huffman code, and the average length of an optimal prefix code with restricted maximal length L.

Theorem 2.3 shows that the heights of the leaves of a tree T_x with height L are optimal for problem P_L augmented by some constraints of type $l_i \leq L - 1$. Since this last problem is a relaxation of problem P_{L-1}, it follows that $A_W \leq A_{L-1}$. Nevertheless, Milidiú and Laber [15] show that if $L \geq \lceil \log n \rceil + 1$, then $A_L - A_H \leq 1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 1}$. Hence, $\epsilon = A_W - A_H \leq A_{L-1} - A_H < 1/\psi^{L - \lceil \log(n + \lceil \log n \rceil - L) \rceil - 2}$. □

This result shows that the average code length difference is quite small for values of L not too close to $\lceil \log n \rceil$. For example, if $L - \lceil \log n \rceil \geq 15$, then the difference ϵ is less than 0.01. For $L - \lceil \log n \rceil \geq 20$, we have that $\epsilon < 0.0003$. Nevertheless, undesirable results like the difference being greater than 1 could occur. However, that would only happen if $L \leq \lceil \log n \rceil + 1$.

6.1. Limitations. Theorem 2.1 shows that if all the leaves with weight x in a T_x tree have height L, then T_x is an optimal code tree with restricted maximal height L. Unfortunately, for some list of weights there is no such x. For instance, consider $L = 3$ and the list of weights given by $[1, 1, 1, 13, 15]$. Figure 5 (a) shows the tree topology for all T_x with $1 \leq x \leq 5$, and Figure 5 (b) shows the tree topology for all T_x with $5 \leq x \leq 15$. Observe that, in the tree topology of picture (b), not all the leaves with weight less than 5 have height 3. Moreover, the average code length for this tree is 2.064. On the other hand, this measure is equal to 2.032 for the optimal code tree exhibited in Figure 5 (c).

One could ask what happens if we choose the multipliers in a different way. For the list of weights presented in Figure 5, it has been proved [11] that the optimal code tree can not be obtained, even if we try all the sets of multipliers.

7. Experimental results. The source data for our experiment was extracted from the *Calgary corpus*. This corpus is a collection of 14 files available from the University of Calgary, Canada. In order to generate the input vector of weights for each source data, we considered two different models to parse the data. The first method defines each single byte as a symbol of the alphabet. The second method packs

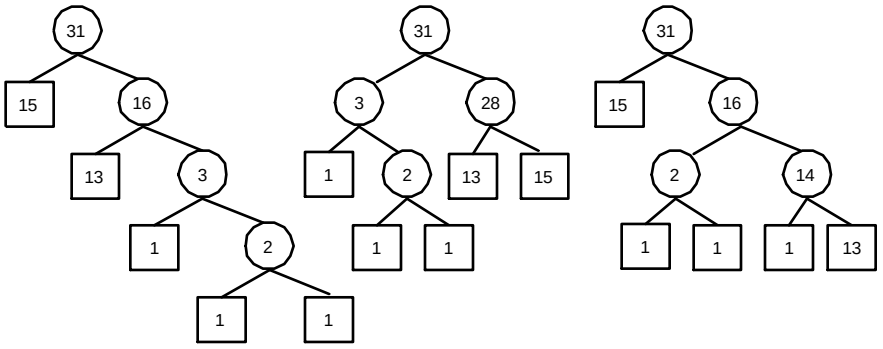


FIG. 5. (a) *Tree T_x for the list of weights $W = [1, 1, 2, 3, 5, 8, 13]$ and $1 \leq x \leq 5$.* (b) *Tree T_x for the list of weights W and $5 \leq x \leq 15$.* (c) *An optimal tree with restricted maximal height 3 for the list W.*

any sequence of alphanumeric American standard code for information interchange (ASCII) codes[3] into a single symbol. Each nonalphanumeric ASCII code is also considered a symbol. This second method is suitable for English text files and source code files. In order to generate each list of weights, the parsing method scans each source data counting the frequency of each distinct symbol. The first method is referred as the *Byte* method and the second as the *Word* method.

Table 1 shows the average codeword length, measured in bits, obtained by each algorithm for the Calgary corpus files. For each file, we considered L running from $\lceil \log n \rceil$ to $\lceil \log n \rceil + 6$. A letter (b) beside the file name indicates that the file was parsed through a Byte model, whereas the letter (w) indicates the word model. The second column in the table indicates the number n of different symbols. From this table, we get the following observations.

1. The average length of the codes produced by the WARM-UP algorithm was equal to that produced by the LazyPM algorithm, except for two cases. These two cases are printed in bold in Table 1.
2. In a few cases, the ROT algorithm generated very inefficient codes. In the worst case, the compression loss was 63.5% for the *pic* file and $L = \lceil \log n \rceil$. Optimal codes were generated on seven experiments.

In this section, we report some experiments involving three methods that generate length restricted prefix codes. The first method is the ROT Heuristics proposed in [5]. The second method is the WARM-UP, with the implementation presented in Figure 4, except that we removed Phase 2. The third one is the Package-Merge [12], an exact method to construct optimal length restricted prefix codes. In these experiments, we measured the average length of the codes obtained through these algorithms.

8. Conclusions. In this paper, we introduced the WARM-UP algorithm, a novel technique to construct length-restricted prefix codes. This algorithm is very easy to implement, since it consists of successive constructions of Huffman trees. Even though it is approximative, it has an optimal behavior for practical settings.

We believe that the techniques developed here can be useful to address the problem of constructing optimal prefix codes satisfying other kinds of constraints.

[3]The *underscore* character is treated as alphanumeric.

TABLE 1

The average codeword length in bits obtained by each algorithm for Calgary corpus files for L from $\lceil \log n \rceil$ to $\lceil \log n \rceil + 6$.

File	n	Method	$L - \lceil \log n \rceil$						
			0	1	2	3	4	5	6
bib (b)	81	ROT	5.5429	5.3198	5.2628	5.2420	5.2349	5.2328	5.2320
		WARM-UP	5.5305	5.3176	5.2606	5.2419	5.2349	5.2328	5.2320
		PM	5.5305	5.3176	5.2606	5.2419	5.2349	5.2328	5.2320
book1 (w)	3186	ROT	9.4033	7.5160	6.5794	6.3838			
		WARM-UP	7.8299	6.6587	6.4491	6.3762			
		PM	7.8299	6.6587	6.4491	6.3762			
book2 (w)	7944	ROT	10.2614	7.4370	6.7921	6.6226	6.5920		
		WARM-UP	9.6401	7.0257	6.6917	6.6119	6.5910		
		PM	9.6401	7.0257	6.6917	6.6119	6.5910		
geo (b)	256	ROT	8.0000	5.8181	5.6801	5.6693			
		WARM-UP	8.0000	5.8073	5.6800	5.6693			
		PM	8.0000	5.8073	5.6800	5.6693			
news (w)	9975	ROT	9.3045	7.2137	6.8939				
		WARM-UP	7.5054	6.9916	6.8835				
		PM	7.5054	6.9916	6.8835				
obj1 (b)	256	ROT	8.0000	6.1892	6.0040	5.9749	5.9715		
		WARM-UP	8.0000	6.1295	5.9983	5.9744	5.9715		
		PM	8.0000	6.1295	5.9983	5.9744	5.9715		
obj2 (b)	256	ROT	8.0000	6.5182	6.3432	6.3053	6.2947	6.2918	6.2913
		WARM-UP	8.0000	6.4710	6.3386	6.3051	6.2947	6.2918	6.2913
		PM	8.0000	6.4710	6.3386	6.3051	6.2947	6.2918	6.2913
paper (w)	11834	ROT	8.3159	6.5581	6.2835				
		WARM-UP	7.8441	6.4905	6.2786				
		PM	7.8441	6.4905	6.2786				
paper (w)	42499	ROT	7.1453	6.1232	5.7413				
		WARM-UP	6.3766	5.8064	**5.8052**				
		PM	6.3766	5.8064	**5.7399**				
pic (b)	159	ROT	4.2635	1.7553	1.6953	1.6836	1.6672	1.6628	1.6614
		WARM-UP	**2.6361**	1.7511	1.6915	1.6727	1.6655	1.6625	1.6614
		PM	**2.6072**	1.7511	1.6915	1.6727	1.6655	1.6625	1.6614
progc (w)	1148	ROT	7.0186	5.6552	5.4895				
		WARM-UP	6.0109	5.5740	5.4816				
		PM	6.0109	5.5740	5.4816				
progl (w)	979	ROT	7.9505	5.6576	5.1389	4.9679	4.9340		
		WARM-UP	7.7133	5.4060	5.0484	4.9607	4.9336		
		PM	7.7133	5.4060	5.0484	4.9607	4.9336		
progp (w)	595	ROT	6.3733	5.3580	4.8991	4.8067	4.7837		
		WARM-UP	5.3977	4.9586	4.8525	4.7964	4.7834		
		PM	5.3977	4.9586	4.8525	4.7964	4.7834		
trans (b)	99	ROT	6.0556	5.7082	5.6207	5.5843	5.5735	5.5700	5.5689
		WARM-UP	6.0526	5.7065	5.6123	5.5836	5.5734	5.5699	5.5689
		PM	6.0526	5.7065	5.6123	5.5836	5.5734	5.5699	5.5689

REFERENCES

[1] A. AGGARWAL, B. SCHIEBER, AND T. TOKUYAMA, *Finding a minimum-weight k-link path in graphs with the concave Monge property and applications*, Discrete Comput. Geom., 12 (1994), pp. 263–280.

[2] R. K. AHUJA, T. L. MAGNANTI, AND J. B. ORLIN, *Network Flows. Theory, Algorithms, and Applications,* Prentice-Hall, Englewood Cliffs, NJ, 1993.

[3] Y. CHOUEKA, S. T. KLEIN, AND Y. PERL, *Efficient variants of Huffman codes in high level languages*, in Proceedings of the 8th ACM SIGIR Conference on Research and Development in Information Retrieval, Montreal, Canada, 1985, pp. 122–130.

[4] S. Even, *Graph Algorithms*, Computer Science Press, Woodland Hills, CA, 1979.
[5] A. S. Fraenkel and S. T. Klein, *Bounding the depth of search trees*, The Computer Journal, 36 (1993), pp. 668–678.
[6] R. Gallager, *Variations on a theme of Huffman*, IEEE Trans. Inform. Theory, 24 (1978), pp. 668–674.
[7] M. R. Garey, *Optimal binary search trees with restricted maximal depth*, SIAM J. Comput., 3 (1974), pp. 101–110.
[8] E. N. Gilbert, *Codes based on inaccurate source probabilities*, IEEE Trans. Inform. Theory, 17 (1971), pp. 304–314.
[9] T. C. Hu and K. C. Tan, *Path length of binary search trees*, SIAM J. Appl. Math., 22 (1972), pp. 225–234.
[10] D. A. Huffman, *A method for the construction of minimum-redundancy codes.*, in Proc. Inst. Radio Eng., 40 (1952), pp. 1098–1101.
[11] E. S. Laber, *Um algoritmo eficiente para construção de códigos de prefixo com restrição de comprimento*, Master's thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil, 1997.
[12] L. L. Larmore and D. S. Hirschberg, *A fast algorithm for optimal length-limited Huffman codes*, J. ACM, 37 (1990), pp. 464–473.
[13] L. L. Larmore and T. M. Przytycka, *Constructing Huffman trees in parallel*, SIAM J. Comput., 24 (1995), pp. 1163–1169.
[14] N. Megiddo, *Applying parallel computation algorithms in the design of serial algorithms*, J. ACM, 30 (1983), pp. 852–865.
[15] R. L. Milidiú and E. S. Laber, *Improved bounds on the inefficiency of length restricted codes*, Algorithmica, to appear.
[16] R. L. Milidiú, E. S. Laber, and A. A. Pessoa, *A work efficient parallel algorithm for constructing Huffman codes*, in Proceedings of the IEEE Data Compression Conference, J. A. Storer and M. Cohn, eds., Snowbird, UT, 1999, pp. 277–286.
[17] R. L. Milidiú, A. A. Pessoa, and E. S. Laber, *In-place length-restricted prefix coding*, in Proceedings of the South American Symposium on String Processing and Information Retrieval, B. A. R. Neto, ed., Santa Cruz de La Sierra, Bolivia, 1998, pp. 50–59.
[18] R. L. Milidiú, A. A. Pessoa, and E. S. Laber, *Efficient implementation of the WARM-UP algorithm for the construction of length-restricted prefix codes*, in Proceedings of the Workshop on Algorithm Engineering and Experimentation, Baltimore, MD, 1999, Lecture Notes in Comput. Sci. 1619, Springer, New York, 1999, pp. 1–17.
[19] R. L. Milidiú, A. A. Pessoa, and E. S. Laber, *Two space-economical algorithms for calculating minimum redundancy prefix codes*, in Proceedings of the IEEE Data Compression Conference, J. A. Storer and M. Cohn, eds., Snowbird, UT, 1999, pp. 267–276.
[20] M. Minoux, *Mathematical Programming, Theory and Algorithms*, John Wiley and Sons, New York, 1986.
[21] A. Moffat and J. Katajainen, *In-place calculation of minimum-redundancy codes*, in Algorithms and Data Structures, 4th International Workshop, Kingston, Ontario, Canada, 1995, Lecture Notes in Comput. Sci. 955, S. G. Akl, F. K. H. A. Dehne, J.-R. Sack, and N. Santoro, eds., Springer, New York, 1995, pp. 393–402.
[22] A. Moffat and A. Turpin, *Efficient construction of minimum-redundancy codes for large alphabets*, IEEE Trans. Inform. Theory, 44 (1998), pp. 1650–1657.
[23] B. Schieber, *Computing a minimum-weight k-link path in graphs with the concave Monge property*, J. Algorithms, 29 (1998), pp. 204–222.
[24] E. S. Schwartz, *An optimum encoding with minimum longest code and total number of digits*, Information and Control, 7 (1964), pp. 37–44.
[25] A. Turpin and A. Moffat, *Practical length-limited coding for large alphabets*, The Computer Journal, 38 (1995), pp. 339–347.
[26] J. van Leeuwen, *On the construction of Huffman trees*, in Proceedings of the Third International Colloquium on Automata, Languages and Programming, University of Edinburgh, Edinburgh, Scotland, S. Michaelson and R. Milner, eds., Edinburgh University Press, Edinburgh, Scotland, 1976, pp. 382–410.
[27] D. C. van Voorhis, *Constructing codes with bounded codeword lengths*, IEEE Trans. Inform. Theory, 20 (1974), pp. 288–290.
[28] J. S. Vitter, *Design and analysis of dynamic Huffman codes*, J. ACM, 34 (1987), pp. 825–845.
[29] J. Zobel and A. Moffat, *Adding compression to a full-text retrieval system*, Software—Practice and Experience, 25 (1995), pp. 891–903.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1427–1442

A NEAR-TIGHT LOWER BOUND ON THE TIME COMPLEXITY OF DISTRIBUTED MINIMUM-WEIGHT SPANNING TREE CONSTRUCTION*

DAVID PELEG† AND VITALY RUBINOVICH‡

Abstract. This paper presents a lower bound of $\Omega(D+\sqrt{n}/\log n)$ on the time required for the distributed construction of a minimum-weight spanning tree (MST) in weighted n-vertex networks of diameter $D=\Omega(\log n)$, in the bounded message model. This establishes the asymptotic near-optimality of existing time-efficient distributed algorithms for the problem, whose complexity is $O(D+\sqrt{n}\log^* n)$.

Key words. distributed algorithm, minimum weight spanning tree, lower bound

AMS subject classifications. 05C85, 68Q22, 68Q25

PII. S0097539700369740

1. Introduction.

The study of distributed algorithms for minimum-weight spanning tree (MST) construction was initiated by the pioneering work of Gallager, Humblet, and Spira [GHS83], which introduced a basic distributed technique for the problem and presented a message-optimal algorithm with time complexity $O(n\log n)$ on an n-vertex network. This result was later improved to a message-optimal algorithm with time complexity $O(n)$ by Awerbuch [A87].

However, for many natural distributed network problems, the parameter controlling the time complexity is not the number of vertices but rather the network's diameter D, namely, the maximum distance between any two vertices (measured in hops). This holds, for example, for leader election and related problems [P90].

It is easy to verify that $\Omega(D)$ time is required for distributed MST construction in the worst case. More formally, for every two integers $n\geq 2$ and $1\leq D\leq \lfloor n/2\rfloor$ there exist weighted n-vertex networks of diameter D (say, based on a $2D$-vertex ring with $n-2D$ vertices attached to it as leaves) on which any distributed MST algorithm will require at least D time.

Hence, a natural question is whether $O(D)$-time algorithms exist for distributed MST construction as well. More generally, the problem of devising $o(n)$ (though possibly not message-optimal) distributed algorithms for MST construction was introduced in [GKP98].

Clearly, in the extreme model allowing the transmission of an unbounded-size message on a link in a single time unit (cf. [L92]), the problem can be trivially solved in time $O(D)$ by collecting the entire graph's topology and all the edge weights into a central vertex, computing an MST locally and broadcasting the result throughout the network. The problem thus becomes interesting in the more realistic, and rather

*Received by the editors March 27, 2000; accepted for publication (in revised form) August 1, 2000; published electronically November 8, 2000. An extended abstract has appeared in *A near-tight lower bound on the time complexity of distributed MST construction*, in Proceedings of the 40th IEEE Symposium on Foundations of Comoputer Science, IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 253–261.

http://www.siam.org/journals/sicomp/30-5/36974.html

†Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel (peleg@wisdom.weizmann.ac.il). The work of this author was supported in part by a grant from the Israel Ministry of Science and Art.

‡Department of Mathematics and Computer Science, Bar Ilan University, Ramat Gan, Israel (rabinov@macs.biu.ac.il).

common, *B-bounded-message* model (henceforth referred to simply as the B model), in which message size is bounded by some value B (usually taken to be either constant or $O(\log n)$), and a vertex may send at most one message on each edge at each time unit.

The algorithm presented in [GKP98] for distributed MST construction in this model (with $B = O(\log n)$-bit messages) has time complexity $O(D + n^{\epsilon} \log^* n)$ for $\epsilon = \ln 6 / \ln 3 \approx 0.613$. This was later improved to $O(D+\sqrt{n}\log^* n)$ in [KP98]. Similar bounds were recently obtained by us using other algorithmic methods, but none of those methods were able to break the $\sqrt{n}$-time barrier, indicating that distributed MST might be harder than other distributed network problems such as leader election or breadth-first search (BFS) tree construction.

It is important to mention that the algorithms of [GHS83, A87, GKP98, KP98] discussed above were analyzed under the (natural) assumption that the weight of each edge can be represented as an integer small enough to be included in a single message. This assumption is adopted in the current paper.

The current paper concerns establishing the asymptotic near-optimality of the algorithm of [KP98], by showing that $\tilde{\Omega}(\sqrt{n})$ is a lower bound[1] as well, even on low diameter networks. Specifically, for any integers $K, m \geq 2$, we construct a family of $O(m^{2K})$-vertex networks of diameter $D = O(Km)$ for which $\Omega(m^K/(BK))$ time is required for constructing a minimum spanning tree in the B model. Fixing some positive integer $m \geq 2$, we get that for every integer $n \geq 1$ there exists a family of n-vertex networks of diameter $\Theta(\log n)$ for which MST construction requires $\Omega(\sqrt{n}/(B \log n))$ time in the B model.

While it is not clear that the $\Omega(\log n)$ limitation on the diameters for which the lower bound holds is essential, *some* limitation must apparently exist. This follows from the observation that the n-vertex complete graph ($D = 1$) admits a simple $O(\log n)$ time distributed MST construction algorithm.

Towards proving the lower bound on distributed MST construction, we first establish a lower bound on the time complexity of a problem referred to as the *mailing problem*, which can be informally stated as follows. Given a particular type of graph named F_m^K, for integers $m, K \geq 2$, and two vertices s and r in it, it is required to deliver an m^K-bit string $\mathcal{X}$ generated in s to r. The graph F_m^K has $n = O(m^{2K})$ vertices and diameter $O(Km)$, yet we show that the time required for mailing from s to r on F_m^K in the B model is considerably larger than the diameter, namely, $\Omega(m^K/(BK)) = \Omega(\sqrt{n}/(BK))$.

The rest of the paper is organized as follows. First, a definition of the model and the mailing problem is given in section 2. Section 3 handles the mailing problem for the case of $K = 2$. It defines the graphs F_m^2, having diameter $D = O(m) = O(n^{1/4})$ and shows a lower bound of $O(m^2) = O(\sqrt{n})$ on the time complexity of the mailing problem for m^2-bit strings. This result is then used in section 4 to prove that the same lower bound applies also to the time complexity of the MST problem on weighted versions of the graphs F_m^2. The next two sections extend these two results, respectively, to graphs F_m^K, $K \geq 3$ with diameters down to $O(\log n)$. Finally, section 7 discusses some open problems.

2. Preliminaries.

2.1. The model. A point-to-point communication network is modeled as an undirected graph $G(V, E)$, where the vertices of V represent the network processors

[1] $\tilde{\Omega}$ is a relaxed variant of the Ω notation that ignores polylog factors.

and the edges of E represent the communication links connecting them. Vertices are allowed to have unique identifiers. The vertices do not know the topology or the edge weights of the entire network, but they may know the IDs of their neighbors and the weights of the corresponding edges.

A weight function $\omega : E \to \mathbb{R}^+$ associated with the graph assigns a nonnegative integer weight $\omega(e)$ to each edge $e = (u, v) \in E$. The weight $\omega(e)$ is known to the adjacent vertices, u and v. The vertices can communicate only by sending and receiving messages over the communication links. Communication is carried out in a synchronous manner; i.e., all the vertices are driven by a global clock. Messages are sent at the beginning of each round and are received at the end of the round. (Clearly, our lower bounds hold for asynchronous networks as well.) At most one B-bit message can be sent on each link in one direction on every round. It is assumed that B is large enough to allow the transmission of an edge weight in a single message. The model also allows vertices to detect the absence of a message on a link at a given round, which can be used to convey information. Hence at each communication round, a link can be at one of $2^B + 1$ possible states, i.e., it can either transmit any of 2^B possible messages or remain silent.

The length of a path p in the network is the number of edges it contains. The *distance* between two vertices u and v is defined as the length of the shortest path connecting them in G. The *diameter* of G, denoted D, is the maximum distance between any two vertices of G.

2.2. The mailing problem. The mailing problem is defined in the following situation. We are given a graph G with two distinguished vertices s and r, referred to as the *sender* and the *receiver*, respectively. Both the sender s and the receiver r store b boolean variables each, $X_1^s, \ldots, X_b^s$ and $X_1^r, \ldots, X_b^r$, respectively, for some integer $b \geq 1$. An instance of the problem consists of an initial assignment $\mathcal{X} = \{x_i \mid 1 \leq i \leq b\}$, where $x_i \in \{0, 1\}$, to the variables of s, such that $X_i^s = x_i$. Given such an instance, the mailing problem requires s to deliver the string $\mathcal{X}$ to the receiver r, i.e., upon termination, the variables of r should contain the output $X_i^r = x_i$ for every $1 \leq i \leq b$. Henceforth, we refer to this problem as $\texttt{Mail}(G, s, r, b)$. Throughout sections 3 and 4, we consider this problem on graphs F_m^2 with $b = m^2$ for some integer $m \geq 2$. In sections 5 and 6, we deal with the problem on graphs F_m^K with $b = m^K$ for $K \geq 3$.

2.3. The distributed MST problem. Formally, the *minimum spanning tree (MST)* problem can be stated as follows. Given a graph $G(V, E)$ and a weight function ω on the edges, it is required to find a spanning tree $MST(G) \subseteq E$ whose total weight, $\omega(MST(G)) = \sum_{e \in MST(G)} \omega(e)$, is minimal. In the distributed model, the input and output of the MST problem are represented as follows. Each vertex knows the ID's of its closest neighbors and the weights of the corresponding edges. A degree-d vertex $v \in V$ with neighbors $u_1, \ldots, u_d$ has d *weight variables* $W_1^v, \ldots, W_d^v$, with W_i^v containing the weight of the edge connecting v to u_i, i.e., $W_i^v = \omega(v, u_i)$. The output of the MST problem at each vertex v is an assignment to the (boolean) output variables $Y_1^v, \ldots, Y_d^v$, assigning

$$Y_i^v = \begin{cases} 1, & (u_i, v) \in MST(G), \\ 0 & \text{otherwise.} \end{cases}$$

3. A lower bound for the mailing problem on F_m^2.

3.1. The graphs F_m^2. Let us now define the collection of graphs denoted F_m^2 for $m \geq 2$. The two basic units in the construction are the *ordinary path* $\mathcal{P}$ on m^2+1 vertices,

$$V(\mathcal{P}) = \{v_0, \ldots, v_{m^2}\} \quad \text{and} \quad E(\mathcal{P}) = \{(v_i, v_{i+1}) \mid 0 \leq i \leq m^2 - 1\},$$

and the *highway* $\mathcal{H}$ on $m+1$ vertices,

$$V(\mathcal{H}) = \{h_{im} \mid 0 \leq i \leq m\} \quad \text{and} \quad E(\mathcal{H}) = \{(h_{im}, h_{(i+1)m}) \mid 0 \leq i \leq m-1\}.$$

Each highway vertex h_{im} is connected to the corresponding path vertex v_{im} by a *spoke* edge (h_{im}, v_{im}), as in Figure 1.

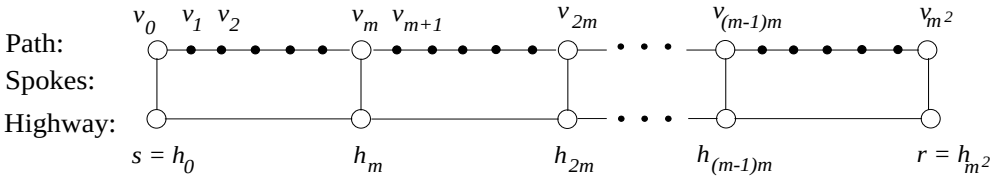


FIG. 1. *The connections between the path and the highway.*

The graph F_m^2 is constructed by taking m^2 copies of the ordinary path $\mathcal{P}$, denoted $\mathcal{P}^1, \ldots, \mathcal{P}^{m^2}$, and connecting all of them to the *same* highway $\mathcal{H}$. The vertex h_0 is the intended sender s, and the vertex h_{m^2} is the intended receiver r. (See Figure 2.)

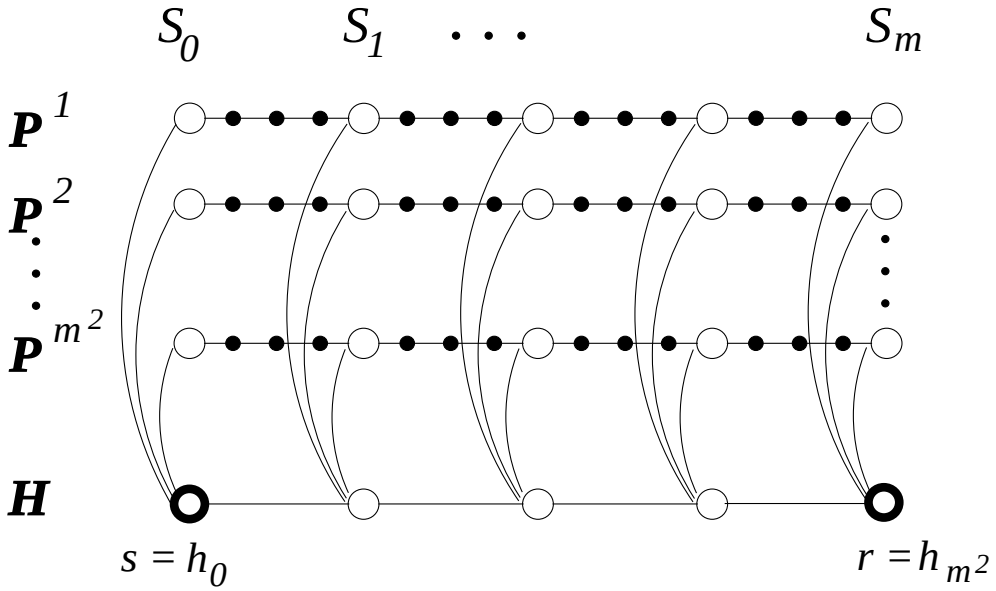


FIG. 2. *The graph* F_m^2.

Visualizing the graph F_m^2 as organized in a cylindrical shape, the spoke edges can be grouped into $m+1$ *stars* S_i, $0 \leq i \leq m$, where each star S_i consists of the highway vertex h_{im} and the m^2 vertices $v_{im}^1, \ldots, v_{im}^{m^2}$ connected to it by spoke edges. Hence

$$V(S_i) = \{h_{im}\} \cup \{v_{im}^1, \ldots, v_{im}^{m^2}\} \quad \text{and} \quad E(S_i) = \{(v_{im}^j, h_{im}) \mid 1 \leq j \leq m^2\}.$$

The vertex and edge sets of the graph F_m^2 are thus

$$V(F_m^2) = V(\mathcal{H}) \cup \bigcup_{j=1}^{m^2} V(\mathcal{P}^j) \quad \text{and} \quad E(F_m^2) = \bigcup_{i=0}^{m} E(S_i) \cup \bigcup_{j=1}^{m^2} E(\mathcal{P}^j) \cup E(\mathcal{H}).$$

FACT 3.1. *The graph F_m^2 consists of $n = \Theta(m^4)$ vertices, and its diameter is $\Theta(m)$.*

3.2. The lower bound. We would now like to prove that solving the mailing problem on the graph F_m^2 with a $b = m^2$-bit string $\mathcal{X}$ requires $\Omega(m^2/B)$ time in the B model. Intuitively, this happens because routing the string $\mathcal{X}$ from s to r along ordinary paths would be too slow; hence our only hope is to route the string along the highway, or at least use interleaved paths, mixing highway segments with segments of ordinary paths. However, F_m^2 does not have sufficient capacity for routing all m^2 bits from s to r along such short (or "relatively short") paths.

This intuition yields a rather simple proof of the claim if we limit ourselves to a restricted class of algorithms, referred to as *explicit delivery* algorithms. These are algorithms in which the input bits are required to be delivered in an explicit way, namely, each bit x_i must be shipped from s to t along some path p_i. (Naturally, the paths of different bits may be identical or partly overlap.) However, we would like the lower bound to apply also to *arbitrary* algorithms, in which the information can be conveyed from s to r in arbitrary ways. This may include applying arbitrary functions to the bits at s and sending the resulting values, possibly modifying and "recombining" these values in intermediate nodes along the way, in a way that will allow r to extract the original bits from the messages it receives. For handling such a general class of algorithms, the proof must be formalized in a more careful way.

Let us start with an outline of the proof. Consider the set of possible states a vertex v may be in at any given stage t of the execution of a mailing algorithm on some m^2-bit input $\mathcal{X}$. (The state of a vertex consists of all its local data; hence it is affected by its input, topological knowledge, and history, namely, all incoming messages.) As the computation progresses, the tree of possible executions grows, and thus the set of possible states of v becomes larger. In particular, when the execution starts at round 0, each of the vertices is in one specific initial local state, except for the sender s, which may be in any one of 2^{m^2} states, determined by the value of the input string $\mathcal{X}$. Upon termination, the string $\mathcal{X}$ should be known to the receiver r, meaning that r should be in one of 2^{m^2} states. Our argument is based on analyzing the growth process of the sets of possible states, and showing that this process is slow, forcing the algorithm to spend at least $\Omega(m^2/B)$ time until the set of possible states of r is of size 2^{m^2}.

We now continue with a more detailed and formal proof. Consider some arbitrary algorithm $\mathcal{A}_{\text{mail}}$, and let $\varphi_{\mathcal{X}}$ denote the execution of $\mathcal{A}_{\text{mail}}$ on an m^2-bit input $\mathcal{X}$ in the graph F_m^2. For $1 \leq i \leq m$, define the *tail set* of the graph F_m^2, denoted T_i, as follows. For every $1 \leq j \leq m^2$, define the *tail* of the path $\mathcal{P}^j$ as

$$T_i(\mathcal{P}^j) = \{v_l^j \mid i \leq l \leq m^2\}.$$

Let $\beta(i)$ denote the least integer δ such that $\delta m \geq i$, and define the tail of $\mathcal{H}$ as

$$T_i(\mathcal{H}) = \{h_{jm} \mid \beta(i) \leq j \leq m\}.$$

Now, the tail set of F_m^2 is the union of those tails,

$$T_i = T_i(\mathcal{H}) \cup \bigcup_j T_i(\mathcal{P}^j).$$

(See Figure 3.) For $i = 0$, the definition is slightly different, letting

$$T_0 = V \setminus \{h_0\}.$$

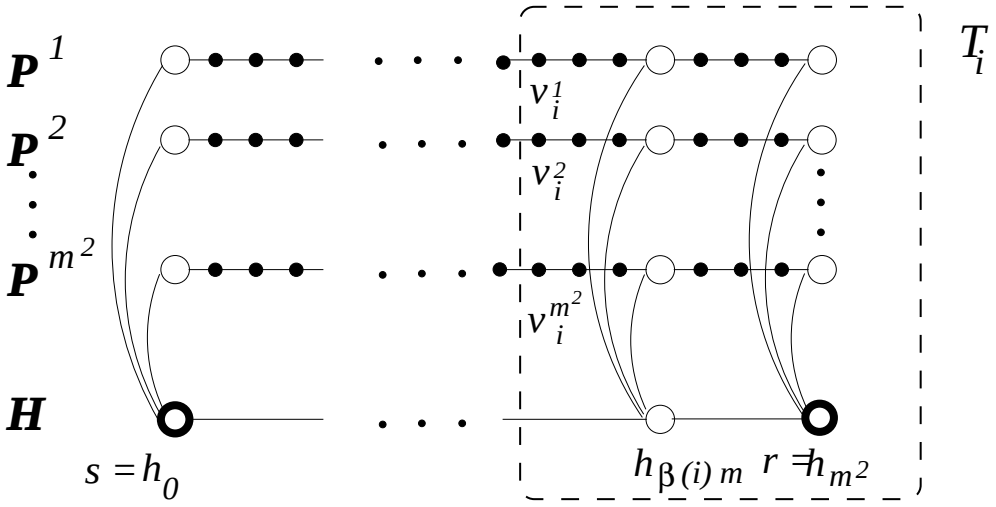


FIG. 3. *The tail set T_i in the graph F_m^2.*

Denote the *state* of the vertex v at the beginning of round t during the execution $\varphi_{\mathcal{X}}$ on the input $\mathcal{X}$ by $\sigma(v, t, \mathcal{X})$. In two different executions $\varphi_{\mathcal{X}}$ and $\varphi_{\mathcal{X}'}$, a vertex reaches the same state at time t, i.e., $\sigma(v, t, \mathcal{X}) = \sigma(v, t, \mathcal{X}')$, iff it receives the same sequence of messages on each of its incoming links; for different sequences, the states are distinguishable.

For a given set of vertices $U = \{v_1, \ldots, v_l\} \subseteq V$, a *configuration*

$$C(U, t, \mathcal{X}) = \langle \sigma(v_1, t, \mathcal{X}), \ldots, \sigma(v_l, t, \mathcal{X}) \rangle$$

is a vector of the states of the vertices of U at the beginning of round t of the execution $\varphi_{\mathcal{X}}$. Denote by $\mathcal{C}[U, t]$ the collection of all possible configurations of the subset $U \subseteq V$ at time t over all executions $\varphi_{\mathcal{X}}$ of algorithm $\mathcal{A}_{\text{mail}}$ (i.e., on all legal inputs $\mathcal{X}$), and let $\rho[U, t] = |\mathcal{C}[U, t]|$.

Prior to the beginning of the execution (i.e., at the beginning of round $t = 0$), the input string $\mathcal{X}$ is known only to the sender s. The rest of the vertices are found in some initial state, described by the configuration $C_{init} = C(T_0, 0, \mathcal{X})$, which is independent of $\mathcal{X}$. Thus, in particular, $\rho[T_0, 0] = 1$. (Note, however, that $\rho[V, 0] = 2^{m^2}$.)

Our main lemma is the following.

LEMMA 3.2. *For every $0 \leq t < m^2$,*

$$\rho[T_{t+1}, t+1] \leq (2^B + 1) \cdot \rho[T_t, t].$$

Proof. The lemma is proved by showing that in round $t+1$ of the algorithm, each configuration in $\mathcal{C}[T_t, t]$ branches off into at most $2^B + 1$ different configurations of $\mathcal{C}[T_{t+1}, t+1]$.

Fix a configuration $\hat{C} \in \mathcal{C}[T_t, t]$, and let $\delta = \beta(t+1)$. The tail set T_{t+1} is connected to the rest of the graph by the highway edge $f_{\delta-1} = (h_{(\delta-1)m}, h_{\delta m})$ and by the m^2 path edges $e_t^j = (v_t^j, v_{t+1}^j)$, $1 \le j \le m^2$. (See Figure 4.)

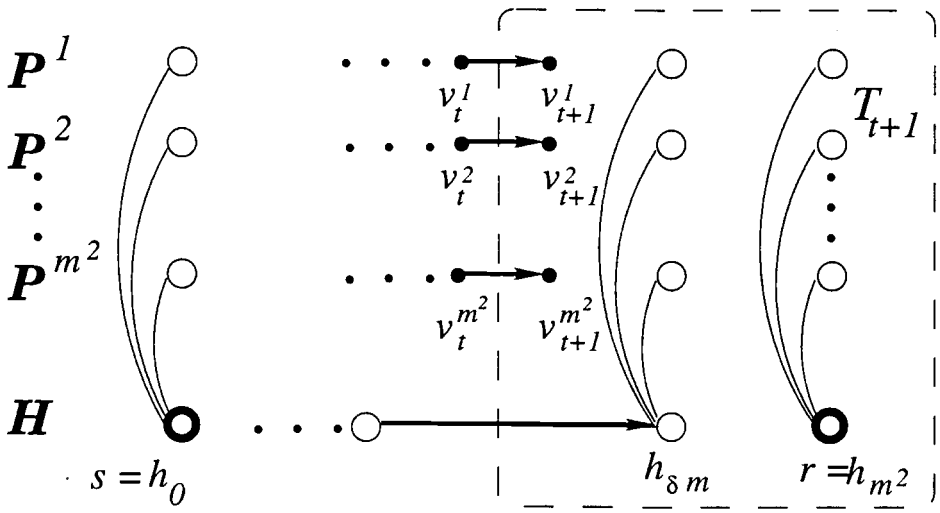


FIG. 4. *The edges entering the tail set* T_{t+1}.

Let us count the number of different configurations in $\mathcal{C}[T_{t+1}, t+1]$ that may result from the configuration $\hat{C}$. Starting from the configuration $\hat{C}$, each vertex v_t^j is restricted to a single state, and hence it sends a single (well determined) message over the edge e_t^j to v_{t+1}^j, thus not introducing any divergence in the execution. The same applies to all the edges internal to T_{t+1}. As for the highway edge $f_{\delta-1}$, the vertex $h_{(\delta-1)m}$ is not in the set T_t; hence it may be in any one of many possible states, and the value passed over this edge into the set T_{t+1} is not determined by the configuration $\hat{C}$. However, due to the restriction of the B-bounded-message model, at most $2^B + 1$ different behaviors of $f_{\delta-1}$ can be observed by $h_{\delta m}$. Thus altogether, the configuration $\hat{C}$ branches off into at most $2^B + 1$ possible configurations $\hat{C}_1, \ldots, \hat{C}_{2^B+1} \in \mathcal{C}[T_{t+1}, t+1]$, differing only by the state $\sigma(h_{\delta m}, t+1, \mathcal{X})$. The lemma follows. □

Applying Lemma 3.2 and the fact that $\rho[T_0, 0] = 1$, we get the following result.

COROLLARY 3.3. *For every* $0 \le t < m^2$,

$$\rho[T_t, t] \le (2^B + 1)^t.$$

Let t_{end} denote the time it takes algorithm $\mathcal{A}_{\text{mail}}$ to complete the mailing. As argued earlier, at that time, the receiver r may be in at least 2^{m^2} different states, hence necessarily $\rho[T_{t_{end}}, t_{end}] \ge 2^{m^2}$. Applying Corollary 3.3, we get that $(2^B + 1)^{t_{end}} \ge 2^{m^2}$, implying the following.

LEMMA 3.4. *For every* $m \ge 1$, *solving the mailing problem* $\texttt{Mail}(F_m^2, h_0, h_{m^2}, m^2)$ *in the* B *model requires* $\Omega(m^2/B)$ *time.*

4. A lower bound for the MST problem on $\mathcal{J}_m^2$. In this section, we use the results achieved in the previous sections and show that in the B model for $B \geq 3$, the distributed MST problem cannot be solved faster than $\Omega(m^2/B)$ on weighted versions of the graphs F_m^2. In order to prove this lower bound, we define in subsection 4.1 a family of weighted graphs $\mathcal{J}_m^2$, based on F_m^2 but differing in their weight assignments. Then in subsection 4.2, we show that any algorithm solving the MST problem on the graphs of $\mathcal{J}_m^2$ can also be used to solve the mailing problem on F_m^2 with the same time complexity. Subsequently, the lower bound for the distributed MST problem follows from the lower bound given in the previous section for the mailing problem in F_m^2.

4.1. The graph family $\mathcal{J}_m^2$. The graphs F_m^2 defined earlier were unweighted. In this subsection, we define for every graph F_m^2 a family of weighted graphs

$$\mathcal{J}_m^2 = \{J_{m,\gamma}^2 = (F_m^2, \omega_\gamma) \mid 1 \leq \gamma \leq 2^{m^2}\},$$

where ω_γ is an edge weight function.

Recall that in the graph F_m^2 there are three types of edges, namely, highway edges, edges of paths $\mathcal{P}^j$, and star spokes. In all the weight functions ω_γ, all the edges of the highway $\mathcal{H}$ and the paths $\mathcal{P}^j$ are assigned the weight 0. The spokes of all stars except S_0 and S_m are assigned the weight 4. The spokes of the star S_m are assigned the weight 2.

The only differences between different weight functions ω_γ occur on the m^2 spokes of the star S_0. Specifically, each of these m^2 spokes is assigned a weight of either 1 or 3; thus there are 2^{m^2} possible combinations of weight assignments. (See Figure 5.)

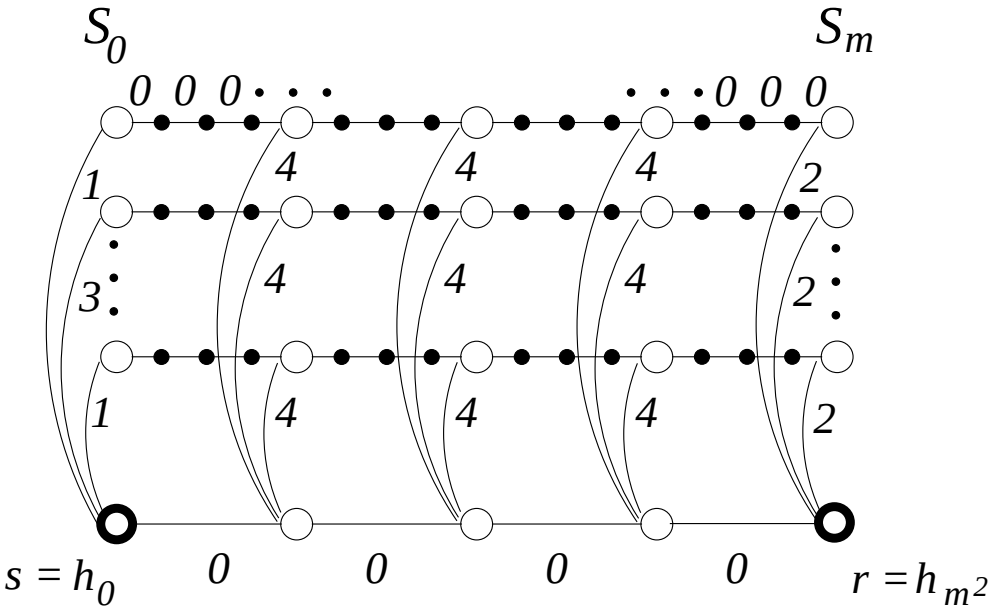


FIG. 5. *The edge weights assigned to* $J_{m,\gamma}^2$.

Since discarding all spoke edges of weight 4 from the graph $J_{m,\gamma}^2$ leaves it connected, and since every spoke edges of weight 4 is the heaviest edge on some cycle in the graph, the following is clear.

LEMMA 4.1. *No spoke edge of weight* 4 *belongs to the MST of* $J^2_{m,\gamma}$ *for every* $1 \le \gamma \le 2^{m^2}$.

Let us pair the spoke edges of S_0 and S_m, denoting the jth pair (for $1 \le j \le m^2$) by

$$PE^j = \{(s, v_0^j), (r, v_{m^2}^j)\}.$$

LEMMA 4.2. *For every* $1 \le \gamma \le 2^{m^2}$ *and* $1 \le j \le m^2$, *exactly one of the two edges of* PE^j *belongs to the MST of* $J^2_{m,\gamma}$, *namely, the lighter one.*

Proof. Since the MST must be connected, at least one of the two edges of PE^j must belong to it, as otherwise the path $\mathcal{P}^j$ is completely disconnected from the rest of the graph, by Lemma 4.1. It remains to show that the MST cannot contain both edges of PE^j.

The proof is by contradiction. Consider the cycle in $J^2_{m,\gamma}$ consisting of the edges of $\mathcal{H}$, PE^j, and $\mathcal{P}^j$, and suppose that *both* edges of PE^j are in the MST. In order for the MST to be cycle-free, at least one edge e of either the highway $\mathcal{H}$ or the path $\mathcal{P}^j$ must not belong to the MST. Since the edges of $\mathcal{H}$ and $\mathcal{P}^j$ have zero weight, $\omega_\gamma(e) = 0$. Hence deleting the heavier edge of the pair PE^j and adding the edge e instead leaves us with a lighter tree than the original one, leading us to contradiction. □

LEMMA 4.3. *For every* $m \ge 2$ *and* $1 \le \gamma \le 2^{m^2}$, *all the edges of the highway* $\mathcal{H}$ *and the paths* $\mathcal{P}^j$, *for* $1 \le j \le m^2$, *belong to the MST of* $J^2_{m,\gamma}$.

Figure 6 illustrates the remaining candidate edges to join the MST. Bold edges belong to the MST under any edge weight function. Of the remaining edges, exactly one of each pair will join the MST, depending on the particular weight assignment to the spoke edges of the star S_0.

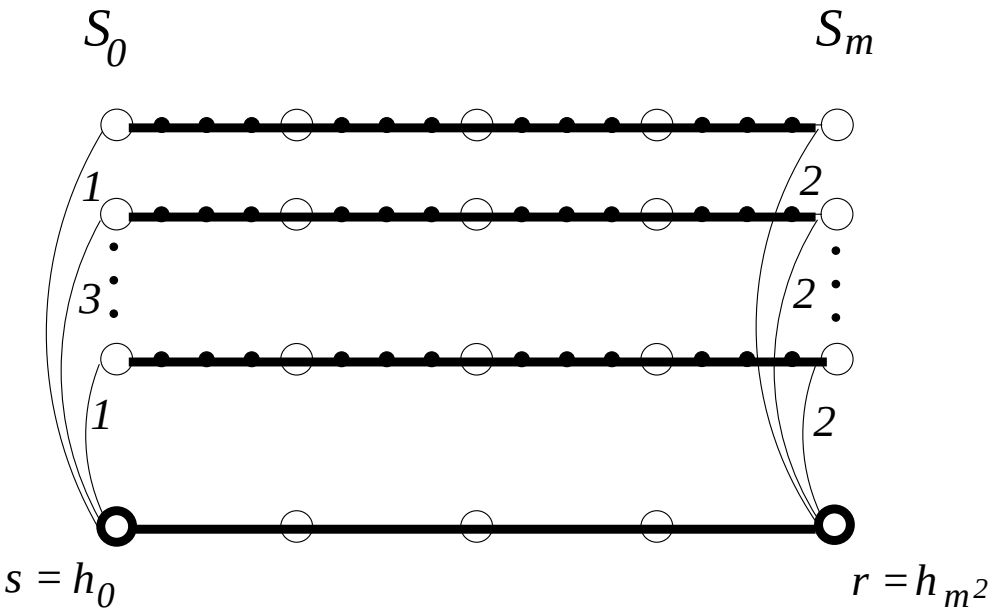


FIG. 6. *The remaining candidate edges to join the MST of* $J^2_{m,\gamma}$. *Bold edges belong to the MST under any edge weight function.*

4.2. The lower bound.

LEMMA 4.4. *Any distributed algorithm for constructing an MST on the graphs of the class $\mathcal{J}_m^2$ can be used to solve the* $\texttt{Mail}(F_m^2, h_0, h_{m^2}, m^2)$ *problem on F_m^2 with the same time complexity.*

Proof. Consider an algorithm $\mathcal{A}_{\rm mst}$ for the MST problem, and suppose that we are given an instance of the $\texttt{Mail}(F_m^2, h_0, h_{m^2}, m^2)$ problem with input string $\mathcal{X}$. We use the algorithm $\mathcal{A}_{\rm mst}$ to solve this instance of the mailing problem as follows. The sender $s = h_0$ initiates the construction of an instance of the MST by turning F_m^2 into a weighted graph from $\mathcal{J}_m^2$, setting the edge weights as follows: for each $x_i \in \mathcal{X}$, $1 \le i \le m^2$, it sets the weight variable W_i^s corresponding to the spoke edge $e_i \in E(S_0)$ to be

$$W_i^s = \begin{cases} 3, & x_i = 1, \\ 1, & x_i = 0. \end{cases}$$

The rest of the graph edges are assigned fixed weights as specified in subsection 4.1. Note that the weights for all the edges except those connecting s to its immediate neighbors in S_0 do not depend on the particular input instance at hand. Hence a single round of communication between s and its S_0 neighbors suffices for performing this assignment; s assigns its edges weights according to its input string $\mathcal{X}$, and needs to send at most one message to each of its neighbors on S_0, to notify it about the weight of the spoke connecting them.

Every vertex v in the network, upon receiving the first message of algorithm $\mathcal{A}_{\rm mst}$, assigns the values defined by the edge weight function ω_γ to its variables W_i^v. (As discussed earlier, this does not require v to know γ, as its assignment is identical under all weight functions ω_γ, $1 \le \gamma \le 2^{m^2}$.) From this point on, v may proceed with executing algorithm $\mathcal{A}_{\rm mst}$ for the MST problem.

Once algorithm $\mathcal{A}_{\rm mst}$ terminates, the receiver vertex r determines its output for the mailing problem, by setting $X_i^r \leftarrow Y_i^r$ for $1 \le i \le m^2$.

By Lemma 4.2, the lighter edge of each pair PE^j, for $1 \le j \le m^2$, belongs to the MST; thus in the set of variables $Y_1^r, \ldots, Y_{m^2}^r$ obtained by the vertex r as a result of solving the MST problem, $Y_j^r = 1$ corresponds to the assignment of $\omega(h_0, v_0^j) = 3$ to the jth edge of S_0, while $Y_j^r = 0$ corresponds to the assignment of 1 to that edge; hence for every j, Y_j^r equals x_j, the jth bit of $\mathcal{X}$. Hence the resulting algorithm has correctly solved the given instance of the mailing problem. □

Combined with Lemma 3.4, we now have the following theorem.

THEOREM 4.5. *For every $m \ge 1$, any distributed algorithm for constructing an MST on the graphs of the family $\mathcal{J}_m^2$ in the B model for $B \ge 3$ requires $\Omega(m^2/B)$ time.* □

COROLLARY 4.6. *Any distributed algorithm for the MST problem in the B model for $B \ge 3$ requires $\Omega(\sqrt{n}/B)$ time on some n-vertex graphs of diameter $O(n^{1/4})$.*

5. A lower bound on the mailing problem on F_m^K. This section generalizes the results of the previous section to the graphs F_m^K for $K \ge 3$, thus establishing the desired lower bound.

5.1. The graphs F_m^K. Given two integer parameters $m, K \ge 2$, construct the graph F_m^K as follows. The two basic units are still the path and the highway, with the following changes. The basic path $\mathcal{P}$ now has $m^K + 1$ vertices, i.e.,

$$V(\mathcal{P}) = \{v_0, \ldots, v_{m^K}\} \quad \text{and} \quad E(\mathcal{P}) = \{(v_i, v_{i+1}) \mid 0 \le i \le m^K - 1\}.$$

There are $K-1$ highways, denoted $\mathcal{H}^1,\ldots,\mathcal{H}^{K-1}$. The level-$\ell$ highway $\mathcal{H}^\ell$ consists of $m^\ell+1$ vertices, i.e.,

$$V(\mathcal{H}^\ell)=\{h^\ell_{im^{K-\ell}} \mid 0\le i\le m^\ell\} \quad \text{and}$$
$$E(\mathcal{H}^\ell)=\{(h^\ell_{im^{K-\ell}},h^\ell_{(i+1)m^{K-\ell}}) \mid 0\le i\le m^\ell-1\}.$$

Each highway vertex $h^\ell_{im^{K-\ell}}$ is connected to the corresponding path vertex $v^j_{im^{K-\ell}}$ by a spoke edge. Figure 7 depicts these connections for the case of $m=K=3$.

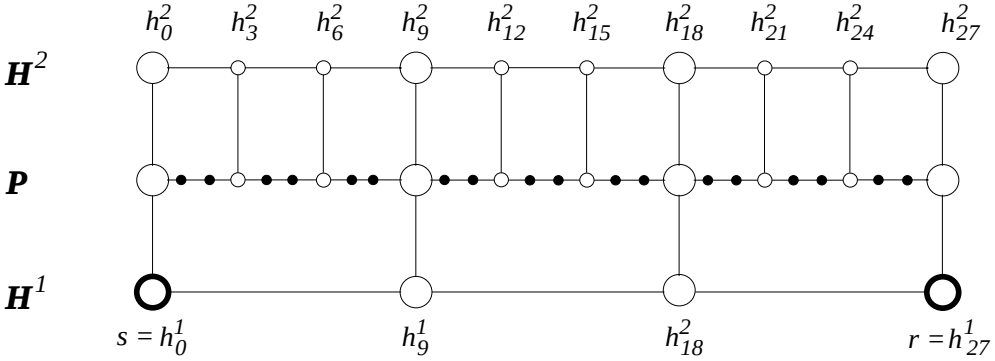


FIG. 7. *The connections between the path $\mathcal{P}$ and the highways $\mathcal{H}^1$ and $\mathcal{H}^2$ for $m=K=3$.*

The graph F^K_m is constructed by taking m^K copies of the path $\mathcal{P}$, denoted $\mathcal{P}^1,\ldots,\mathcal{P}^{m^K}$, and connecting them all to the same level-ℓ highway $\mathcal{H}^\ell$, for each $1\le\ell\le K-1$. The vertex h^1_0 is the intended sender s, and the vertex $h^1_{m^K}$ is the intended receiver r. (See Figure 8, showing the graph F^3_3.)

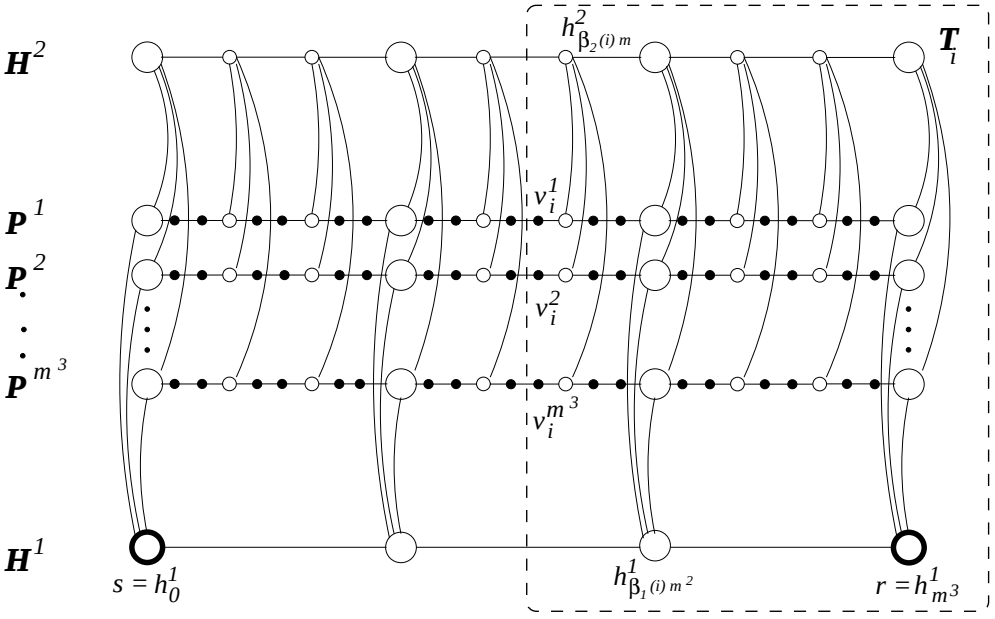


FIG. 8. *The graph F^3_m (here also $m=3$).*

The following two facts are easily verified.

LEMMA 5.1. *The cardinality of F_m^K is $n = \Theta(m^{2K})$ and its diameter is $\Theta(Km)$.*

5.2. The lower bound. The lower bound for the mailing problem can be extended from F_m^2 to F_m^K for $K \geq 3$ in a natural way. Consider some arbitrary algorithm $\mathcal{A}_{\text{mail}}$, and let $\varphi_{\mathcal{X}}$ denote the execution of $\mathcal{A}_{\text{mail}}$ on the input $\mathcal{X}$ in the graph F_m^K. The notion of a tail set is generalized to F_m^K for $K \geq 3$ as follows. For every $1 \leq j \leq m$, define the *tail* of the path $\mathcal{P}^j$ as before, i.e.,

$$T_i(\mathcal{P}^j) = \{v_l^j \mid i \leq l \leq m^K\}.$$

Let $\beta^\ell(i)$ denote the least integer δ such that $\delta m^{K-\ell} \geq i$, and define the tail of $\mathcal{H}^\ell$ as

$$T_i(\mathcal{H}^\ell) = \{h^\ell_{jm^{K-\ell}} \mid \beta^\ell(i) \leq j \leq m^\ell\}.$$

The tail set of F_m^K is the union of those tails,

$$T_i = T_i(\mathcal{H}) \cup \bigcup_j T_i(\mathcal{P}^j).$$

(See Figure 8.) Again, for $i = 0$ the definition is $T_0 = V \setminus \{h_0^1\}$.

The main lemma becomes the natural extension of Lemma 3.2, and its proof is similar. The notions of configuration, collection of configurations, and absolute size ρ of collections of possible configurations are defined in the same way as in section 3.2.

LEMMA 5.2. *For any $0 \leq t < m^K$,*

$$\rho[T_{t+1}, t+1] \leq (2^B + 1)^{K-1} \cdot \rho[T_t, t].$$

Proof. The lemma is proved by showing that in round $t+1$ of the algorithm, each configuration in $\mathcal{C}[T_t, t]$ branches off into at most $(2^B+1)^{K-1}$ different configurations of $\mathcal{C}[T_{t+1}, t+1]$.

Fix a configuration $\hat{C} \in \mathcal{C}[T_t, t]$. The tail set T_{t+1} is connected to the rest of the graph by the highway edges $f^\ell_{\beta^\ell(t+1)-1} = (h^\ell_{(\beta^\ell(t+1)-1)m^{K-\ell}}, h^\ell_{\beta^\ell(t+1)m^{K-\ell}})$, for every $1 \leq \ell \leq K-1$, and by the m^K path edges $e_t^j = (v_t^j, v_{t+1}^j)$, $1 \leq j \leq m^K$.

Consider the number of different configurations in $\mathcal{C}[T_{t+1}, t+1]$ that may result from $\hat{C}$. Starting from the configuration $\hat{C}$, each vertex v_t^j is restricted to a single state, and hence it sends a single (well determined) message over the edge e_t^j to v_{t+1}^j, thus not introducing any divergence in the execution. The same applies to all the edges internal to T_{t+1}.

The situation with highway edges $f^\ell_{\beta^\ell(t+1)-1}$ is different as there are K possible cases. When $\beta^\ell(t+1) = \beta^\ell(t)$ for $1 \leq \ell \leq K-1$, the vertices $h^\ell_{(\beta^\ell(t)-1)m^{K-\ell}}$ are not in the set T_t; hence their state is not defined by the choice of $\hat{C}$. The value passed over the edge $f^\ell_{\beta^\ell(t)-1}$ into the set T_{t+1} is thus unknown. However, due to the restriction of the B-bounded-message model, at most 2^B+1 different behaviors of can be observed by each vertex $h^\ell_{\beta^\ell(t)m^{K-\ell}}$, resulting in 2^B+1 possible states for each such node. Considering the entire set $\{h^\ell_{\beta^\ell(t)m^{K-\ell}} \mid 1 \leq \ell \leq K-1\}$, the single state $\hat{C}$ results in $(2^B+1)^{K-1}$ states of the tail set T_{t+1} at time $t+1$.

In the rest of the cases, $\beta^\ell(t+1) = \beta^\ell(t) + 1$ for at least one ℓ, when passing to the next tail set causes the exclusion of the highway point $h^\ell_{\beta^\ell(t)m^{K-\ell}}$ from the tail set. Here, a well defined message is sent over $f^\ell_{\beta^\ell(t+1)-1}$ since the state of $h^\ell_{\beta^\ell(t)m^{K-\ell}}$

is defined by the configuration $\hat{C}$. It follows that in these cases the number of possible states of T_{t+1} is less than $(2^B+1)^{K-1}$.

Altogether, the configuration $\hat{C}$ branches off into at most $(2^B+1)^{K-1}$ possible configurations $\hat{C}_1, \ldots, \hat{C}_{(2^B+1)^{K-1}} \in \mathcal{C}[T_{t+1}, t+1]$, differing by the states $\sigma(h^\ell_{\beta^\ell(t+1)m^{K-\ell}}, t+1, \mathcal{X})$. The lemma follows. □

COROLLARY 5.3. *For any* $0 \leq t < m^K$,

$$\rho[T_t, t] \leq (2^B+1)^{(K-1)t}.$$

Letting t_{end} denote the time it takes algorithm $\mathcal{A}_{\text{mail}}$ to complete the mailing, we derive, similar to the proof for $K=2$, that necessarily

$$(2^B+1)^{(K-1)t_{end}} \geq \rho[T_{t_{end}}, t_{end}] \geq 2^{m^K},$$

implying the following.

LEMMA 5.4. *For every* $K, m \geq 2$, *solving the mailing problem* $\texttt{Mail}(F^K_m, h^1_0, h^1_{m^K}, m^K)$ *in the* B *model requires* $\Omega(m^K/(BK))$ *time.*

6. A generalized lower bound on the MST on $\mathcal{J}^K_m$. Finally, we show the lower bound for the MST problem on the weighted versions of the graphs F^K_m.

6.1. The graph families $\mathcal{J}^K_m$. Let us define the families of weighted graphs $\mathcal{J}^K_m$. For every two integers $m, K \geq 2$, let

$$\mathcal{J}^K_m = \{J^K_{m,\gamma} = (F^K_m, \omega^K_\gamma) \mid 1 \leq \gamma \leq 2^{m^K}\},$$

where ω^K_γ is the weight function defined as follows. All the edges of the highways $\mathcal{H}^\ell$ for $1 \leq \ell \leq K-1$ and the paths $\mathcal{P}^j$ for $1 \leq j \leq m^K$ are assigned zero weight. It remains to assign the weights to the spoke edges.

Consider a subgraph of F^K_m, consisting of all the available paths $\mathcal{P}_j$, $1 \leq j \leq m^K$, a single highway $\mathcal{H}^\ell$ and all the connections of this highway to the paths. Consider a single node $h^\ell_{im^{K-\ell}}$ of the highway $\mathcal{H}^\ell$ and the set of all its connections to $\mathcal{P}^1, \ldots, \mathcal{P}^{m^K}$. Following the terminology of the case $K=2$, this is termed the *level-*ℓ *star* $S^{K,\ell}_{m,i}$. There are m^ℓ such stars at level ℓ.

Consider the collection of the stars $S^{K,\ell}_{m,i}$ for $1 \leq i \leq m^\ell$, $2 \leq \ell \leq K-1$ (excluding the first star of each level ℓ). The spokes of these stars are assigned the weight 4. The spokes of the first star of each level, $S^{K,\ell}_{m,0}$, are assigned as follows. The spokes connecting the star centers h^ℓ_0, $1 \leq \ell \leq K-1$ to the extreme vertex v^1_0 of the path $\mathcal{P}^1$ are assigned zero weight. The rest of the spokes are assigned the weight 4.

For the collection of the level-1 stars, $S^{K,1}_{m,i}$, the assignment is as follows. The spokes of all the stars except the two extreme ones, $S^{K,1}_{m,0}$ and $S^{K,1}_{m,m}$, are assigned the weight 4. The spokes of the last star $S^{K,1}_{m,m}$ are assigned weight 2. The weight assignment to the m^K spokes of the star $S^{K,1}_{m,0}$ depends on the particular function ω^K_γ, with each spoke assigned a value of either 1 or 3, as in section 4.2.

LEMMA 6.1. *No spoke edge of weight* 4 *belongs to the MST of* $J^K_{m,\gamma}$ *for every* $1 \leq \gamma \leq m^K$.

Proof. Following the proof of Lemma 4.1, it can be shown that the elimination of all spoke edges of weight 4 from the graph $J^K_{m,\gamma}$ leaves the graph connected. Since all the edges of all the highways and basic paths have zero weight, none of their edges is eliminated. Consider the connectivity of the highway $\mathcal{H}^1$ and the basic paths

$\mathcal{P}^1, \ldots, \mathcal{P}^{m^K}$. By construction, the spokes of two stars, namely, $S^{K,1}_{m,0}$ and $S^{K,1}_{m,m}$, have edges of weight at most 3, which guarantees that every basic path is connected to $\mathcal{H}_1$. The rest of the highways $\mathcal{H}^\ell$, for $2 \leq \ell \leq K-1$, are connected to the node v^1_0 of the basic path $\mathcal{P}^1$ by their nodes h^ℓ_0 via a zero-weight edge. Thus all the highways are connected to the path $\mathcal{P}^1$ and through it to the highway $\mathcal{H}^1$ and all the other basic paths.

Hence every spoke edge of weight 4 occurs as the heaviest edge on some cycle in the graph, implying the lemma. □

LEMMA 6.2. *For every $2 \leq \ell \leq K-1$, the edge (h^ℓ_0, v^1_0) belongs to the MST.*

Proof. By Lemma 6.1, no spoke edge of weight 4 belongs to the MST. By construction, each of the highways $\mathcal{H}^\ell$ for $2 \leq \ell \leq K-1$ is connected to the rest of the graph by spokes of weight 4 and by a single zero-weight spoke of the star $S^{K,\ell}_{m,0}$ connecting it to $\mathcal{P}^1$. Thus in order for the MST to be connected, the zero-weight edge must belong to the MST. □

Let us pair the spoke edges of $S^{K,1}_{m,0}$ and $S^{K,1}_{m,m}$ connecting $\mathcal{H}^1$ to $\mathcal{P}^j$ for $1 \leq j \leq m^K$, denoting the jth pair (for $1 \leq j \leq m^K$) by

$$PE^j = \{(s, v^j_0), (r, v^j_{m^K})\}.$$

By a proof similar to that of Lemma 4.2, we get the following lemma.

LEMMA 6.3. *For every $1 \leq j \leq m^K$, exactly one of the two edges of PE^j belongs to the MST of $J^K_{m,\gamma}$, namely, the lighter one.*

6.2. The generalized lower bound on distributed MST. We obtained an instance of the MST problem, in which the membership of edges in the MST is predetermined for all but the m^K edge pairs PE^j. Following the proof method of Lemma 4.4, we show that any algorithm solving the distributed MST problem on $\mathcal{J}^K_m$ can be used for solving the mailing problem in the same time complexity, implying the following.

THEOREM 6.4. *For every $m, K \geq 2$, any distributed algorithm for constructing an MST on the graphs of the family $\mathcal{J}^K_m$ in the B model for $B \geq 3$ requires $\Omega(m^K/(BK))$ time.*

Proof. Consider an algorithm $\mathcal{A}_{\mathrm{mst}}$ for the MST problem, and suppose that we are given an instance of the $\texttt{Mail}(F^K_m, h^1_0, h^1_{m^K}, m^K)$ problem with input string $\mathcal{X}$. We use the algorithm $\mathcal{A}_{\mathrm{mst}}$ to solve this instance of the mailing problem as follows. The sender $s = h^1_0$ initiates the construction of an instance of the MST by turning F^K_m into a weighted graph from $\mathcal{J}^K_m$, setting the edge weights as follows: for each $x_i \in \mathcal{X}$, $1 \leq i \leq m^K$, it sets the weight variable W^s_i corresponding to the spoke edge $e_i \in E(S^{K,1}_{m,0})$ (the first 1-level star), to be as in the proof of Lemma 4.4. The rest of the graph edges are assigned fixed weights as specified in section 6.1. Note that again, the weights for all the vertices except s and its immediate neighbors in $S^{K,1}_{m,0}$ do not depend on the particular input instance at hand; hence a single round of communication between s and its $S^{K,1}_{m,0}$ neighbors suffices for performing this assignment.

From this point on, we may proceed with executing algorithm $\mathcal{A}_{\mathrm{mst}}$ for the MST problem. Once algorithm $\mathcal{A}_{\mathrm{mst}}$ terminates, the receiver r determines its output for the mailing problem, by setting $X^r_i \leftarrow Y^r_i$ for $1 \leq i \leq m^K$.

The fact that the resulting algorithm has correctly solved the given instance of the mailing problem is established as in the proof of Lemma 4.4, relying on Lemma 6.3. The theorem now follows from Lemma 5.4. □

COROLLARY 6.5. *For every $K \geq 2$, there exists a family of n-vertex graphs of diameter $O(Kn^{1/(2K)},)$ such that any distributed algorithm for the MST problem in the B model for $B \geq 3$ requires $\Omega(\sqrt{n}/(BK))$ time on some of those graphs.*

COROLLARY 6.6. *For every $n \geq 2$, there exists a family of n-vertex graphs of diameter $O(\log n)$ such that any distributed algorithm for the MST problem in the B model for $B \geq 3$ requires $\Omega(\sqrt{n}/(B \log n))$ time on some of those graphs.*

Finally, let us comment that it has recently been shown that using Yao's method [Yao77] it is possible to extend the lower bound of Lemma 5.4 on the mailing problem into a lower bound on the expected time complexity of any *randomized* (Las Vegas) distributed algorithm for the mailing problem (see [P00, Chapter 24, Exercise 9]). This, in turn, yields the following lower bound on the time complexity of randomized algorithms for distributed construction: For every $n \geq 2$, there exists a family of n-vertex graphs of diameter $O(\log n)$ such that any randomized Las Vegas distributed algorithm for the MST problem in the B model requires $\Omega(\sqrt{n}/(B \log n))$ expected time on some of those graphs.

7. Open problems. Several interesting problems can be considered for future research. The first direction concerns the limitations of the presented lower bound. To begin with, the lower bound does not seem to extend to diameters lower than $O(\log n)$. As graphs with $D = 1$ admit an $O(\log n)$ distributed algorithm for MST construction, one may expect an interesting interdependence between the time to construct an MST and the network's diameter.

Second, one may consider a model allowing L-bit edge weights for $L > B$. While our lower bound still holds, stronger bounds may apply. Note that the transmission of an edge weight can be carried out in this model by sending $\Theta(L/B)$ separate messages. Hence each of the existing algorithms for distributed MST can be adapted to this model with a multiplicative slowdown of L/B. The algorithm of [KP98], for instance, will have time complexity $O((D + \sqrt{n} \log^* n)L/B)$. However, it is less clear whether this slowdown is necessary or if it can be avoided. It seems easy to verify (say, by considering a ring with two diametrically opposing edges having the extreme weights) that $\Omega(L/B)$ is indeed a lower bound on the time complexity of the problem in this model. However, it is plausible that the algorithm of [KP98] can be modified using pipelining ideas to yield a time complexity close to $O(D + \sqrt{n} \log^* n + L/B)$.

Another research direction is to try to reduce the *communication* complexity of the nearly time optimal algorithm of [KP98] from $O(|E| + n^{3/2})$ towards the lower bound of $O(|E| + n \log n)$.

Finally, one may consider the possibility of devising faster algorithms that construct an *approximation* to the MST, namely, a spanning tree whose total weight is near-minimum. To the best of our knowledge, nothing nontrivial is currently known about this problem, and little is known about distributed approximation algorithms in general, but this direction may well deserve further study.

REFERENCES

[A87] B. AWERBUCH, *Optimal distributed algorithms for minimum-weight spanning tree, counting, leader election and related problems*, in Proceedings of the 19th Symposium on Theory of Computing, ACM, New York, 1987, pp. 230–240.

[GHS83] R. GALLAGER, P. HUMBLET, AND P. SPIRA, *A distributed algorithm for minimum-weight spanning trees*, ACM Trans. Programming Languages Systems, 5 (1983), pp. 66–77.

[GKP98] J. A. GARAY, S. KUTTEN, AND D. PELEG, *A sublinear time distributed algorithm for minimum-weight spanning trees*, SIAM J. Comput., 27 (1998), pp. 302–316.

[KP98] S. Kutten and D. Peleg, *Fast distributed construction of small k-dominating sets and applications*, J. Algorithms, 28 (1998), pp. 40–66.

[L92] N. Linial, *Locality in distributed graph algorithms*, SIAM J. Comput., 21 (1992), pp. 193–201.

[P90] D. Peleg, *Time-optimal leader election in general networks*, J. Parallel Distrib. Comput., 8 (1990), pp. 96–99.

[P00] D. Peleg, *Distributed Computing: A Locality-Sensitive Approach*, SIAM, Philadelphia, PA, 2000.

[PR99] D. Peleg and V. Rubinovich, *A near-tight lower bound on the time complexity of distributed MST construction*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1999, pp. 253–261.

[Yao77] A. Yao, *Probabilistic computations: Towards a unified measure of complexity*, in Proceedings of the 17th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, CA, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1443–1461

FAST AND OPTIMAL PARALLEL MULTIDIMENSIONAL SEARCH IN PRAMS WITH APPLICATIONS TO LINEAR PROGRAMMING AND RELATED PROBLEMS*

MARTIN E. DYER† AND SANDEEP SEN‡

Abstract. We describe a deterministic parallel algorithm for linear programming in fixed dimension d that takes poly($\log\log n$) time in the common concurrent read concurrent write (CRCW) PRAM model and does optimal $O(n)$ work. In the exclusive read exclusive write (EREW) model, the algorithm runs in $O(\log n \cdot \log\log^{d-1} n)$ time. Our algorithm is based on multidimensional search and effective use of approximation algorithms to speed up the basic search in the CRCW model. Our method also yields very fast poly($\log\log n$) algorithms for smallest enclosing sphere and approximate ham-sandwich cuts and an $O(\log n)$ time work-optimal algorithm for exact ham-sandwich cuts of separable point sets. For these problems, in particular for fixed-dimensional linear programming, $o(\log n)$ time efficient deterministic PRAM algorithms were not known until very recently.

Key words. parallel algorithms, computational geometry, linear programming

AMS subject classifications. 68W10, 68W40, 65D18, 90C05

PII. S0097539797325727

1. Introduction.

We consider the standard linear programming problem: given a set $\mathcal{H}$ of n half-spaces in $\mathbb{R}^d$ and a vector y, find a point $x \in \bigcap \mathcal{H}$ that minimizes $y \cdot x$. We restrict ourselves to the case where n is much larger than d, and we focus attention on parallel algorithms that achieve good performance with respect to n. Since the general problem is known to be P-complete [12], the restricted version assumes more relevance. For the most part we will regard d as a constant. However, since the running time of our algorithm grows rapidly with d, we will attempt to examine the exact nature of this dependence. Megiddo [32] described a linear-time algorithm for linear programming in fixed dimension d (hereafter referred to as LPd), using an elegant multidimensional search technique. The same search technique, also known as prune-and-search, yielded optimal algorithms for other optimization problems (see Megiddo [31] and Dyer [13, 14]). Following Megiddo's linear-time algorithm, there have been significant improvements in the constant factor (which was doubly exponential in d) due to Dyer [14] and Clarkson [4]. Further progress was made, using random sampling, by Clarkson [5]. This algorithm was later derandomized by Chazelle and Matoušek [9]. With more careful analysis and some additional ideas, Matoušek, Sharir, and Welzl [30] improved the algorithm further, achieving an expected running time of $O(d^2 \cdot n + e^{O(\sqrt{d \log d})})$, a "subexponential" dependence on d. Independently, Kalai [26] discovered an algorithm with matching performance, and these are presently the fastest known algorithms.

In the context of parallel algorithms for LPd, there have been a number of results in the direction of finding a fast analogue of Megiddo's linear-time sequential

*Received by the editors August 12, 1997; accepted for publication (in revised form) June 12, 2000; published electronically November 8, 2000. Preliminary versions of the main results of this paper appeared in [*Proceedings of the* 11th *ACM Symposium on Computational Geometry*, Vancouver, Canada, 1995, pp. 345–349.] and [*Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, Padua, Italy, 1996, pp. 251–260.].

http://www.siam.org/journals/sicomp/30-5/32572.html

†School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK (dyer@scs.leeds.ac.uk).

‡Department of Computer Science and Engineering, Indian Institute of Technology, New Delhi 110016, India (ssen@cse.iitd.ernet.in).

method [1, 2, 11, 34]. A straightforward parallelization of Megiddo's method yields an $O(\log^d n)$ time n-processor exculsive read exclusive write (EREW) PRAM algorithm. (We write $\log^k n$ for $(\log n)^k$, and, similarly, $\log\log^k n$ for $(\log\log n)^k$.) However, this is far from the best that can be achieved. Alon and Megiddo [2] obtained an optimal parallel algorithm that runs in constant time (i.e., time depending only on d) on an n-processor randomized concurrent read concurrent write (CRCW) PRAM model. They used a variant of Clarkson's algorithm [5], which has smaller constants (in terms of d) than Megiddo's. More recently, Ajtai and Megiddo [1] developed a deterministic $O(\log\log^d n)$ algorithm in an n processor model which allows $O(\log\log n)$ time selection. Since there is an $\Omega(\log n/\log\log n)$ bound for exact selection [3], their model is considerably more powerful than the CRCW model, and partly nonuniform. It may also be noted that the work bound in the algorithm of Ajtai and Megiddo is superlinear, since they use a linear number of processors. Thus an $o(\log n)$ time optimal deterministic algorithm for LPd had hitherto proved elusive. Very recently, Goodrich [19] (see also Goodrich and Ramos [20]) independently obtained a poly$(\log\log n)$ time CRCW algorithm with optimal speed-up using fast parallel derandomization techniques. His approach is different from ours, being directly based on parallel derandomization techniques. His underlying randomized algorithm is similar to that of Dyer and Frieze [16]. This is derandomized using fast deterministic constructions of ε-approximations in $\mathbb{R}^d$. Here, we generalize the basic multidimensional search algorithm of Megiddo. Instead of pruning a constant fraction of the constraints in each round, we increase the rate of pruning as a function of *processor advantage*. The processor advantage is proportional to the ratio of the number of processors to the number of constraints, which increases monotonically during the course of the search. This allows us to improve the $\Omega(\log n)$ phases seemingly required by Megiddo's approach, and leads to a fast EREW algorithm. However, it does not directly yield a poly$(\log\log n)$ CRCW algorithm for the following reason. All known versions of multidimensional search algorithms rely on exact selection procedures, either deterministic or randomized. We make use of the observation that exact selection may be substituted by *approximate splitting* (defined formally in section 3) without significantly affecting the asymptotic convergence rate of the procedure. This is crucial in order to surmount the $\Omega(\log n/\log\log n)$ barrier for exact parallel selection: an impossibility in Ajtai and Megiddo' s approach. We apply this observation, using fast deterministic splitting algorithms in the CRCW model, to obtain a poly$(\log\log n)$ time algorithm. Moreover, we are able to achieve linear work simultaneously. Although all presently known poly$(\log\log n)$ approximate splitting algorithms are based on derandomization, they are relatively simpler and more efficient than constructing ε-nets in $\mathbb{R}^d$ as in Goodrich's algorithm [19]. However, it may be noted that the fastest algorithm known for approximate splitting follows from applying Goodrich's techniques. This could be used in our algorithm to save an $O(\log\log n)$ factor. However, we have chosen to ignore this in our later description. Rather, we feel that other poly$(\log\log n)$ time approximate splitting algorithms, like that of Hagerup and Raman [24], which are relatively simpler (although slower), are more in keeping with the basic simplicity of our approach. Our method also implies a very simple poly$(\log\log n)$ randomized CRCW algorithm, although this is asymptotically slower than Alon and Megiddo's algorithm. Our approach has immediate applications to other optimization problems where Megiddo's prune-and-search technique is effective. We are not aware of any previous poly$(\log\log n)$ time deterministic algorithms for problems like *minimum enclosing sphere* and *ham-sandwich cuts*. The latter especially has numerous applications to divide-and-conquer algorithms, including those for convex hulls and range

searching. The paper is organized as follows. In section 2, we describe our basic approach, generalizing Megiddo's multidimensional search technique. This directly yields a fast EREW algorithm. In section 3, we modify the algorithm of section 2, substituting some of the exact procedures by their faster approximate counterparts. In section 4, we analyze the algorithms for the EREW and CRCW PRAM and bound the total work by $O(n)$. In section 5, we present further applications of our search algorithm to problems like *Euclidean* 1-*center* and ham-sandwich cuts for separable planar point-sets. Because of the close similarity of these algorithms to that for LPd, we omit detailed analysis. In section 6, we make some observations regarding the relationship between *hyperplane cuttings* and our approach. We show the existence of a direct geometric construction for cuttings that does not require derandomization. However, the size of this cutting is probably too large for general application.

2. A parallel multidimensional search method. Without loss of generality, we assume that LPd has feasible region

$$\left\{ x \in \mathbb{R}^d : x_d \leq \sum_{j=1}^{d-1} \alpha_{ij} x_j + b_i \quad (i = 1, 2, \ldots, n) \right\},$$

and the objective is to minimize x_d. Megiddo [32] noted that LPd can be viewed as determining the *critical* linear constraints which define the optimal point x^*. If we assume nondegeneracy (no $(d+1)$-hyperplanes intersect in a point of the feasible region), then exactly d constraints define the optimal point, and the remaining constraints may be eliminated without affecting the optimum value. In its full generality, the LPd problem also requires us to report if the optimum is unbounded or if the feasible region is empty. In the description that follows, we do not discuss these conditions explicitly, noting that Megiddo's approach can handle these cases effectively. Suppose we have a set of n hyperplanes

$$H_i = \{ x \in \mathbb{R}^d : a_i \cdot x = b_i \} \quad (i = 1, 2, \ldots, n).$$

The *sign* of H_i with respect to $y \in \mathbb{R}^d$ is defined as $\{-, +, =\}$ depending on

$$a_i \cdot y - b_i \begin{array}{c} < \\ = \\ > \end{array} 0.$$

Geometrically, we wish to determine which side of the hyperplane contains y (including the possibility that y may lie on the hyperplane). We would like to determine the sign of y with respect to all the hyperplanes H_i $(i = 1, 2, \ldots n)$. To do this, we extend our definition to the sign of an *arbitrary* hyperplane in a similar fashion. Megiddo showed that, by determining the signs of a constant number $A(d)$ of carefully chosen hyperplanes, we can determine the sign of a constant fraction $B(d)$ of the H_i's. Here $A(d)$ and $B(d)$ are functions of d only. By repeatedly applying this strategy, one can determine all the signs relatively quickly (as compared to evaluating them individually). Megiddo used an elegant inductive argument to show the existence of $A(d)$ and $B(d)$, starting from the observation that for $d = 1$, $A(1) = 1$ and $B(1) = 1/2$ (by determining the sign of the median half-line). If y is x^*, and the hyperplanes are the constraints of LPd, then determining the sign of all hyperplanes is clearly equivalent to solving LPd. Hereafter, we will be interested in determining the sign of a hyperplane (not necessarily a constraint) with respect to x^*. However, x^* is itself unknown at the outset. Megiddo showed that the sign of an arbitrary hyperplane can

be determined by solving at most three linear programming problems (recursively) in one lower dimension. Using a nontrivial recursive strategy (involving recursion in both n and d), Megiddo constructed an algorithm for LPd with running time linear in n but doubly exponential in d. For convenience, we assume that no constraint hyperplane is parallel to x_d. (This can be achieved by standard perturbation techniques, if necessary.) From here, we focus on the following *hyperplane location* problem. Given a set N of hyperplanes of the form

$$H_i = \left\{ x \in \mathbb{R}^d : x_d = \sum_{j=1}^{d-1} a_{ij} x_j + b_i \right\} \quad (i = 1, 2, \ldots, n),$$

we wish to determine the sign at x^* of all H_i. This is often referred to as the *multidimensional search* problem. We will say that H_i is *located* if we know the sign of H_i with respect to the linear programming optimum x^*. As already noted, a straightforward parallelization of Megiddo's algorithm yields an $O(\log^d n)$ time algorithm with n processors. It is inefficient because, in the later stages of the algorithm when relatively few constraints remain, most of the processors are idle. We now describe a parallelization of multidimensional search which uses processors more efficiently in the later stages, when the number of processors greatly exceeds the number of constraints. The algorithm proceeds in stages, where during each stage, we determine the sign of some hyperplanes. At the beginning of the stage i, we denote the set of hyperplanes by N_i and their number by n_i, and we define processor advantage r_i to be $\max\{2, (p/n_i)^{1/d}\}$, where p is the number of processors. For example, $n_0 = n$ and $r_0 = 2$ if $p \leq n$. Again, for simplicity of presentation, we assume that r_i is an integer (or else take the floor function). Also, in the remaining part of this section we will write N, n, and r instead of N_i, n_i, and r_i since the description is limited to iteration i for some i.

Par_Mul_Search(N, p, d)

1. The set N of hyperplanes is partitioned into r equal sized sets based on their $a_{\ell 1}$ values ($\ell = 1, 2, \ldots, n$). This is done by selecting the $k(n/r)$th ranked elements (denoted by σ_k) from $\{a_{11}, a_{21} \ldots a_{n1}\}$ for $k = 1, 2, \ldots, r$. For convenience, write $\sigma_0 = -\infty$. Denote this partition of N by $\mathcal{P}$, where the kth *class* of $\mathcal{P}$ consists of hyperplanes H_ℓ which satisfy $\sigma_{k-1} < a_{\ell 1} \leq \sigma_k$ ($k = 1, 2, \ldots, r$).
2. For $d = 1$, solve the problem directly and exit. (See below.) Otherwise, partition N into r-sized subsets (*groups*) by picking (for each group) one constraint from each class of $\mathcal{P}$. Consider two hyperplanes H_i and H_j in a group and say $a_{i1} \leq a_{j1}$. Since they come from different classes of $\mathcal{P}$, there are values σ_k, σ_l such that $\sigma_k \in [a_{i1}, a_{j1}]$ and $\sigma_l \notin [a_{i1}, a_{j1}]$.
3. Using the transformation
$$x_1' = x_1 + \sigma_k \cdot x_2 \text{ and } x_2' = x_1 + \sigma_l \cdot x_2,$$
we obtain two hyperplanes H_{ij} and H_{ji}, where H_{ij} (respectively, H_{ji}) is obtained by eliminating x_1' (respectively, x_2') between H_i and H_j. Intuitively, these are planes that pass through the intersection of H_i and H_j and are parallel to the x_1' and x_2' coordinate axes, respectively. There are two such hyperplanes for each pair in the group. From Megiddo's observation (see also Dyer [14]), it follows that if we can locate both H_{ij} and H_{ji} (which are hyperplanes in $\mathbb{R}^{d-1}$), then we can locate at least one of H_i and H_j. Figure 1 illustrates the situation in two dimensions.

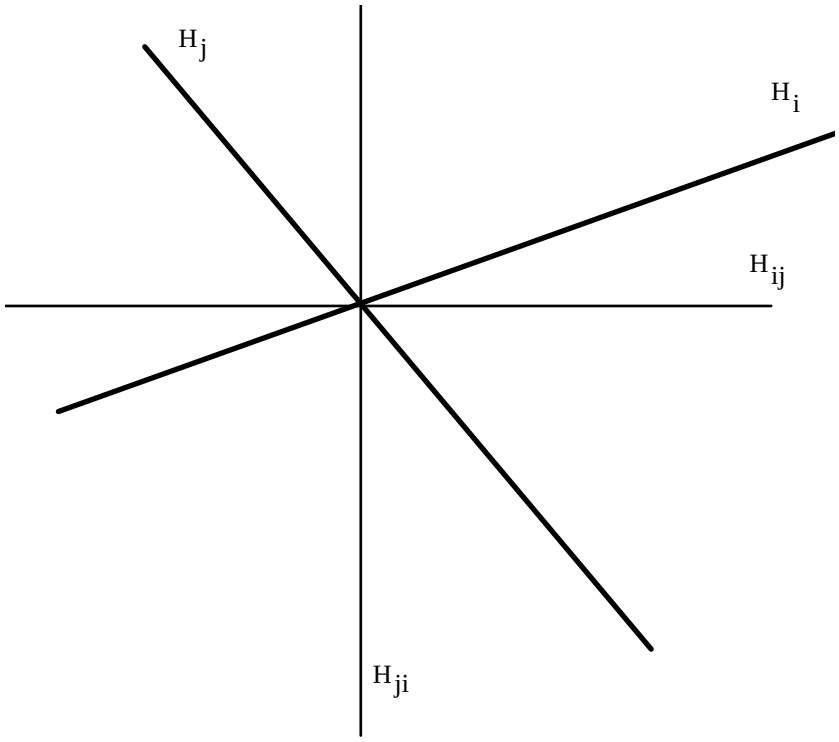


FIG. 1. *Each quadrant is completely contained in a half-plane determined by H_i or H_j. So by determining the signs of both H_{ij} and H_{ji}, we can determine the sign of H_i or H_j. For example, if the optimum lies in the NW quadrant, H_i's sign is determined. (H_{ij} and H_{ji} are not necessarily perpendicular.)*

4. We recursively apply the search algorithm in one lower dimension to all such hyperplanes H_{ij}, H_{ji}. Note that all H_{ij} lie in a subspace corresponding to the absence of $x_1{}'$ but possible presence of $x_2{}'$. Note there are only r different variables of the type $x_1{}'$, one for each of the r different σ_k values. Thus there are only r distinct subspaces for which the algorithm must be called recursively. The algorithm is called in parallel for all the distinct subspaces, allocating processors to the subproblems proportional to their sizes. At the bottom of this recursion on d, we will generate *special* hyperplanes with respect to which we must locate x^*. Location for these special hyperplanes involves recursively solving three $(d-1)$-dimensional linear programming problems. Each such problem is a restriction of the original LPd to a particular hyperplane. (See Megiddo [32] for further details of these lower-dimensional problems.)
5. Repeat the previous step R_d times, where R_d is a function of d that we will determine during the analysis. By repeating this step, we eliminate a fraction of the remaining hyperplanes.
6. The located hyperplanes are now eliminated from consideration, the new processor advantage is determined, and the above procedure is repeated.

The total number of hyperplanes generated in step 2 is $(n/r)r(r-1) < nr$ since every pair of hyperplanes in a group generates two lower-dimensional hyperplanes. Step 4 actually involves a double recursion, one of which we have unfolded (i.e., how to generate the special hyperplanes inductively in $\mathbb{R}^d$) for exposition of the basic

algorithm. This is similar to Megiddo's exposition [32] of the sequential algorithm, where a certain proportion $B(d)$ of hyperplanes is located using $A(d)$ queries, these functions being defined inductively. However, instead of the number of queries, we bound the number of *rounds* of parallel recursive calls. The number R_d will actually be defined according to the following recursive scheme. For the same set of σ_k's and r, we do c rounds of location (where c is some suitable constant) for the H_{ij}' s, where a certain proportion of H_{ij}'s are located recursively. This is done in order to boost the proportion of hyperplanes located in d dimensions for inductive application. This idea has been used before in [4, 14]. The difference here is that we cannot wait for the results of location in one group before we start on the next, as is done in the sequential algorithm. The number of groups r may be large, and so we must operate on them all in parallel. For $d = 1$, the constraints are half-lines, and the linear programs can be solved directly. The problem is equivalent to extremal selection (selection of minimum or maximum), which can be solved in $O(\log n)$ time on an EREW PRAM and $O(\log\log n)$ time on a CRCW PRAM [25]. As constraints are eliminated in each repetition of step 4, data compaction must be done, to ensure efficient processor utilization in the next. The processor requirement is determined by the total size of all problems that have to be solved simultaneously on all levels of the recursion on d. We must show that this remains bounded by the total number of processors. We will show this below and, more significantly, we will show that the recursive procedure in d dimensions takes only $O(\log\log n)$ rounds to eliminate all constraints. This will lead to an overall time bound of $O(\log n \log\log^{d-1} n)$ for LPd on the EREW PRAM.

3. Multidimensional search using approximate computations. To circumvent the $\Omega(\log n/\log\log n)$ lower bound for exact selection in the CRCW PRAM, we will modify step 1 of the previous section. We propose to use approximate splitting in the partitioning step and approximate compaction to do load balancing. As noted previously, extremal selection can be done in $O(\log\log n)$ time in CRCW PRAM. Formally, the problem of *approximate splitting* is defined as follows.

> *Given a set of N of n elements, an integer r, $1 \le r \le n$, and an expansion factor $\theta > 1$, choose a set R of r elements (out of N) such that the maximum number of elements of N in any interval induced by (the sorted order of) R is less than $\theta \cdot n/r$.*

Intuitively, $\theta = 1$ gives even splitting but we will relax that significantly to speed up our algorithm. For approximate splitting on the CRCW model, we use the following result implied by the work of Hagerup and Raman [24] and Goldberg and Zwick [22] (see the appendix for details).

LEMMA 3.1. *For all given integers $n \ge 4$, and $C \le r \le n^{1/4}$, approximate splitting can be done with expansion factor $\sqrt{r}$ in $O(\log\log^2 n)$ time and $O(r \cdot n)$ CRCW processors where C is a sufficiently large constant.*

For smaller r, we partition the a_{i2}'s into two sets by an approximate median which can be found quickly using the following result of Goldberg and Zwick.

LEMMA 3.2. *For a given set X of n elements and any fixed $\epsilon > 0$, an element $x \in X$ whose rank r' satisfies $\frac{n}{2} \le r' \le \frac{n}{2}(1+\epsilon)$ can be found in $O(\log\log n)$ steps using $n/\log\log n$ CRCW processors.*

The problem of *approximate compaction* was defined by Hagerup [23] as follows.

> *Given an n-element set, of which a are* active*, place the active elements in an array of size $(1+\lambda)a$, where λ is the* padding factor.

As one might expect, the running time of fast parallel algorithms for this problem increases as λ decreases. Hagerup described an $O(\log\log^3 n)$ time work-optimal CRCW

algorithm for the above problem for $\lambda = 1/\text{poly}(\log\log n)$. The following improved result was recently obtained by Goldberg and Zwick [22].

LEMMA 3.3. *The approximate compaction problem can be solved on a CRCW PRAM in time* $t \geq \log\log n$, *using* n/t *processors, with a padding factor of* $1/\text{poly}(\log\log n)$.

Suppose we substitute exact compaction by approximate compaction. Then, over the $O(d)$ levels of recursion, there will only be a slowdown in running time by a factor of $(1+\lambda)^{O(d)}$, i.e., a constant. We will also use the following result on *semisorting* to collect the elements of subproblem (i.e., a class or a group) together. The semisorting problem may be defined as follows.

> *Let* $\mathcal{A}$ *be an array of* n *elements such that each element has an integer label* $\ell \in \{1, 2, \ldots, k\}$. *Sort the elements of* $\mathcal{A}$ *into disjoint subarrays* $\mathcal{A}_\ell$ $(\ell = 1, 2, \ldots, k)$ *such that, if there are* n_ℓ *elements with label* ℓ, *the subarray*$\mathcal{A}_\ell$ *must be of size* $O(n_\ell)$.

The following is known about semisorting from [24].

LEMMA 3.4. *Semisorting problems of array size* n *and range size* k *can be solved in* $O(\log\log n)$ *CRCW time using* kn *processors.*

So, if C is the constant for the splitting algorithm above, the modified algorithm is as follows. If $(p/n_i)^{1/d} \geq \beta$, we choose $r_i = (p/n_i)^{1/d}$; otherwise, we choose $r_i = 2$, where $\beta \geq C$ is a constant that we will determine below, and we use the method of Lemma 3.1 to partition the a_{i1}'s, $1 \leq i \leq n$, into classes. Once the splitters are determined, the partitioning can be done by brute force, comparing a_{i1} against all splitters simultaneously in parallel. (There are sufficient processors for this to be done.) We put the hyperplanes in each class in near-contiguous locations by an application of semisorting and approximate compaction. Then we give each hyperplane in a class a group number corresponding to its position in the derived subarray. We now form groups by semisorting on group number, followed by approximate compaction of the group subarray. Thus each group will pick at most one hyperplane from each class, each hyperplane will belong to exactly one group, and each group is held in an array a little longer than its size. There are sufficient processors throughout since each uncompacted array is only a constant factor larger than the group size, by Lemma 3.4. The per-processor overhead caused by arrays not being exactly compacted is a small constant (in fact $o(1)$), and hence we ignore it in the analysis. Note, however, that some groups may have less than r hyperplanes. The number of groups is determined by the size of the largest partition since we take one hyperplane from each partition. Because of the approximate splitting, there could be $n/\sqrt{r}$ groups, rather than the n/r which would result from exact splitting. However, the size of each group is no more than r because the number of classes is only r. We will denote the set of groups by $\mathcal{G}$, and their number by $|\mathcal{G}|$. As indicated in the previous section, we pair hyperplanes from a group and solve the lower dimensional problem recursively a constant number of times, c (to be determined below). Then we start a new iteration by throwing out constraints and compacting the remaining using Lemma 3.3. At the bottom level (when the hyperplanes are points), we choose r splitters, so that the size of the largest partition is at most $n/\sqrt{r}$. Location with respect to the r splitters can be done simultaneously in $O(\log\log n)$ time using extremal selection. We have chosen r so that we will have enough processors at the bottom level.

4. Analysis. First we will prove a slightly weaker result, that our algorithm takes $O(\log\log^{d+1} n)$ in a CRCW PRAM using $p = n$ processors, and subsequently reduce the number of processors. The analysis is generalized to handle approximate

splitting. Therefore, in particular, the bounds that we prove are applicable to the basic algorithm of section 2. For convenience of presentation, we will refer to hyperplanes in $\mathbb{R}^d$ as d-hyperplanes. Our first objective is to bound the number of rounds of hyperplane locations required to eliminate a constant fraction of d-hyperplanes. Let π_d denote the fraction of d-hyperplanes that are located in R_d rounds for a fixed r_i, say r. Since the maximum size of a partition at the bottom level is $n/\sqrt{r}$, $\pi_1 \geq (\sqrt{r}-1)/\sqrt{r}$. The following will be used to write a recurrence for π_d. The groups refer to the groupings of $\mathcal{G}$ as defined in the previous section.

LEMMA 4.1. *The number of $(d-1)$-hyperplanes that have to be located in the recursive call is bounded by $n \cdot (r-1)$.*

Proof. As pointed out in the previous section, although $|\mathcal{G}|$ could be $n/\sqrt{r}$ because of uneven splitting, the number of $(d-1)$-hyperplanes can be bounded by $\sum_{g\in\mathcal{G}} n_g(n_g-1)$, where n_g is the number of d-hyperplanes in group $g \in \mathcal{G}$. Since $n_g \leq r$ and $\sum n_g = n$, the number of $(d-1)$-hyperplanes is bounded by $n/r \cdot r(r-1)$. □

Observation 4.1. Suppose, in a certain group of d-hyperplanes, at most $m(m-1)/2$ of the $(d-1)$-hyperplanes have not been located. Then at most m d-hyperplanes are not located.

Proof. Recall that the $(d-1)$-hyperplanes are obtained by taking all possible pairs in a group and generating two $(d-1)$-hyperplanes from each pair. Suppose there are $m'(>m)$ unlocated d-hyperplanes; consider the $m'(m'-1)/2 > m(m-1)/2$ pairs formed by these. At least one $(d-1)$-hyperplane from each pair has not been located, since locating both $(d-1)$-hyperplanes in a pair would imply that one of the d-hyperplanes is located. This contradicts the antecedent of the observation that at most $m(m-1)/2$ pairs have not been located. □

In order to write a recurrence for π_d, we let $\bar{n}_g$ denote the number of unlocated d-hyperplanes in group g and let u_g denote the number of unlocated $(d-1)$-hyperplanes (among the $n_g(n_g-1)$ generated in g). From the above notations,

$$\pi_d = \frac{1}{n}\left(n - \sum_g \bar{n}_g\right).$$

From Lemma 4.1 and the recursive application (i.e., a fraction π_{d-1} of the $(d-1)$-hyperplanes are located in R_{d-1} rounds),

$$\sum_g u_g \leq (1-\pi_{d-1})^c \cdot n(r-1). \tag{1}$$

From Observation 4.1,

$$u_g \geq \frac{1}{2}\bar{n}_g \cdot (\bar{n}_g - 1),$$

implying

$$2u_g + 1 \geq \bar{n}_g^2 - \bar{n}_g + 1,$$

implying, in turn,

$$2u_g + 1 \geq n_g.$$

Therefore, $\sum_g \bar{n}_g \leq \sum_g (2u_g + 1)$. Substituting in (1), we can write the recurrence for π_d as follows:

$$\pi_d \geq \frac{1}{n}(n - 2(1 - \pi_{d-1})^c n(r-1) - n/\sqrt{r}), \tag{2}$$

where the last term on the right corresponds to $\sum_g 1 = |\mathcal{G}| \leq n/\sqrt{r}$. Simplifying, we obtain

$$\pi_d \geq 1 - 2(1 - \pi_{d-1})^c (r-1) - 1/\sqrt{r}.$$

It is straightforward to verify by induction that, for $d \geq 2$, $\pi_d \geq \frac{\sqrt{r}-2}{\sqrt{r}}$, provided $r \geq 8$ and $c \geq 12$. If $r = 2$, we choose an approximate median with accuracy $1/6$ (i.e., $\epsilon = 1/6$ in Lemma 3.2). Then (2) becomes

$$\pi_d \geq \frac{1}{n}(n - 2(1 - \pi_{d-1})^c \cdot n - 2n/3),$$

since $|\mathcal{G}| \leq n/2 + n/6 = 2n/3$. For $c = 12$ and $\pi_1 \geq 1/3$, $\pi_d \geq 1/4$ by induction. Thus we take $c = 12$ and $\beta = \max\{8, C\}$ in our algorithm description where C is the constant from Lemma 3.1. The total number of rounds R_d for location in dimension d satisfies $R_d = c^{d-1}$, since $R_d = c \cdot R_{d-1}$ and $R_1 = 1$. Hence we can write the recurrence for n_i, the number of surviving constraints at the beginning of iteration i, as

$$n_{i+1} \leq n_i(1 - \pi_d) \leq \begin{cases} \frac{3}{4} n_i & (r_i = 2), \\ \frac{2}{\sqrt{r_i}} n_i & (r_i \geq \beta). \end{cases}$$

After $O(d)$ iterations with $r_i = 2$, we will have $(n/n_i)^{1/d} \geq \beta$. Note that then $n_i/n \leq \frac{1}{8}$. Thereafter,

$$n_{i+1} \leq 2n_i^{1+1/2d}/n^{1/2d}.$$

Letting $\xi_i = n_i/n$, we can rewrite the previous inequality as

$$\xi_i \leq 2\xi_i^{1+1/2d},$$

where $\xi_1 = \frac{1}{8}$. Therefore, $\xi_i = O(1/n)$ (i.e., $n_i = O(1)$) when $i \geq c_1 \log\log n$ for some constant $c_1 = O(d)$. Each iteration involves splitting, compaction, and a constant number of recursive calls to lower dimensional problems. Let $T_d(n)$ be the running time for a d-dimensional problem of size n. Then we can write the recurrence

$$T_d(n) \leq c_1 \log\log n[c^{d-1}(T_{alloc} + T_{split} + 3T_{d-1}(n))], \tag{3}$$

where T_{alloc} and T_{split} denote the times for data compaction and approximate splitting, respectively. From Lemma 3.3, $T_{alloc} = O(\log\log n)$, and from Lemma 3.1, $T_{split} = O(\log\log^2 n)$, respectively. Using $T_1(n) = O(\log\log n)$, $T_d(n)$ is $O(\log \log^{d+1} n)$ for $d \geq 2$. Since $c_1 = O(d)$, the implied constant is $2^{O(d^2)}$, which is of the same form as that in the improvement of Megiddo's algorithm due to Dyer [14] and Clarkson [4]. The processor requirement is bounded by the number of $(d-1)$-dimensional linear programming problems which must be solved simultaneously at the bottom level. Each such problem has n_i constraints per problem. This number is the

total number of hyperplanes that must be located simultaneously. This is bounded by $n_i(r_i-1)^{d-1}$ from Lemma 4.1, since the number of distinct linear programming instances grows by an (r_i-1) factor at each level of the recursion on d. But this is at most n, since either $r_i = 2$ or $r_i = (n/n_i)^{1/d} \geq 8$. We can now state our intermediate result as the following lemma.

LEMMA 4.2. *Linear programming in fixed dimension $d \geq 2$ can be solved in $O(\log \log^{d+1} n)$ CRCW PRAM steps, using n processors .*

Remark. Using a faster approximate splitting algorithm due to Goodrich [19], with $T_{split} = O(\log \log n)$, the running time would decrease to $O(\log \log^d n)$. The previous analysis also holds for the basic algorithm of section 2, and therefore we can analyze the performance of the algorithm on the EREW PRAM simply by substituting bounds for exact selection and compaction. Since then $T_{alloc} = O(\log n)$ and $T_{split} = O(\log n)$, we obtain, similarly to the above, the following result.

LEMMA 4.3. *Linear programming in fixed dimension $d \geq 2$ can be solved in $O(\log n \log \log^{d-1} n)$ steps using n processors in an EREW PRAM.*

To reduce the number of operations to $O(n)$, we use a slow-down technique, applied inductively. We outline our strategy in the case of CRCW PRAM. We have observed that the one-dimensional problem can be solved in $O(\log \log n)$ steps using $n/\log \log n$ processors. Assume that the $(d-1)$-dimensional problem can be solved in $O(\log \log^d n)$ steps optimally. The standard slow-down method implies that, for any time $t > \log \log^d n$, these problems can be solved with optimal work. We start with $p = n/\log \log^{d+1} n$ processors and reduce the number of constraints to $n/\log \log^{d+1} n$ by running the algorithm with $r_i = 2$ for $O(\log \log \log n)$ iterations. Let $t_d = \log \log^{d+1} n$. In each iteration, we apply the work-optimal $(d-1)$-dimensional method. Now iteration i (starting with $i = 0$) takes $\max\{\gamma^i t_d, O(\log \log^d n)\}$ steps. Here $\gamma \leq \frac{3}{4}$ is the fraction of constraints eliminated at each iteration. In each iteration, we use the result of Lemma 3.3 to do load balancing. This takes $O(\max\{\gamma^i t_d, \log \log n\})$ steps. The total time taken to reduce the number of constraints to $n/\log \log^{d+1} n$ by the above method can therefore be bounded by $O(\log \log^{d+1} n + \log \log^d n \cdot \log \log \log n)$. At this stage, we switch to the n-processor algorithm described previously. Therefore, we can state our final result.

THEOREM 4.4. *Linear programming in fixed dimension $d \geq 2$ can be solved in $2^{O(d^2)} \log \log^{d+1} n$ CRCW PRAM steps using $n/\log \log^{d+1} n$ processors.*

Following an identical strategy, but using $t_d = \log n \log \log^{d-1} n$ and $O(\log n)$ for the load balancing time, we obtain an analogous result for the EREW PRAM. However, the running time increases by an $O(\log^* n)$ factor, since the best work-optimal EREW selection algorithm known runs in $O(\log n \log^* n)$ time (Cole [10]).

THEOREM 4.5. *Linear programming in fixed dimension $d \geq 2$ can be solved in $2^{O(d^2)} \log n \log^* n \log \log^{d-1} n$ EREW PRAM steps, using $n/(\log n \log^* n \log \log^{d-1} n)$ processors.*

5. Other applications. Although the search algorithm was described in the context of linear programming, the method extends to problems where prune-and-search methodology has produced linear-time sequential algorithms. We outline two of these applications here, namely, finding *the smallest enclosing circle* (or, more generally, *the Euclidean* 1*-center problem*) and finding *ham-sandwich cuts* of separable point-sets on the plane.

5.1. Smallest enclosing circle. Given a set $N = \{(a_i, b_i), 1 \leq i \leq n\}$ of points in the plane, we wish to determine a circle $\mathcal{C}$ that encloses all the points and

is smallest among all such circles. This has a natural analogue in higher dimension, where we determine the smallest enclosing sphere in the appropriate dimension. Megiddo [31] described a linear-time algorithm for the smallest enclosing circle problem. Dyer [14] extended this to solve the more general *weighted Euclidean* 1-*center* problem in any fixed dimension, using additional tools from convex optimization. The idea of Megiddo's solution is very similar to the linear programming algorithm. The minimum enclosing circle $\mathcal{C}$ is defined by at most three points, so the solution remains unchanged if we eliminate the remaining points. The algorithm proceeds in iterative phases, where in each phase a constant fraction of the input points are eliminated with linear work. The crucial issue is to recognize which points can be eliminated using carefully designed queries. In the following, we follow Megiddo's [31] description with appropriate modifications for the parallel algorithm. We will first solve a restricted version of the problem, where the center of $\mathcal{C}$ is constrained to lie on a given straight line. Without loss of generality, assume this line is the x-axis. Denote the center of $\mathcal{C}$ by x_c. Pair up the points arbitrarily and denote the perpendicular bisector of pair $(a_i, b_i), (a_j, b_j)$ by $\perp_{ij}$. It can be readily seen that there exists a critical value x_{ij} (the intersection of $\perp_{ij}$ with the x-axis), such that depending on $x_c \begin{smallmatrix} < \\ = \\ > \end{smallmatrix} x_{ij}$, one of the points in the pair the one that is closer to x_c, can be eliminated. So if we can determine answers to queries of the form "Is $x \begin{smallmatrix} < \\ = \\ > \end{smallmatrix} x_c$?" for arbitrary x, we can use these to eliminate some of the points. For example, by evaluating the query at x_m, the median value of x_{ij}s, we can eliminate one point from every pair for half the number of pairs, i.e., a quarter of the points. Determining the sign of an arbitrary x ($x \begin{smallmatrix} < \\ = \\ > \end{smallmatrix} x_c$?) is easily done by finding the furthest point (in Euclidean distance) from x. Denote the square of this distance by $g(x)$. Let $I = \{i : (x - a_i)^2 + b_i^2 = g(x)\}$. If $x < a_i$ for every i, then $x < x_c$. If $x > a_i$ for every i, then $x > x_c$, else $x = x_c$. All this can be done in linear time. So, applying the above procedure recursively to the remaining points (at most $3/4n$), x_c can be determined in $O(n)$ steps. More formally, we are solving the following optimization problem (unrestricted case).

$$\min_{x,y} f(x, y) = \max_i \{(x - a_i)^2 + (y - b)^2\}.$$

Note that $f(x, y)$ is a convex function of its arguments. In the preceding paragraph, we described a method to solve the constrained problem when $y = 0$. Given an arbitrary y, the sign of y is defined to be $\{+, =, -\}$ depending on $y \begin{smallmatrix} < \\ = \\ > \end{smallmatrix} y^*$, where (x^*, y^*) is the unconstrained optimum of $g(x, y)$. Because of the convexity, the sign of y can be determined from the sign at (x_c, y), where x_c is the constrained optimum. The sign at (x_c, y) can be determined by an application of linear programming in plane or, alternatively, by a direct method of finding a line separator (see Megiddo [31]). The overall algorithm is very similar to the linear programming algorithm. Here we consider pairs of points instead of pairs of constraints. Each pair of point $(a_{2i-1}, b_{2i-1}), (a_{2i}, b_{2i})$ defines a perpendicular bisector $\perp_i$ such that, if we could determine which half-plane of $\perp_i$ contains (x^*, y^*), we could eliminate one point from the pair. This is similar to determining the sign of $\perp_i$. We pair perpendicular bisectors, $\perp_i$ and $\perp_j$, say, and compute their intersection p_{ij}. Consider the two lines X_{ij} and Y_{ij}, passing through p_{ij}, parallel to the x and y axes, respectively. If the slopes of $\perp_i$ and $\perp_j$ do not have the same signs, then by determining the signs of X_{ij} and Y_{ij}, we can eliminate one of the four points (defining $\perp_i$ and $\perp_j$). Figure 2 illustrates this situation. Determining the sign of X_{ij} or Y_{ij} is essentially a lower dimensional problem (the constrained

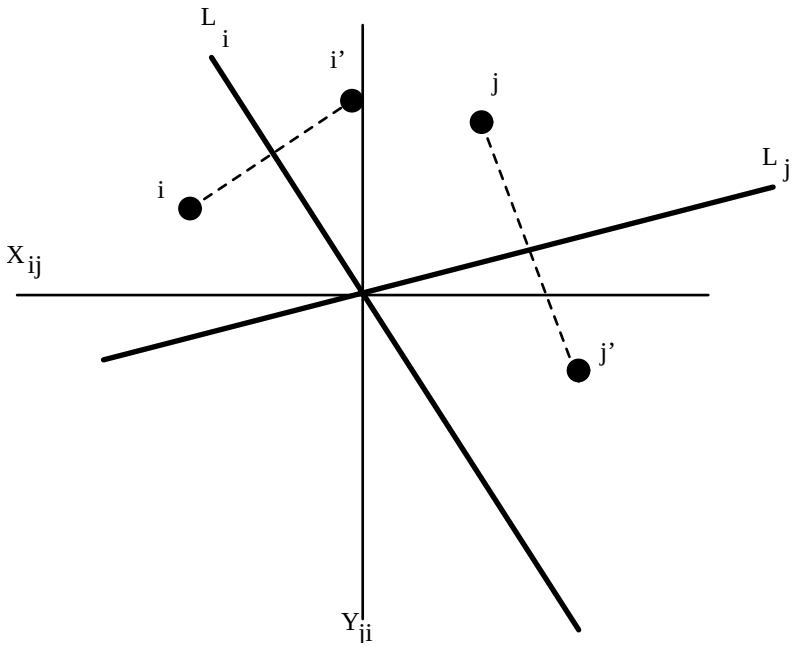


FIG. 2. *If the center of the optimum circle is located in the SW quadrant, clearly point i will be strictly closer than i′ and hence can be eliminated.*

version) that we solved before. So, we have reduced the searching problem to one lower dimension. The strategy is identical to that of computing signs of hyperplanes in section 2. Here the hyperplanes are defined by pairs of points (the perpendicular bisectors). The groupings are done on the basis of slopes and the algorithm is applied recursively to the r different subspaces generated. Here r is processor advantage, which will be defined similarly to the linear programming algorithm. The analysis is carried out in a fashion almost identical to section 4, and we omit further details. Note that computing the sign of a bisector eliminates at least one of the two defining points. Hence we can summarize with the following theorem.

THEOREM 5.1. *The minimum enclosing circle of n points on the plane can be determined in $O(\log\log^3 n)$ CRCW time, using $n/\log\log^3 n$ processors.*

The above algorithm extends to any fixed dimension d. To determine the sign of a hyperplane in dimension $d > 2$, we will use linear programming in dimension d after determining the center of the constrained smallest sphere. (The center lies on the hyperplane.) This is to determine if the center of the constrained sphere is contained within the convex hull of the points I determining the optimum. If it is contained, then we have the global unconstrained optimum. Otherwise, the optimum lies in the direction perpendicular to the hyperplane bounding the d extreme points of I. This additional complication increases the running time by an $O(\log\log n)$ factor compared with the linear programming algorithm.

COROLLARY 5.1.1. *The minimum enclosing sphere of n points in E^d for $d > 2$ can be determined in $O(\log\log^{d+2} n)$ time using $n/\log\log^{d+2} n$ CRCW processors.*

5.2. Ham-sandwich cuts and partitioning a planar point-set. A line l is called the *bisector* of a point set S if each open half-plane defined by the line contains

at most half the points of S. It is known that, given two point sets P and Q, it is possible to bisect them simultaneously using a line l. This line is called a *ham-sandwich* cut of P and Q. In the special case where P and Q are linearly separable, Megiddo [33] described a linear-time algorithm based on prune-and-search. Later, Lo and Steiger [27] described a linear-time algorithm for the general case. Megiddo's algorithm is useful in situations where we need to partition a point set into four quadrants using two lines. We can choose one line to be the median with respect to one of the axes. This partitions the points into two sets S_1 and S_2. Then, using Megiddo's algorithm, we can take the other line to be a ham-sandwich cut of S_1 and S_2. This has numerous applications to divide-and-conquer algorithms. Below we show that our search strategy yields a fast parallel algorithm for this special case of ham-sandwich cuts. However, it must be noted that ham-sandwich cuts can be used to compute an exact median of a set of points on the x-axis. (Simply take this set as P and an arbitrary two-point set in the upper half-plane as Q.) Thus the lower bound on exact median-finding prevents us from attaining poly$(\log\log n)$ performance. Therefore, we will first design an $O(\log n)$ time algorithm for this problem. Then we will show how to obtain *approximate ham-sandwich* cuts in poly$(\log\log n)$ time. An approximate ham-sandwich cut will mean that each of P and Q will be partitioned *approximately* equally. We describe Megiddo's method briefly. (Our version actually follows the description in [17].) The problem is solved in the dual space where a point $p = (a, b)$ is mapped to the line $D(p) : y = 2ax - b$, and a nonvertical line $l : y = \lambda_1 x + \lambda_2$ is mapped to the point $D(l) : (\lambda_1/2, -\lambda_2)$. Thus $D^2 = D$, and it is known, moreover, that D preserves incidence and the below-above relationship. (See [17] for details.) Thus the P and Q are mapped to sets of lines $G = D(P)$ and $H = D(Q)$. Assume, without loss of generality, that the lines of G and H have nonpositive and nonnegative slopes, respectively (by choosing the x-axis as a separating line in the primal plane). In the arrangement $\mathcal{A}(S)$ formed by a set of lines S, a *level* k $(1 \le k \le |S|)$ is the set of points of $\mathcal{A}(S)$ that lie below *exactly* k lines of S. We will write $\mathcal{L}_i(S)$ for the ith level of S. In the arrangements $\mathcal{A}(G)$ and $\mathcal{A}(H)$, the median levels correspond to lines that bisect P and Q in the primal plane. Because of the slope constraints we have imposed, the levels in G and H are monotonically nonincreasing and nondecreasing, respectively. This implies that $\mathcal{L}_i(G)$ and $\mathcal{L}_j(H)$ have a nonempty intersection for all i, j. We wish to determine a point p^* in the intersection of the median levels. Then $D(p^*)$ is a ham-sandwich cut for P and Q. We actually solve a more general problem. We determine a point s in the intersection of the ith level of G and the jth level of H. Let $m = |G|$ and $n = |H|$. The algorithm proceeds in phases, where in the kth phase, we are looking for a point common to the levels g_k of $\mathcal{A}(G^k)$ and h_k of $\mathcal{A}(H^k)$. Here G^k and H^k are the subsets of G and H active during the kth phase. Initially, $k = 0$, $g_0 = i$, $h_0 = j$, $G^0 = G$, and $H^0 = H$. In the kth phase, we eliminate lines from G^k and H^k that cannot contain the common point using line *tests*. A test consists of determining which side of a given line t contains p^*. (The test may also yield a point in the intersection, in which case the algorithm terminates.) Then new sets G^{k+1}, H^{k+1} are calculated after elimination of some lines. New values of g_{k+1} and h_{k+1} are chosen, and we proceed to iteration $k + 1$. A test with respect to a line t is similar to computing the sign in our general search strategy. It can be done in linear time sequentially and involves computing a g_kth and h_kth intersection point of G^k and H^k, respectively, on the line t. There are various cases, and we refer the reader to [17, pp. 339–341] for details. We use the following optimal selection algorithm from Chaudhuri, Hagerup, and Raman [7].

LEMMA 5.2. *For all integers $n \geq 4$, selection problems of size n can be solved in $O(\log n/\log\log n)$ CRCW time using $n\log\log n/\log n$ processors.*

Using this as a subroutine, we can compute the *sign* of a line t in $O(\log n/\log\log n)$ time. Consider the set of lines $L = H \cup G$. Suppose $l_1, l_2 \in L$ have slopes of the opposite sign and $p_{1,2}$ is their common intersection point. Let t_x and t_y be the horizontal and vertical lines through $p_{1,2}$. By determining the signs of t_x and t_y, we can determine at least one of the lines l_1 or l_2 which cannot contain any point s in the intersection of the levels. (Otherwise, we can actually determine such a point s, and the algorithm terminates.) We can therefore apply our search strategy to this situation by partitioning the lines of L using their slopes. Here we can use exact selection using Lemma 5.2 since we have more time at our disposal. The remaining algorithm is very similar to our previous search algorithms, and the analysis follows along similar lines. Since each testing costs us $O(\log n/\log\log n)$ time and the algorithm has $O(\log\log n)$ stages, the total time is $O(\log n)$ using n processors. We may also use exact compaction. Finally, using a technique similar to Theorem 4.4, we can reduce the number of processors to $O(n/\log n)$. Thus, we have the following theorem.

THEOREM 5.3. *A ham-sandwich cut of two linearly separable sets P and Q can be computed in $O(\log n)$ CRCW steps, using $n/\log n$ processors, where $n = |P \cup Q|$.*

An *approximate* ham-sandwich cut of P and Q, with *relative accuracy* λ ($0 < \lambda < 1$), will be defined as follows. The cut is a line l which simultaneously partitions P and Q, so that the partition of P (respectively, Q) does not contain more than $|P|(1+\lambda)/2$ (respectively, $|Q|(1+\lambda)/2$) points. In the context of the dual setting this implies that we have to determine a point s that lies in the intersection of $\mathcal{L}_i(G)$ and $\mathcal{L}_j(H)$, where $(1-\lambda)|G|/2 \leq i \leq (1+\lambda)|G|/2$ and $(1-\lambda)|H|/2 \leq j \leq (1+\lambda)|H|/2$. Ghouse and Goodrich [21] describe a simple algorithm for this problem using random sampling followed by verification. We follow the lines of our previous algorithms, with the modification that testing with respect to a line is done using approximate selection. We use the following algorithm for parallel approximate selection from [7].

LEMMA 5.4. *For all integers $n \geq 4$ and $t \geq \log\log^4 n$, approximate selection with relative accuracy $2^{-t/\log\log^4 n}$ can be achieved in $O(t)$ time, using a CRCW PRAM optimally in $O(n)$ operations. Furthermore, for $q \geq 1$, a relative accuracy of 2^{-q} can be achieved in $O(q + \log\log^4 n)$ time, using $O(qn)$ operations.*

We also use Lemma 3.3 to update the values of g_k, h_k at each iteration. Because of the use of approximate algorithms, testing with respect to a line t returns an answer satisfying the following property. Given g_k, h_k, and a line t, the test returns a half-plane (bounded by t) which contains an intersection point of level $(1 \pm \epsilon)g_k$ of G^k with level $(1 \pm \epsilon)h_k$ of H^k. (Here ϵ is the relative accuracy of our selection algorithm.) Moreover, the updated values of g_k and h_k are accurate within a multiplicative factor $(1 + 1/\text{poly}(\log n))$. Since there are $O(\log\log n)$ iterative phases of the algorithm, the overall accuracy can be bounded by $(1 \pm \epsilon)^{O(\log\log n)}$. From Lemma 5.4, $\epsilon \leq 1/\text{poly}(\log n)$ is achievable in $\text{poly}(\log\log n)$ steps. Hence an overall relative accuracy of $1/\text{poly}(\log n)$ is achievable using this approach. Therefore, we have the following result.

THEOREM 5.5. *An approximate ham-sandwich cut of linearly separable sets P and Q, with relative accuracy $\lambda \leq \log^{-a} n$ for some fixed constant a, can be computed in $O(\log\log^6 n)$ steps, using $O(n)$ operations, where $n = |P \cup Q|$.*

6. Some observations on our method. In this section we offer some insights into the multidimensional search technique relative to other methods, in particular, random-sampling based strategies. The randomized algorithm of Clarkson [5] (later

derandomized by Chazelle and Matoušek [9]) exploits efficient geometric partitioning methods based on ϵ-net constructions (often referred to as *cuttings* in the context of this problem). A $1/r$-*cutting* $\mathcal{C}$, for a set H of hyperplanes in $\mathbb{R}^d$, is a collection of d-dimensional simplices (with disjoint interiors) which covers $\mathbb{R}^d$. Furthermore, no simplex intersects more than $|H|/r$ hyperplanes. The number of simplices in $\mathcal{C}$ is called the *size* of the cutting. The derandomization methods are actually techniques for constructing cuttings whose existence is guaranteed by the probabilistic method. It was shown by Matoušek [29] and Chazelle [8] that $1/r$-cuttings of size $O(r^d)$ can be constructed in time $O(n \cdot r^{d-1})$, using a derandomization technique based on the method of conditional probabilities (known as the *Raghavan–Spencer* technique). It is fairly clear that, given such a cutting, the multidimensional search method becomes easier to implement. We will show that the converse is also true—that the multidimensional search method implies a cutting. This was noted somewhat less explicitly in some previous papers (see [6, 28]). While the size of the cutting is much worse than one obtains from the derandomization methods, it does not involve derandomization and is a direct geometric constructon. The intuition is as follows. The multidimensional search in d dimensions eliminates a fixed fraction $B(d)$ of hyperplanes by location with respect to a set of hyperplanes R of size $A(d)$. Let us denote the arrangement in $\mathbb{R}^d$ induced by the hyperplanes in R by $\mathcal{A}(R)$, and let us denote the region in $\mathcal{A}(R)$ that contains the optimum by Δ^*. Clearly, the number of hyperplanes intersecting Δ^* cannot exceed $|H|(1 - B(d))$, since any plane that has been located cannot intersect Δ^*. However, it is not obvious how many hyperplanes intersect other regions $\Delta \in \mathcal{A}(R)$. If the same bound applies, then we have a $(1 - B(d))$-cutting. This is indeed so, since Megiddo's algorithm selects, at the bottom-most level of recursion, a fixed set of hyperplanes and does the location with respect to these. Irrespective of where the query point lies, the previous bound holds. The size of this cutting is less obvious. We reduce the number of hyperplanes intersecting Δ^* by iterating a fixed number of times c on the lower dimensional problem. This is an adaptive improvement that is local to Δ^*, and it is not clear how it affects other regions. However, modifying (2) from section 4 as follows will give us a bound on the size of a $1/r$-cutting.

$$\pi_d \geq \frac{1}{n}(n - 2(1 - \pi_{d-1}) \cdot n(r - 1) - n/r). \tag{4}$$

We have set the exponent $c = 1$ for the reasons mentioned above, and we have replaced $n/\sqrt{r}$ with n/r since we may do exact splitting in the present context. In (4), it can be verified that $\pi_d \geq 1 - 2/r$ if $\pi_{d-1} \geq 1 - \frac{1}{2r^2}$. Applying this inductively, we find that $\pi_1 \geq 1 - \frac{1}{2^{d-1}r^{2^{d-1}}}$ implies $\pi_d \geq 1 - 2/r$. This implies a $2/r$ cutting in $\mathbb{R}^d$. For any m, a $1/m$-cutting of $\mathbb{R}^1$ has size $(m - 1)$ (by choosing $(m - 1)$ equally spaced values). Thus the overall cutting contains $2^{d-1}r^{2^{d-1}}$ hyperplanes. This is exponentially larger than the cuttings of size $O(r^d)$ obtained from ε-net constructions. However, for small constants r and d, this may be a reasonable alternative to the derandomized schemes for constructing $1/r$-cuttings. We may summarize the discussion of this section as follows.

Observation 6.1. Megiddo's algorithm implies a simple linear-time method (without derandomization) for constructing $1/r$-cuttings of size $(2^{d-1}r^{2^{d-1}})^d$ in $\mathbb{R}^d$, where d is a fixed constant.

7. Concluding remarks. This paper makes two main contributions to parallel fixed-dimensional linear programming. First, we show that an alternative implementation of Megiddo's [32] technique enables efficient processor utilization. Second, we

circumvent the bottleneck of median-finding, found in earlier adaptations of Megiddo's approach, by substituting approximate selection algorithms. (Note that even randomized selection has the same bottleneck.) However, the underlying approximate algorithms have large associated constants, and the algorithms of Goldberg and Zwick [22] use expanders. We can eliminate the need for expanders by settling for a somewhat slower algorithm (by a factor of $O(\log\log^3 n)$), using the method of Hagerup and Raman [24]. On the other hand, the running time of Theorem 5.5 can be improved somewhat by using the selection algorithm of [22]. Finally, we remark that reducing the dependence on d of the running time of our algorithm for LPd is a challenging open problem.

8. Appendix. We give a brief outline of the proof of Lemma 3.1. The ideas are adapted mainly from Hagerup and Raman [24], where the reader can find more detailed proofs. Throughout, we will use $\nu(r)$ to denote a factor of the form $1 + \frac{1}{\text{polylog}(r)}$.

Proof of Lemma 3.1. Hagerup and Raman [24] describe a method for computing r approximate splitters in $O(\log\log^5 n)$ CRCW time, using rn processors. Below we describe how to speed up their method by substituting faster routines for *approximate prefix computation* due to Goldberg and Zwick [22]. The problem *approximate prefix sum* is defined as follows.

A sequence $0 = b_0, b_1, \ldots, b_n$ is said to be an ϵ-approximate prefix sum of a given nonnegative sequence $a_1, \ldots, a_n$ if we have

$$\sum_{j=1}^{i} a_j \le b_i \le (1+\epsilon)\sum_{j=1}^{i} a_j$$

and $b_i - b_{i-1} \ge a_i$ $(1 \le i \le n)$. Lemma 3.3 is actually derived from the following result on approximate prefix sums.

LEMMA 8.1 (see [22]). *For any fixed $\alpha > 0$ and fixed $\delta > 0$, a $1/(\log^\alpha n)$-approximate prefix sum sequence can be computed in $O(1)$ CRCW time, using $n^{1+\delta}$ processors.*

Substituting the previous result in the proof of Lemma 11 of [24], we obtain the following.

LEMMA 8.2. *A set of n keys can be* padded-sorted *with* padding factor $\nu(n)$ *in $O(\log\log n)$ time, using a polynomial number of processors.*

To improve the processor bound in the previous lemma, we apply Lemma 8.1 recursively to a sample-sort algorithm. For this we compute a *group sample* as follows.

Let A be a set of n numbers, let m be an integer, and let $s = \frac{n}{m^2}$. Partition A into s groups $A_1, \ldots, A_s$, each of size m^2. Padded-sort the A_i's into subarrays of size $\nu(n)m^2$, and let B_i be the multiset consisting of the elements with rank $\nu m, 2\nu m \ldots \nu m^2$ in A_i. Then $B = \bigcup_i B_i$ is a group sample of A.

Remark. Computing a group sample of size n/m involves sorting sets of size $O(m^2)$. By choosing a group sample of appropriate size and using its members as splitters in a quicksort-like algorithm (but for only a constant depth of recursion), we obtain the following result, along the same lines as Hagerup and Raman [24].

LEMMA 8.3. *For any fixed $\epsilon > 0$, n keys can be padded-sorted in $O(\log\log n)$ CRCW time, with padding-factor $\nu(n)$, using $n^{1+\epsilon}$ processors.*

An (m, λ)-*sample* of a set A is a subset B such that, for each pair of elements

$x, y \in B$,

$$|\text{rank}_A(x) - \text{rank}_A(y)| \leq \frac{m|A|}{|B|}|\text{rank}_B(x) - \text{rank}_B(y)| + \lambda|A|.$$

By choosing a group sample $B \subseteq A$, using the procedure described previously, we can show the following.

LEMMA 8.4. *B is a $(\nu(m), \nu(m)/m)$-sample of A, computable in $O(\log\log n)$ CRCW time, using mn processors. Moreover, $|B| = n/m$.*

The proof follows by summing $(|rank_{B_i}(x) - rank_{B_i}(y)| + 1)$ over all B_i. From the definition of an (m, λ) sample, the following can easily be verified.

LEMMA 8.5 (see [24]). *Let B be an (r, λ)-sample of A, and let C be an (r', λ') sample of B. Then C is an $(rr', r\lambda' + \lambda)$ sample of A.*

We now describe the algorithm for finding r approximate splitters of a set X of n numbers. We may assume we have nr processors initially. Thus, using Lemma 8.4, we compute a subset $B_1 \subset X$, which is a $(\nu(m), \nu(m)/m)$-sample, where $m = r$. Note that $|B_1| = n/r$, so we now have a processor advantage of r^2. Next we compute B_2, which is a $(\nu(r^2), \nu(r^2)/r^2)$-sample of B_1. At stage i, we will compute a $(\nu(r^{2^i}), \nu(r^{2^i})/r^{2^i})$-sample of B_i. By induction, we can show that $|B_i| = n/r^{2^i-1}$. Now, from Lemma 8.5, we can show that B_i is a $(\prod_i \nu(r^{2^i}), \prod_i \nu(r^{2^i})/r)$-sample of X. We continue this process until $B_j \leq n^{3/4}$ for the first time, and we call this subset D. Clearly, $j < \log\log n$ and from $r \leq n^{1/4}$ it follows that $n^{1/4} \leq |D| \leq n^{3/4}$. Thus we can ensure that D is a $(\prod_i \nu(r^{2^i}), \prod_i \nu(r^{2^i})/r)$-sample of X. Since $(1+x) \leq e^x$, it follows that $\prod_i \left(1 + \frac{1}{2^i}\right) = O(1)$. So D is a $(c', c'/r)$ sample of X for some constant c'. Suppose $|D| = q$, then choose r equally spaced elements from D, and call this set S. We claim that S is a set of r elements which approximately splits X with expansion factor $\sqrt{r}$. To see this, note that between two consecutive elements of S there are q/r elements of D. Since D is a $(c', c'/r)$ sample of X, it follows (from the definition of an (m, λ)-sample) that there are a maximum of $c'(n/q)(q/r) + c'(1/r)n$ elements of X. Substituting $c'' = c'^2$, the number of elements can be bounded by $O(\sqrt{c''}n/r)$. For $r \geq c''$, this is less than $\frac{n}{\sqrt{r}}$. This completes the proof of Lemma 3.1. Note that $C = c''$ in the statement of Lemma 3.1.

Remark. The running time of the above algorithm for finding approximate splitters can be improved to $O(\log\log n)$ time by noting that only the first group sampling step takes $O(\log\log n)$ and $O(1)$ thereafter by exploiting processor advantage carefully. This was pointed out by one of the reviewers.

Acknowledgments. The second author wishes to thank Rajeev Raman for very helpful discussions regarding formulating the present version of Lemma 3.1 and Pankaj Aggarwal for bringing [19] to his attention. The authors also acknowledge several helpful comments of the anonymous reviewers that helped improve the presentation. In particular, one of the reviewers pointed out that a simple modification in the approximate splitting algorithm (described in the appendix) can improve the running time by a factor of $O(\log\log n)$.

REFERENCES

[1] M. AJTAI AND N. MEGIDDO, *A deterministic poly(log log n)-time n-processor algorithm for linear programming in fixed dimension*, SIAM J. Comput., 25 (1996), pp. 1171–1195.

[2] N. ALON AND N. MEGIDDO, *Parallel linear programming in fixed dimensions almost surely in constant time*, J. ACM, 1994, pp. 422–434.

[3] P. Beame and J. Hastad, *Optimal bounds for decision problems on the CRCW PRAMS*, in Proceedings of the 19th Annual ACM Symposium on Theory of Computing, New York, NY, 1987, pp. 83–93.

[4] K. Clarkson, *Linear programming in $O(n3^{d^2})$ time*, Inform. Process. Lett., 22 (1986), pp. 21–24.

[5] K. L. Clarkson, *Las Vegas algorithms for linear and integer programming when the dimension is small*, J. ACM, 42 (1995), pp. 488–499.

[6] T. Chan, J. Snoeyink, and C. Yap, *Primal dividing and dual pruning: Output-sensitive construction of 4-d polytopes and 3-d Voronoi diagrams*, Discrete Comput. Geom., 18 (1997), pp. 433–454.

[7] S. Chaudhuri, T. Hagerup, and R. Raman, *Approximate and exact deterministic parallel selection*, in Mathematical Foundations of Computer Science 1993, Lecture Notes in Comput. Sci. 711, Springer-Verlag, Berlin, 1993, pp. 352–361.

[8] B. Chazelle, *Cutting hyperplanes for divide-and-conquer*, Discrete Comput. Geom., 9 (1993), pp. 145–158.

[9] B. Chazelle and J. Matoušek, *On linear-time deterministic algorithms for optimization problems in fixed dimension*, in Proceedings of the 4th ACM Symposium on Discrete Algorithms, Austin, TX, 1993, pp. 281–290.

[10] R. Cole, *An optimally efficient selection algorithm*, Inform. Process. Lett., 26 (1988), pp. 295–299.

[11] X. Deng, *An optimal parallel algorithm for linear programming in the plane*, Inform. Process. Lett., 35 (1990), pp. 213–217.

[12] D. Dobkin, R. Lipton, and S. Reiss, *Linear programming is log-space hard for P*, Inform. Process. Lett., 8 (1979), pp. 96–97.

[13] M. Dyer, *Linear time algorithms for two- and three-variable linear programs*, SIAM J. Comput., 13 (1984), pp. 31–45.

[14] M. Dyer, *On a multidimensional search technique and its application to the Euclidean one-centre problem*, SIAM J. Comput., 15 (1986), pp. 725–738.

[15] M. Dyer, *A parallel algorithm for linear programming in fixed dimension*, in Proceedings of the 11th ACM Symposium on Computational Geometry, Vancouver, Canada, 1995, pp. 345–349.

[16] M. Dyer and A. Frieze, *A randomized algorithm for fixed-dimensional linear programming*, Math. Programming, 44 (1989), pp. 203–212.

[17] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, EATCS Monographs on Theoretical Computer Science 10, Springer-Verlag, Berlin, 1986.

[18] M. Goodrich, *Geometric partitioning made easier even in parallel*, in Proceedings of the 9th ACM Symposium on Computational Geometry, San Diego, CA, 1993, pp. 73–82.

[19] M. Goodrich, *Fixed-dimensional parallel linear programming via relative ε-approximations*, in Proceedings of the Seventh ACM–SIAM Symposium on Discrete Algorithms, Atlanta, GA, 1996, pp. 132–141.

[20] M. Goodrich and E. Ramos, *Bounded-Independence Derandomization of Geometric Partitioning with Applications to Parallel Fixed-Dimensional Linear Programming*, Discrete Comput. Geom., 18 (1997), pp. 397–420.

[21] M. Ghouse and M. T. Goodrich, *In-place techniques for parallel convex-hull algorithms*, in Proceedings of the 3rd ACM Symposium on Parallel Algorithms and Architectures, Hilton Head, SC, 1991, pp. 192–203.

[22] T. Goldberg and U. Zwick, *Optimal deterministic approximate parallel prefix sums and their applications*, in Third Israel Symposium on the Theory of Computing and Systems (Tel Aviv, 1995), IEEE Computer Society Press, Los Alamitos, CA, 1995, pp. 220–228.

[23] T. Hagerup, *Fast deterministic processor allocation*, in Proceedings of the 4th ACM Symposium on Discrete Algorithms, Austin, TX, 1993, pp. 1–10.

[24] T. Hagerup and R. Raman, *Fast deterministic approximate and exact parallel sorting*, in Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, Velen, Germany, 1993, pp. 1–10.

[25] J. Ja' Ja', *Introduction to Parallel Algorithms*, Addison–Wesley, Reading, MA, 1992.

[26] G. Kalai, *A subexponential randomized simplex algorithm*, in Proceedings of the 24th ACM Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 475–482.

[27] C. Y. Lo and W. Steiger, *An optimal-time algorithm for ham-sandwich cuts in plane*, in Proceedings of the 2nd Canadian Conference on Computational Geometry, 1990, pp. 5–9.

[28] J. Matoušek, *Computing the center of planar point sets*, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 6, Amer. Math. Soc., Providence, RI, 1991, pp. 221–230.

[29] J. Matoušek, *Cutting hyperplane arrangements*, Discrete Comput. Geom., 13 (1991), pp. 385–406.
[30] J. Matoušek, M. Sharir, and E. Welzl, *A subexponential bound for linear programming*, Algorithmica, 16 (1996), pp. 498–516.
[31] N. Megiddo, *Linear time algorithm for linear programming in R^3 and related problems*, SIAM J. Comput., 12 (1983), pp. 759–776.
[32] N. Megiddo, *Linear programming in linear time when dimension is fixed*, J. ACM, 31 (1984), pp. 114–127.
[33] N. Megiddo, *Partitioning with two lines in the plane*, J. Algorithms, 6 (1985), pp. 430–433.
[34] S. Sen, *Finding an approximate median with high probability in constant parallel time*, Inform. Process. Lett., 34 (1990), pp. 77–80.
[35] S. Sen, *Parallel multidimensional search using approximate algorithms: With applications to linear programming and related problems*, in Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, Padua, Italy, 1996, pp. 251–260.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1462–1484

ON THE RELATIVE COMPLEXITY OF RESOLUTION REFINEMENTS AND CUTTING PLANES PROOF SYSTEMS*

MARIA LUISA BONET†, JUAN LUIS ESTEBAN†, NICOLA GALESI†, AND JAN JOHANNSEN‡

Abstract. An exponential lower bound for the size of tree-like cutting planes refutations of a certain family of conjunctive normal form (CNF) formulas with polynomial size resolution refutations is proved. This implies an exponential separation between the tree-like versions and the dag-like versions of resolution and cutting planes. In both cases only superpolynomial separations were known [A. Urquhart, *Bull. Symbolic Logic,* 1 (1995), pp. 425–467; J. Johannsen, *Inform. Process. Lett.*, 67 (1998), pp. 37–41; P. Clote and A. Setzer, in *Proof Complexity and Feasible Arithmetics,* Amer. Math. Soc., Providence, RI, 1998, pp. 93–117]. In order to prove these separations, the lower bounds on the depth of monotone circuits of Raz and McKenzie in [*Combinatorica,* 19 (1999), pp. 403–435] are extended to monotone real circuits.

An exponential separation is also proved between tree-like resolution and several refinements of resolution: negative resolution and regular resolution. Actually, this last separation also provides a separation between tree-like resolution and ordered resolution, and thus the corresponding superpolynomial separation of [A. Urquhart, *Bull. Symbolic Logic,* 1 (1995), pp. 425–467] is extended.

Finally, an exponential separation between ordered resolution and unrestricted resolution (also negative resolution) is proved. Only a superpolynomial separation between ordered and unrestricted resolution was previously known [A. Goerdt, *Ann. Math. Artificial Intelligence,* 6 (1992), pp. 169–184].

Key words. resolution, cutting planes proof system, computational complexity, proof complexity, circuit complexity

AMS subject classifications. 03F20, 68Q17, 68T15

PII. S0097539799352474

1. Introduction.

The motivation for research on the proof length of propositional proof systems is double. First, by the work of Cook and Reckhow [10] we know that the claim that *for every propositional proof system there is a class of tautologies that have no polynomial size proofs* is equivalent to $NP \neq co\text{-}NP$. This connection explains the interest in developing combinatorial techniques to prove lower bounds for proof systems. The second motivation comes from the interest in studying efficiency issues in automated theorem proving. The question is which proof systems have efficient algorithms to find proofs. Actually, the proof system most widely used for implementations is resolution or refinements of resolution. Our work is relevant to both motivations. On one hand, all the separation results of this paper improve previously known superpolynomial separations to exponential. On the other hand,

*Received by the editors February 26, 1999; accepted for publication (in revised form) August 8, 2000; published electronically November 8, 2000. A preliminary version of this paper appeared in *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science,* IEEE Computer Society, Los Alamitos, CA, 1998, pp. 638–647 and as Electronic Colloquium on Computational Complexity TR98-035.

http://www.siam.org/journals/sicomp/30-5/35247.html

†Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 08034 Barcelona, Spain (bonet@lsi.upc.es, esteban@lsi.upc.es, galesi@lsi.upc.es). The first author was partially supported by projects SPRIT 20244 ALCOM-IT, TIC 98-0410-C02-01, and PB98-0937-C04-03. The second author was partially supported by project KOALA:DGICYT:PB95-0787. The third author was supported by a European Community grant under the TMR project.

‡Institut für Informatik, Ludwig Maximilians Universität München, München, Germany (jjohanns@informatik.uni-muenchen.de). The research of this author was done at the Department of Mathematics, University of California, San Diego, and was supported by DFG grant Jo 291/1-1.

these exponential separations harden the known results showing inefficiency of several widely used strategies for finding proofs, especially for the resolution system.

Haken [16] was the first to prove exponential lower bounds for unrestricted resolution. He showed that the pigeonhole principle requires exponential size resolution refutations. Urquhart [28] found another class of tautologies with the same property. Chvátal and Szemerédi [7] showed that in some sense, almost all classes of tautologies require exponential size resolution proofs (see [2, 3] for simplified proofs of these results). These exponential lower bounds are bad news for automated theorem proving, since they mean that often the time used in finding proofs will be exponentially long in the size of the tautology, just because the shortest proofs are exponentially long in the size of the tautology.

Many strategies for finding resolution proofs are described in the literature (see, e.g., Schöning's textbook [27]). One commonly used type of strategy is to reduce the search space by defining restricted versions of resolution that are still complete. Such restricted forms are commonly referred to as *resolution refinements*. One particularly important resolution refinement is tree-like resolution. Its importance stems from the close relationship between the complexity of tree-like resolution proofs and the runtime of a certain class of satisfiability testing algorithms, the so-called *DLL Algorithms* (cf. [24, 1]). We prove an exponential separation between tree-like resolution and unrestricted resolution (Corollary 4.3), thus showing that finding tree-like resolution proofs is not an efficient strategy for finding resolution proofs. Until now only superpolynomial separations were known [29, 8].

We also consider three more of the most commonly used resolution refinements: negative resolution, regular resolution, and ordered resolution. We show an exponential separation between tree-like resolution and each one of the above restrictions. (See Corollary 4.3 for negative resolution and Corollary 4.6 for both regular and ordered resolution.)

Goerdt [14, 13, 15] gave several superpolynomial separations between unrestricted resolution and some refinements of resolution; in particular, he gave a superpolynomial separation between ordered resolution and unrestricted resolution. In this paper we consider the case of ordered resolution and we improve his separation to exponential. We prove that a certain conjunctive normal form (CNF) formula requires exponential size ordered resolution refutations but can be refuted with a polynomial size negative resolution proof (Corollary 5.7), thus, in particular, showing that unrestricted resolution can have an exponential speed-up over ordered resolution.

The cutting planes proof system, CP from now on, is a refutation system based on manipulating integer linear inequalities. Exponential lower bounds for the size of CP refutations have already been proven. Impagliazzo, Pitassi, and Urquhart [17] proved exponential lower bounds for tree-like CP. Bonet, Pitassi, and Raz [6] proved a lower bound for the subsystem CP*, where the coefficients appearing in the inequalities are polynomially bounded in the size of the formula being refuted. This is a very important result because all known CP refutations fulfill this property. Finally, Pudlák [23] and Cook and Haken [9] gave general circuit complexity results from which exponential lower bounds for CP follow. To this day it is still unknown whether CP is more powerful than CP*, i.e., whether it produces shorter proofs or not.

Since there is an exponential speed-up of CP over resolution, it would be nice to find an efficient algorithm for finding CP proofs and a question to ask is whether trying to find tree-like CP proofs would be an efficient strategy for finding CP proofs.

Johannsen [18] gave a superpolynomial separation, with a lower bound of the form $\Omega(n^{\log n})$, between tree-like CP and dag-like CP. (This was previously known for CP* from [6].) Here we improve that separation to exponential (Corollary 4.3). This shows that searching for tree-like proofs is also not a good strategy for finding proofs in CP.

The separation between tree-like and dag-like versions of resolution and CP is obtained using the technique of the interpolation method introduced by Krajíček [21]. Closely related ideas appeared previously in the mentioned works that gave lower bounds for fragments of CP [17, 6]. The interpolation method applied on CP translates proofs of certain formulas to monotone real circuits (a generalization of boolean circuits). The translation has two important features. First, it preserves the size; that is, the size of the circuit is similar to the size of the proof from which the circuit is built. Second, if the proof is tree-like, the circuit will be also tree-like, i.e., a formula. So we can prove size lower bounds for tree-like CP proofs by proving size lower bounds for monotone real formulas.

In section 3 we prove that a certain boolean function $\textsc{Gen}_n$ requires exponential size monotone real formulas. This is a consequence of extending the result of Raz and McKenzie [25], proving linear depth lower bounds for monotone boolean circuits to the case of monotone real circuits. We use these circuit complexity lower bounds to obtain proof complexity lower bounds using the interpolation method.

2. Preliminaries and outline of the paper. In this section we introduce the notions we use and our main results. We also discuss the structure of the paper and the dependency among our main results.

2.1. Proof systems. We start by giving a short description of the proof systems studied in this paper. Most proof systems can be used in a tree-like or dag-like fashion. In a tree-like proof any line in the proof can be used only once as a premise. Should the same line be used twice, it must be rederived. A proof system that only produces tree-like proofs is called *tree-like*. Otherwise we will call it *dag-like,* or when nothing is said it is understood that the system is dag-like.

2.1.1. Resolution. Resolution is a refutation proof system for CNF formulas, which are represented as sets of *clauses*, i.e., disjunctions of literals. Clauses that contain the same literals are considered equal. The only inference rule is the resolution rule

$$\frac{C \vee x \qquad D \vee \bar{x}}{C \vee D}\,.$$

That is, from clauses $C \vee x$ and $D \vee \bar{x}$ we get clause $C \vee D$, called the *resolvent*. We say that the variable x is *eliminated* in this resolution step. A resolution refutation of a set Σ of clauses is a derivation of the empty clause from Σ using the resolution rule. Resolution is a sound and complete refutation system, i.e., a set of clauses has a resolution refutation if and only if it is unsatisfiable.

Several refinements of the resolution proof system have been proposed. These refinements reduce the search space by restricting the choice of pairs of clauses to which the resolution rule can be applied. In this paper we consider the following three refinements, all of which are still complete.

1. The *regular* resolution system: Viewing the refutations as graph, in any path from the empty clause to any initial clause, no variable is eliminated twice.

2. The *ordered*[1] resolution system: There exists an ordering of the variables in the formula being refuted, such that if a variable x is eliminated before a variable y on any path from an initial clause to the empty clause, then x is before y in the ordering. As no variable is eliminated twice on any path, ordered resolution is a restriction of regular resolution.
3. The *negative* resolution system: To apply the resolution rule, one of the two clauses must consist of negative literals only.

There is an algorithm (see, e.g., Urquhart [29]) that transforms a tree-like resolution proof into a possibly smaller regular tree-like resolution proof; therefore, tree-like resolution proofs of minimal size are regular. This means that from the point of view of proof complexity, tree-like resolution and tree-like regular resolution are equivalent.

2.1.2. Cutting planes. The CP proof system is a refutation system for CNF formulas, as resolution is. It works with linear inequalities. The initial clauses are transformed into linear inequalities. A generic clause

$$\bigvee_{i=1}^{k} p_{j_i} \vee \bigvee_{i=1}^{m} \neg p_{l_i}$$

is transformed into a linear inequality

$$\sum_{i=1}^{k} p_{j_i} + \sum_{i=1}^{m} (1 - p_{l_i}) \geq 1.$$

The CP rules are basic algebraic manipulations, additions of two inequalities, multiplication of an inequality by a positive integer, and the following division rule:

$$\frac{\sum_{i \in I} a_i x_i \geq k}{\sum_{i \in I} \frac{a_i}{b} x_i \geq \left\lceil \frac{k}{b} \right\rceil},$$

where b is a positive integer that evenly divides all a_i, $i \in I$. A CP refutation of a set E of inequalities is a derivation of $0 \geq 1$ from the inequalities in E and the axioms $x \geq 0$ and $-x \geq -1$ for every variable x, using the CP rules. It can be shown that a set of inequalities has a CP refutation iff it has no $\{0,1\}$-solution. Any assignment satisfying the original clauses is actually a $\{0,1\}$-solution of the corresponding inequalities, provided that we assign the numerical value 1 to True and the value 0 to False. It is easy to translate (see [11]) resolution refutations into CP refutations similar in size to the original resolution refutations. Moreover, if the resolution refutation is tree-like, the resulting CP refutation is also tree-like.

2.2. Monotone real circuits. An important part of this paper is concerned with monotone real circuits, which were introduced by Pudlák [23]. A *monotone real circuit* is a circuit of fan-in 2 computing with real numbers where every gate computes a nondecreasing real function. We require that monotone real circuits output 0 or 1 on every input of 0's and 1's only, so that they are a generalization of monotone boolean circuits. The depth and size of a monotone real circuit are defined as for boolean circuits. A *formula* is a circuit in which every gate has at most fan-out 1, i.e., a tree-like circuit.

[1] In Goerdt's paper [13] and in the preliminary version [5] of this paper, this refinement is called the *Davis–Putnam resolution*. In the meantime, we have learned that it is better known as the *ordered* resolution.

Pudlák [23], Cook and Haken [9], and Fu [12] gave lower bounds on the size of monotone real circuits. Rosenbloom [26] showed that they are strictly more powerful than monotone boolean circuits, since every slice function can be computed by a linear-size, logarithmic-depth monotone real circuit, whereas most slice functions require exponential size general boolean circuits. On the other hand, Jukna [19] gives a general lower bound criterion for monotone real circuits, and uses it to show that certain functions in $P/poly$ require exponential size monotone real circuits, and hence the computing power of monotone real circuits and general boolean circuits is incomparable.

For a monotone boolean function f, we denote by $d_{\mathbb{R}}(f)$ the minimal depth of a monotone real circuit computing f, and by $s_{\mathbb{R}}(f)$ the minimal size of a monotone real formula computing f.

2.3. Deterministic and real communication complexity. The use of communication complexity as a tool to prove depth lower bounds for monotone circuits was introduced by Karchmer and Wigderson [20]. They gave an $\Omega(\log^2 n)$ lower bound on the depth of monotone circuits computing st-connectivity.

Krajíček [22] introduced a notion of *real communication complexity*, generalizing ordinary communication complexity, that is suitable to prove depth lower bounds for monotone real circuits. This was used by Johannsen [18] to extend the depth lower bound for st-connectivity to monotone real circuits.

Raz and McKenzie [25] proved an $\Omega(n^\epsilon)$ lower bound on the depth of monotone circuits computing a certain function $\textsc{Gen}_n$, which, on the other hand, can be computed by monotone circuits of polynomial size. This gives a strong separation of the depth and size complexity of monotone circuits. We extend this lower bound to monotone real circuits, again using the notion of real communication complexity.

2.3.1. Communication complexity. Let $R \subseteq X \times Y \times Z$ be a multifunction, i.e., for every pair $(x, y) \in X \times Y$, there is a $z \in Z$ with $(x, y, z) \in R$. We view such a multifunction as a search problem, i.e., given input $(x, y) \in X \times Y$, the goal is to find a $z \in Z$ such that $(x, y, z) \in R$.

A deterministic communication protocol P over $X \times Y \times Z$ specifies the exchange of information bits between two players, I and II, that receive as inputs, respectively, $x \in X$ and $y \in Y$ and finally agree on a value $P(x, y) \in Z$ such that $(x, y, P(x, y)) \in R$. The *deterministic communication complexity* of R, $CC(R)$, is the number of bits communicated between players I and II in an optimal protocol for R.

2.3.2. Real communication complexity. A real communication protocol over $X \times Y \times Z$ is executed by two players I and II who exchange information by simultaneously playing real numbers and then comparing them according to the natural order of $\mathbb{R}$. This generalizes ordinary deterministic communication protocols in the following way: in order to communicate a bit, the sender plays this bit, while the receiver plays a constant between 0 and 1, so that he can determine the value of the bit from the outcome of the comparison.

Formally, such a protocol P is specified by a binary tree, where each internal node v is labeled by two functions $f_v^I : X \to \mathbb{R}$, giving player I's move, and $f_v^{II} : Y \to \mathbb{R}$, giving player II's move, and each leaf is labeled by an element $z \in Z$. On input $(x, y) \in X \times Y$, the players construct a path through the tree according to the following rule:

> At node v, if $f_v^I(x) > f_v^{II}(y)$, then the next node is the left son of v and otherwise the right son of v.

The value $P(x,y)$ computed by P on input (x,y) is the label of the leaf reached by this path.

A real communication protocol P solves a search problem $R \subseteq X \times Y \times Z$ if for every $(x,y) \in X \times Y$, $(x,y,P(x,y)) \in R$ holds. The *real communication complexity* $CC_{\mathbb{R}}(R)$ of a search problem R is the minimal depth of a real communication protocol that solves R.

For a natural number n, let $[n]$ denote the set $\{1,\dots,n\}$. Let $f : \{0,1\}^n \to \{0,1\}$ be a monotone boolean function, let $X := f^{-1}(1)$ and $Y := f^{-1}(0)$, and let the multifunction $R_f \subseteq X \times Y \times [n]$ be defined by

$$(x,y,i) \in R_f \quad \text{iff} \quad x_i = 1 \text{ and } y_i = 0.$$

The Karchmer–Wigderson game for f is defined as follows. Player I receives an input $x \in X$ and player II an input $y \in Y$. They have to agree on a position $i \in [n]$ such that $(x,y,i) \in R_f$. The Karchmer–Wigderson game for a monotone boolean function f is also denoted by R_f. As happens with monotone boolean functions and communication complexity, there is a relation between the real communication complexity of R_f and the depth of monotone real circuits (and the size of a monotone real formulas) computing f.

LEMMA 2.1 (see Krajíček [22]). *Let f be a monotone boolean function. Then*

1. $CC_{\mathbb{R}}(R_f) \le d_{\mathbb{R}}(f)$;
2. $CC_{\mathbb{R}}(R_f) \le \log_{3/2} s_{\mathbb{R}}(f)$.

For a proof see [22] or [18]. Notice that by Lemma 2.1 a linear lower bound for the real communication complexity of R_f gives an exponential lower bound for the size of the smallest monotone real formula computing f.

2.4. DART games and structured protocols. Raz and McKenzie [25] introduced a special kind of communication games, called DART games, and a special class of communication protocols, the *structured protocols*, for solving them.

For $m,k \in \mathbb{N}$, $\mathrm{DART}(m,k)$ is the set of communication games specified by a relation $R \subseteq X \times Y \times Z$ such that the following hold.

- $X = [m]^k$; i.e., the inputs for player I are k-tuples of elements $x_i \in [m]$.
- $Y = (\{0,1\}^m)^k$; i.e., the inputs for player II are k-tuples of binary colorings y_i of $[m]$.
- For all $i = 1,\dots,k$ let $e_i = y_i(x_i) \in \{0,1\}$ (i.e., the x_i-th bit in the m-bits string y_i). The relation $R \subseteq X \times Y \times Z$ defining the game only depends on $e_1,\dots,e_k$ and z, i.e., we can describe $R(x,y,z)$ as $R((e_1,\dots,e_k),z)$.
- $R((e_1,\dots,e_k),z)$ can be expressed as a disjunctive normal form (DNF)-search-problem, i.e., there exists a DNF-tautology F_R defined over the variables $e_1,\dots,e_k$ such that Z is the set of terms of F_R, and $R((e_1,\dots,e_k),z)$ holds iff the term z is satisfied by the assignment $(e_1,\dots,e_k)$.

A *structured protocol* for a DART game is a communication protocol for solving the search problem R, where player I gets input $x \in X$, player II gets input $y \in Y$, and in each round, player I reveals the value x_i for some i, and II replies with $y_i(x_i)$. The structured communication complexity of $R \in \mathrm{DART}(m,k)$, denoted by $SC(R)$, is the minimal number of rounds in a structured protocol solving R. In [25] it was proved that $CC(R) = SC(R) \cdot \Omega(\log m)$ for $R \in \mathrm{DART}(m,k)$. We generalize this result to real communication complexity, proving

$$CC_{\mathbb{R}}(R) = SC(R) \cdot \Omega(\log m).$$

Observe that at each structured round the two players transmit $\lceil \log m \rceil + 1$ bits. The first player transmits a number in $[m]$ and the second answers with a bit. Since both players know the structure of the protocol for the game, at each round they both know the coordinate i of the inputs they are talking about and they have no need to transmit it. So for a DART game R we have $CC_{\mathbb{R}}(R) \leq SC(R) \cdot \Omega(\log m)$.

Proving the opposite inequality, which is one of our main results, is much harder. In Theorem 3.4 we show that for every relation $R \in \text{DART}(m,k)$, where $m \geq k^{14}$, $CC_{\mathbb{R}}(R) \geq SC(R) \cdot \Omega(\log m)$.

2.5. The interpolation method. The separations between tree-like CP (respectively, tree-like resolution) and CP (resolution) are among our main results about proof complexity. The lower bound part of the separation is obtained employing the following theorem which relates the size of CP refutations with size of monotone real circuits.

THEOREM 2.2 (see Pudlák [23]). *Let $\vec{p}, \vec{q}, \vec{r}$ be disjoint vectors of variables, and let $A(\vec{p}, \vec{q})$ and $B(\vec{p}, \vec{r})$ be sets of inequalities in the indicated variables such that the variables $\vec{p}$ either have only nonnegative coefficients in $A(\vec{p}, \vec{q})$ or have only nonpositive coefficients in $B(\vec{p}, \vec{r})$.*

Suppose there is a CP refutation P of $A(\vec{p}, \vec{q}) \cup B(\vec{p}, \vec{r})$. Then there is a monotone real circuit $C(\vec{p})$ of size $O(|P|)$ such that for any vector $\vec{a} \in \{0,1\}^{|\vec{p}|}$

$$\begin{aligned} C(\vec{a}) = 0 \quad &\rightarrow \quad A(\vec{a}, \vec{q}) \text{ is unsatisfiable,} \\ C(\vec{a}) = 1 \quad &\rightarrow \quad B(\vec{a}, \vec{r}) \text{ is unsatisfiable.} \end{aligned}$$

Furthermore, if P is tree-like, then $C(\vec{p})$ is a monotone real formula.

The fact that the interpolant $C(\vec{p})$ is a monotone real formula if the refutation is tree-like is not stated explicitly in [23], but it can be checked easily by analyzing the original proof of Theorem 2.2 in [23].

We use this theorem to get lower bounds for CP refutations from lower bounds for monotone real formulas. Recall that a *minterm* (respectively, a *maxterm*) of a boolean function $f : \{0,1\}^n \to \{0,1\}$ is a set of inputs $x \in \{0,1\}^n$ such that $f(x) = 1$ (respectively, $f(x) = 0$) and for each $y \in \{0,1\}^n$ obtained from x by changing a bit from 1 to 0 (respectively, by changing a bit from 0 to 1) it holds that $f(y) = 0$ (respectively, $f(y) = 1$).

For a certain boolean function f we will apply Theorem 2.2 to a CNF formula $A(\vec{p}, \vec{q}) \cup B(\vec{p}, \vec{r})$ such that $A(\vec{p}, \vec{q})$ will encode that $\vec{p}$ is a minterm of f and $B(\vec{p}, \vec{r})$ will encode that $\vec{p}$ is maxterm of f. Clearly the formula is unsatisfiable. Using the interpolation theorem, from any tree-like CP refutation of $A(\vec{p}, \vec{q}) \cup B(\vec{p}, \vec{r})$ we will get an interpolant which is a monotone real formula computing f. Therefore if we prove exponential lower bounds for the size of the tree-like monotone real circuits computing f, we immediately obtain an exponential lower bound for tree-like CP refutations for $A(\vec{p}, \vec{q}) \cup B(\vec{p}, \vec{r})$. The same result also holds for tree-like resolution.

To get the separation results we need a monotone boolean function with some nice properties, namely,

1. exponential lower bounds for monotone real formulas computing the function, and
2. the corresponding $A(\vec{p}, \vec{q}) \cup B(\vec{p}, \vec{r})$ formula must have polynomial-size resolution (and, therefore, also CP) refutations.

The chosen monotone boolean function f is the function $\text{GEN}_n : \{0,1\}^{n^3} \to \{0,1\}$ considered by Raz and McKenzie [25]. The input bits are called $t_{a,b,c}$ for $a, b, c \in [n]$.

The function is defined as follows: $\text{GEN}_n(t_{111}\cdots t_{nnn}) = 1$ iff $\vdash n$, where for $c \in [n]$, $\vdash c$ (meaning c *is generated*) is defined recursively by

$$\vdash c \quad \text{iff} \quad c = 1 \text{ or there are } a, b \le n \text{ with } \vdash a\ , \vdash b, \text{ and } t_{a,b,c} = 1\ .$$

From now on we will write $a, b \vdash c$ for $t_{a,b,c} = 1$.

To get the exponential separation the task to be done is as follows.

1. Prove exponential lower bounds for the size of monotone real formulas computing GEN_n.
2. Find CNF formulas $A(\vec{p}, \vec{q})$ and $B(\vec{p}, \vec{r})$ expressing, respectively, a minterm and a maxterm of GEN_n.
3. Show polynomial size resolution (and CP) refutations for $A(\vec{p}, \vec{q}) \cup B(\vec{p}, \vec{r})$.

In section 3 we will show, among other things, that $CC_{\mathbb{R}}(R_{\text{GEN}_n}) \ge \Omega(n^\epsilon)$ for some $\epsilon > 0$. From this, it follows by part 2 of Lemma 2.1 that $s_{\mathbb{R}}(\text{GEN}_n) \ge 2^{\Omega(n^\epsilon)}$; thus task 1 is achieved. Tasks 2 and 3 will be developed in section 4.

3. Lower bounds for real communication complexity. In this section we prove an $\Omega(n^\epsilon)$ lower bound for the real communication complexity of the Karchmer–Wigderson game associated to GEN_n, denoted by R_{GEN_n}.

THEOREM 3.1. *For some $\epsilon > 0$ and sufficiently large n*

$$CC_{\mathbb{R}}(R_{\text{GEN}_n}) \ge \Omega(n^\epsilon).$$

To prove Theorem 3.1 we define a DART game $\text{PYRGEN}(m, d)$ in section 3.1 related to the GEN_n function. This game is used with parameters $m = d^{28}$ and $n = \binom{d+1}{2}m + 2$, so that $d \approx n^{1/30}$. Then we will prove the following results from which Theorem 3.1 directly follows:

$$\begin{aligned} SC(\text{PYRGEN}(m,d)) &\ge d && \text{(Lemma 3.2)},\\ CC_{\mathbb{R}}(\text{PYRGEN}(m,d)) &\ge SC(\text{PYRGEN}(m,d))\,\Omega(\log m) && \text{(Theorem 3.4)},\\ CC_{\mathbb{R}}(R_{\text{GEN}_n}) &\ge CC_{\mathbb{R}}(\text{PYRGEN}(m,d)) && \text{(Lemma 3.3)}. \end{aligned}$$

Lemma 3.2 is proved in [25]; therefore, we omit its proof. Theorem 3.4 is proved in section 3.2 for any DART game R. Lemma 3.3 is proved in section 3.1. In section 3.3 we deduce some lower bounds for monotone real circuits from these results.

3.1. The pyramidal generation game. For $d \in \mathbb{N}$, let

$$Pyr_d := \{\, (i,j)\ :\ 1 \le j \le i \le d \,\}\,.$$

Following [25], a communication game in $DART(m, \binom{d+1}{2})$ called $\text{PYRGEN}(m, d)$ is defined as follows. We regard the indices as elements of Pyr_d, so that the inputs for the two players I and II in the $\text{PYRGEN}(m, d)$ game are, respectively, sequences of elements $x_{i,j} \in [m]$ and $y_{i,j} \in \{0,1\}^m$ with $(i,j) \in Pyr_d$, and we picture these as laid out in a pyramidal form with $(1,1)$ at the top and (d,j), $1 \le j \le d$, at the bottom. The goal of the game is to find either an element colored 0 at the top of the pyramid, or an element colored 1 at the bottom of the pyramid, or an element colored 1 with the two elements below it colored 0. That is, we have to find indices (i,j) such that one of the following holds:

1. $i = j = 1$ and $y_{1,1}(x_{1,1}) = 0$, or
2. $y_{i,j}(x_{i,j}) = 1$ and $y_{i+1,j}(x_{i+1,j}) = 0$ and $y_{i+1,j+1}(x_{i+1,j+1}) = 0$, or
3. $i = d$ and $y_{d,j}(x_{d,j}) = 1$.

Observe that, setting $e_{i,j} = y_{i,j}(x_{i,j})$ for $1 \leq j \leq i \leq d$, this search problem can be defined as a DNF-search-problem given by the following DNF-tautology:

$$\bar{e}_{1,1} \vee \bigvee_{1 \leq j \leq i \leq d-1} (e_{i,j} \wedge \bar{e}_{i+1,j} \wedge \bar{e}_{i+1,j+1}) \vee \bigvee_{1 \leq j \leq d} e_{d,j}.$$

Therefore, PyrGen(m,d) is a game in DART$(m, \binom{d+1}{2})$.

A lower bound on the structured communication complexity of PyrGen(m,d) was proved in [25].

Lemma 3.2 (see Raz and McKenzie [25]). *$SC($PyrGen$(m,d)) \geq d$.*

The following reduction shows that the real communication complexity of the game PyrGen(m,d) is bounded by the real communication complexity of the Karchmer–Wigderson game for Gen$_n$ (denoted by R_{Gen_n}) for a suitable n. The proof is taken from [25]. It is included because it can help the reader to understand other parts of this paper.

Lemma 3.3. *Let $d, m \in \mathbb{N}$ and let $n := m \cdot \binom{d+1}{2} + 2$. Then*

$$CC_{\mathbb{R}}(\text{PyrGen}(m,d)) \leq CC_{\mathbb{R}}(R_{\text{Gen}_n}).$$

Proof. We prove that any protocol solving the Karchmer–Wigderson game for Gen$_n$ can be used to solve the PyrGen(m,d) game. Recall that PyrGen(m,d) is a DART$(m, \binom{d+1}{2})$ game, so the two players I and II receive inputs, respectively, of the form $(x_{1,1}, \dots, x_{d,d})$, where $x_{i,j} \in [m]$ for all $(i,j) \in Pyr_d$ and $(y_{1,1}, \dots, y_{d,d})$, where $y_{i,j} \in \{0,1\}^m$ for all $(i,j) \in Pyr_d$.

From their respective inputs for the PyrGen(m,d) game, players I and II compute, respectively, a minterm $t^x_{a,b,c}$ and a maxterm $t^y_{a,b,c}$, for Gen$_n$, and then they play the Karchmer–Wigderson game applying the protocol P.

As in [25] we consider fixed the element 1 as a bottom generator and the element n as the element we want to generate. We interpret the remaining $n-2 = \binom{d+1}{2}m$ elements between 2 and $n-1$ as triples (i,j,k), where $(i,j) \in Pyr_d$ and $k \in [m]$.

Now player I computes from his input $(x_{1,1}, \dots, x_{d,d})$ an input $t^x_{a,b,c}$ for Gen$_n$ such that Gen$_n(t^x_{a,b,c}) = 1$ by setting the following (recall that $a, b \vdash c$ means $t_{a,b,c} = 1$):

$$\begin{aligned} &1, 1 \vdash g_{d,j} && \text{for } 1 \leq j \leq d, \\ &g_{1,1}, g_{1,1} \vdash n, && \\ &g_{i+1,j}, g_{i+1,j+1} \vdash g_{i,j} && \text{for } (i,j) \in Pyr_{d-1}, \end{aligned}$$

where $g_{i,j} := (i,j,x_{i,j}) \in \{2, \dots, n-1\}$ and all the other bits $t^x_{a,b,c} = 0$. This completely determines $t^x_{a,b,c}$, and obviously Gen$_n(t^x_{a,b,c}) = 1$ since we have forced a generation of n (in a pyramidal form).

Likewise, player II computes from his input $(y_{1,1}, \dots, y_{d,d})$ a coloring col of the elements from $[n]$ by setting $col(1) = 0$, $col(n) = 1$, and $col((i,j,k)) = y_{i,j}(k)$ (the kth bit of $y_{(i,j)}$). From this coloring, he computes an input $t^y_{a,b,c}$ by setting $t^y_{a,b,c} = 1$ iff it is not the case that $col(c) = 1$ and $col(a) = col(b) = 0$. Obviously, Gen$_n(t^y_{a,b,c}) = 0$.

Running the protocol for the Karchmer–Wigderson game for Gen$_n$ now yields a triple (a,b,c) such that $t^x_{a,b,c} = 1$ and $t^y_{a,b,c} = 0$. By definition of t^y, this means that $col(a) = col(b) = 0$ and $col(c) = 1$, and by definition of t^x one of the following cases must hold.

- $a = b = 1$ and $c = g_{d,j}$ for some $j \leq d$. By definition of col, $y_{d,j}(x_{d,j}) = 1$.

- $c = n$ and $a = b = g_{1,1}$. In this case, $y_{1,1}(x_{1,1}) = 0$.
- $a = g_{i+1,j}$, $b = g_{i+1,j+1}$, and $c = g_{i,j}$. Then we have $y_{i,j}(x_{i,j}) = 1$, and $y_{i+1,j}(x_{i+1,j}) = y_{i+1,j+1}(x_{i+1,j+1}) = 0$.

In either case, the players have solved PyrGen(m, d) without any additional communication. □

3.2. Relation between structured complexity and real communication complexity. We prove here the following general theorem for DART games.

Theorem 3.4. *Let $m, k \in \mathbb{N}$. For every relation $R \in \mathrm{DART}(m, k)$, where $m \geq k^{14}$,*

$$CC_{\mathbb{R}}(R) \geq SC(R) \cdot \Omega(\log m) .$$

We first need some combinatorial notions from [25] and some lemmas. Let $A \subseteq [m]^k$ and $1 \leq j \leq k$. For $x \in [m]^{k-1}$, let $\deg_j(x, A)$ be the number of $\xi \in [m]$ such that $(x_1, \ldots, x_{j-1}, \xi, x_j, \ldots, x_{k-1}) \in A$. Then we define

$$\begin{aligned} A[j] &:= \left\{ x \in [m]^{k-1} \ : \ \deg_j(x, A) > 0 \right\}, \\ \mathrm{AVDEG}_j(A) &:= \frac{|A|}{|A[j]|}, \\ \mathrm{MINDEG}_j(A) &:= \min_{x \in A[j]} \deg_j(x, A), \\ \mathrm{Thickness}(A) &:= \min_{1 \leq j \leq k} \mathrm{MINDEG}_j(A) . \end{aligned}$$

The following lemmas about these notions were proved in [25].

Lemma 3.5 (see [25]). *For every $A' \subseteq A$ and $1 \leq j \leq k$,*

$$\mathrm{AVDEG}_j(A') \geq \frac{|\mathrm{A}'|}{|\mathrm{A}|} \mathrm{AVDEG}_j(A), \tag{3.1}$$

$$\mathrm{Thickness}(A[j]) \geq \mathrm{Thickness}(A). \tag{3.2}$$

Lemma 3.6 (see [25]). *Let $0 < \delta < 1$ be given. If for every $1 \leq j \leq k$, $\mathrm{AVDEG}_j(A) \geq \delta \mathrm{m}$, then for every $\alpha > 0$ there is $A' \subseteq A$ with $|A'| \geq (1 - \alpha)|A|$ and*

$$\mathrm{Thickness}(A') \geq \frac{\alpha \delta m}{k} .$$

In particular, setting $\alpha = \frac{1}{2}$ and $\delta = 4m^{-\frac{1}{14}}$, we get the following corollary.

Corollary 3.7. *If $m \geq k^{14}$ and for every $1 \leq j \leq k$, $\mathrm{AVDEG}_j(A) \geq 4\mathrm{m}^{\frac{13}{14}}$, then there is $A' \subseteq A$ with $|A'| \geq \frac{1}{2}|A|$ and $\mathrm{Thickness}(A) \geq m^{\frac{11}{14}}$.*

For a relation $R \in \mathrm{DART}(m, k)$, $A \subseteq X$ and $B \subseteq Y$, let $CC_{\mathbb{R}}(R, A, B)$ be the real communication complexity of R restricted to $A \times B$.

Definition 3.8 ((α, β, ℓ)-game). *Let $m \in \mathbb{N}$, $m \geq k^{14}$. Let $A \subseteq X$ and $B \subseteq Y$. A triple (R, A, B) is called an (α, β, ℓ)-game if the following conditions hold:*

1. *$R \in \mathrm{DART}(m, k)$,*
2. *$SC(R) \geq \ell$,*
3. *$|A| \geq 2^{-\alpha}|X|$ and $|B| \geq 2^{-\beta}|Y|$,*
4. *$\mathrm{Thickness}(A) \geq m^{\frac{11}{14}}$.*

The following lemma and its proof are slightly different from the corresponding lemma in [25], because we use the strong notion of real communication complexity

where [25] uses ordinary communication complexity. The modification we apply is analogous to that introduced by Johannsen [18] to improve the result of Karchmer and Wigderson [20] to the case of real communication complexity. This modification will affect the proof of the first point of the next lemma. We include a proof of the second part for completeness.

LEMMA 3.9. *For every $\alpha, \ell \geq 0$ and $0 \leq \beta \leq m^{\frac{1}{7}}$, $m \geq 1000^{14}$, and every (α, β, ℓ)-game (R, A, B),*

1. *if for every $1 \leq j \leq k$, $\mathrm{AVDEG}_j(A) \geq 8\mathrm{m}^{\frac{13}{14}}$, then there is an $(\alpha+2, \beta+1, \ell)$-game (R', A', B') with*

$$CC_{\mathbb{R}}(R', A', B') \leq CC_{\mathbb{R}}(R, A, B) - 1;$$

2. *if $\ell \geq 1$ and for some $1 \leq j \leq k$, $\mathrm{AVDEG}_j(A) < 8\mathrm{m}^{\frac{13}{14}}$, then there is an $(\alpha + 3 - \frac{\log m}{14}, \beta + 1, \ell - 1)$-game (R', A', B') with*

$$CC_{\mathbb{R}}(R', A', B') \leq CC_{\mathbb{R}}(R, A, B) \, .$$

Proof (proof of Lemma 3.9 (part 1)). Let (R, A, B) be an (α, β, ℓ)-game. First we show that $CC_{\mathbb{R}}(R, A, B) \neq 0$. Assume by contradiction that $CC_{\mathbb{R}}(R, A, B) = 0$. Then the players have no need to transmit information to solve R. This means that the answer to the game is implicit in the domain $A \times B$ and, therefore, by requirement (4) of DART games there is a term in the DNF-tautology F_R defining R that is satisfied for every $(x, y) \in A \times B$. Therefore, there is at least a coordinate j, $1 \leq j \leq k$, such that $y_j(x_j)$ is constant (i.e., is always 0 or always 1). If γ denotes the number of possible different values of x_j in elements of A, then this implies that $|B| \leq 2^{mk-\gamma}$. On the other hand, $|B| \geq 2^{mk-\beta}$, and hence it follows that $\beta \geq \gamma$, which is a contradiction since $\beta \leq m^{\frac{1}{7}}$, whereas $\mathrm{AVDEG}_j(A) \geq 8\mathrm{m}^{\frac{13}{14}}$ implies $\gamma \geq 8m^{\frac{13}{14}}$.

Let an optimal real communication protocol solving R restricted to $A \times B$ be given. For $a \in A$ and $b \in B$, let ρ_a and σ_b be the real numbers played by I and II in the first round on input a and b, respectively. Without loss of generality we can assume that these are $|A| + |B|$ pairwise distinct real numbers.

Now consider a $\{0,1\}$-matrix of size $|A| \times |B|$ with columns indexed by the ρ_a and rows indexed by the σ_b, both in increasing order, and where the entry in position (ρ_a, σ_b) is 1 if $\rho_a > \sigma_b$ and 0 if $\rho_a \leq \sigma_b$. Thus this entry determines the outcome of the first round, when these numbers are played. It is now obvious that either the upper right quadrant or the lower left quadrant must form a monochromatic rectangle.

Hence there are $A^\circ \subseteq A$ and $B' \subseteq B$ with $|A^\circ| \geq \frac{1}{2}|A|$ and $|B'| \geq \frac{1}{2}|B|$ such that R restricted to $A^\circ \times B'$ can be solved by a protocol with one round fewer than the original protocol. This means that $CC_{\mathbb{R}}(R, A^\circ, B') \leq CC_{\mathbb{R}}(R, A, B) - 1$. By (3.1) of Lemma 3.5, $\mathrm{AVDEG}_j(A^\circ) \geq 4\mathrm{m}^{\frac{13}{14}}$ for every $1 \leq j \leq k$; hence by Corollary 3.7 there is $A' \subseteq A^\circ$ with $|A'| \geq \frac{1}{2}|A^\circ| \geq \frac{1}{4}|A|$ and $\mathrm{Thickness}(A') \geq m^{\frac{11}{14}}$. Thus (R, A', B') is an $(\alpha + 2, \beta + 1, \ell)$-game; moreover, since $A' \subseteq A^\circ$, we have that $CC_R(R, A', B') \leq CC_R(R, A^\circ, B')$, from which the lemma follows.

(Part 2). We proceed like in the proof of the corresponding lemma of [25], with the numbers slightly adjusted. Assume without loss of generality that k is the coordinate for which $\mathrm{AVDEG}_k(A) < 8\mathrm{m}^{\frac{13}{14}}$. Let R_0 and R_1 be the restrictions of R in which the kth coordinate $e_k = y_k(x_k)$ is fixed to 0 and 1, respectively. Obviously, R_0 and R_1 are $\mathrm{DART}(m, k-1)$ relations, and therefore at least one of $SC(R_0)$ and $SC(R_1)$ is at least $\ell - 1$. Assume without loss of generality that $SC(R_0) \geq \ell - 1$. We will prove that there are two sets $A' \subseteq [m]^{k-1}$ and $B' \subseteq (\{0,1\}^m)^{k-1}$ such that the following

properties hold:

$$|A'| \geq \frac{m^{k-1}}{2^{\alpha+3-\frac{\log m}{14}}}, \tag{3.3}$$

$$|B'| \geq \frac{2^{m(k-1)}}{2^{\beta+1}}, \tag{3.4}$$

$$\text{Thickness}(A') \geq m^{\frac{11}{14}}, \tag{3.5}$$

$$CC_{\mathbb{R}}(R_0, A', B') \leq CC_{\mathbb{R}}(R, A, B). \tag{3.6}$$

This means that there is an $(\alpha + 3 - \frac{\log m}{14}, \beta + 1, \ell - 1)$-game (R_0, A', B') such that $CC_{\mathbb{R}}(R_0, A', B') \leq CC_{\mathbb{R}}(R, A, B)$ and this proves part 2 of Lemma 3.9.

Given any set $U \subset [m]$, consider the sets $A_U \subseteq [m]^{k-1}$ and $B_U \subseteq (\{0,1\}^m)^{k-1}$ associated to the set U by the following definition of [25]:

- $(x_1, \ldots x_{k-1}) \in A_U$ iff there is an $u \in U$ such that $(x_1, \ldots x_{k-1}, u) \in A$;
- $(y_1, \ldots y_{k-1}) \in B_U$ iff there is a $w \in \{0,1\}^m$ such that $w(u) = 0$ for all $u \in U$ and $(y_1, \ldots y_{k-1}, w) \in B$.

The following two claims can be proved exactly as the corresponding claims of [25] and we omit their proof.

CLAIM 3.10. *For a random set U of size $m^{\frac{5}{14}}$, with $m \geq 1000^{14}$, we have that*

$$Prob_U\big[A_U = A[k]\big] \geq \frac{3}{4}\,.$$

CLAIM 3.11. *For a random set U of size $m^{\frac{5}{14}}$, with $m \geq 1000^{14}$, we have that*

$$Prob_U\left[|B_U| \geq \frac{|B|}{2^{m+1}}\right] \geq \frac{3}{4}\,.$$

Moreover, it is immediate to see that the same reduction used in Claim 6.3 of [25] also works for the case of real communication complexity. Therefore, we get the following claim.

CLAIM 3.12. *For every set $U \subset [m]$,*

$$CC_{\mathbb{R}}(R_0, A_U, B_U) \leq CC_{\mathbb{R}}(R, A, B)\,.$$

Take a random set U which, with probability greater than $\frac{1}{2}$, satisfies both the properties of Claim 3.10 and Claim 3.11, and define $A' := A_U$ and $B' := B_U$. This means that with probability at least $\frac{1}{2}$ both $A' = A[k]$ and $|B'| \geq \frac{|B|}{2^{m+1}}$ hold.

Recall that $\frac{|A|}{|A'|} = \frac{|A|}{|A[k]|} = \text{AVDEG}_k(A)$ and that, by hypothesis on part 2 of the lemma $|\text{AVDEG}_k(A)| \leq 8\text{m}^{\frac{13}{14}}$. Therefore, we have that

$$|A'| \geq \frac{|A|}{8m^{\frac{13}{14}}} \geq \frac{m^k}{2^\alpha 8m^{\frac{13}{14}}} = \frac{m^{k-1}}{2^{\alpha+3-\frac{\log m}{14}}}\,.$$

This proves (3.3). For (3.4) observe that by Claim 3.11 we have

$$|B'| \geq \frac{|B|}{2^{m+1}} \geq \frac{2^{mk}}{2^\beta 2^{m+1}} = \frac{2^{m(k-1)}}{2^{\beta+1}}\,.$$

The property (3.5) follows directly from Lemma 3.5 (3.2), and finally (3.6) follows from Claim 3.12. □

3.2.1. Proof of Theorem 3.4.

Proof. Let $k \in \mathbb{N}$, $k \geq 1000$. We prove that for any $\alpha, \beta, \ell, m \geq 0$, with $\beta \leq m^{1/7}$, $\ell \geq 1$, and $m \geq k^{14}$, every (α, β, ℓ)-game (R, A, B) is such that

$$CC_{\mathbb{R}}(R, A, B) \geq \ell \cdot \left(\frac{\log m}{42} - \frac{4}{3}\right) - \frac{\alpha + \beta}{3} . \tag{3.7}$$

Observe that by the definition of an (α, β, ℓ)-game, when $\alpha = \beta = 0$ we have that $A = X$ and $B = Y$. Therefore, $CC_{\mathbb{R}}(R, A, B) = CC_{\mathbb{R}}(R)$. Moreover, the right side of (3.7) reduces to $\ell \cdot \Omega(\log m)$. Since by the same definition $\ell \leq SC(R)$ for $\alpha = \beta = 0$ we get the claim of the theorem:

$$CC_{\mathbb{R}}(R) \geq SC(R) \cdot \Omega(\log m).$$

To prove (3.7), we proceed by induction on $\ell \geq 1$ and $\beta \leq m^{1/7}$. In the base case $\ell < 1$ (that is, $\ell = 0$) and $\beta > m^{\frac{1}{7}}$, the inequality (3.7) is trivial, since the right-hand side gets negative for large m. In the inductive step consider (R, A, B) as an (α, β, ℓ)-game, and assume that (3.7) holds for all (α', β', ℓ')-games with $\ell' \leq \ell$ and $\beta' > \beta$. For the sake of contradiction, suppose that $CC_{\mathbb{R}}(R, A, B) < \ell \cdot (\frac{\log m}{42} - \frac{4}{3}) - \frac{\alpha+\beta}{3}$. Then either for every $1 \leq j \leq k$, $\mathrm{AVDEG}_j(A) \geq 8\mathrm{m}^{\frac{13}{14}}$, and Lemma 3.9 gives an $(\alpha + 2, \beta + 1, \ell)$-game (R', A', B') with

$$\begin{aligned} CC_{\mathbb{R}}(R', A', B') &\leq CC_{\mathbb{R}}(R, A, B) - 1 \\ &< \ell \cdot \left(\frac{\log m}{42} - \frac{4}{3}\right) - \frac{(\alpha + 2) + (\beta + 1)}{3} , \end{aligned}$$

or for some $1 \leq j \leq k$, $\mathrm{AVDEG}_j(A) < 8\mathrm{m}^{\frac{13}{14}}$, and Lemma 3.9 gives an $(\alpha + 3 - \frac{\log m}{14}, \beta + 1, \ell - 1)$-game (R', A', B') with

$$\begin{aligned} CC_{\mathbb{R}}(R', A', B') &< \ell \cdot \left(\frac{\log m}{42} - \frac{4}{3}\right) - \frac{\alpha + \beta}{3} \\ &= (\ell - 1) \cdot \left(\frac{\log m}{42} - \frac{4}{3}\right) - \frac{(\alpha + 3 - \frac{\log m}{14}) + (\beta + 1)}{3} , \end{aligned}$$

both contradicting the assumption. ∎

3.3. Consequences for monotone real circuits. As a first corollary to Theorem 3.4, we observe that for DART games, real communication protocols are no more powerful than deterministic communication protocols.

COROLLARY 3.13. *Let $m, k \in \mathbb{N}$. For $R \in \mathrm{DART}(m, k)$ with $m \geq k^{14}$,*

$$CC_{\mathbb{R}}(R) = \Theta(CC(R)) .$$

Proof. $CC(R) \geq CC_{\mathbb{R}}(R) \geq SC(R) \cdot \Omega(\log m) \geq \Omega(CC(R))$. ∎

From Theorem 3.1 we obtain consequences for monotone real circuits analogous to those obtained in [25] for monotone boolean circuits. An immediate consequence of Theorem 3.1 and Lemma 2.1 is the following theorem.

THEOREM 3.14. *Any tree-like monotone real circuit computing the boolean function GEN_n must have size $2^{\Omega(n^\epsilon)}$ for some $\epsilon > 0$.*

DEFINITION 3.15 (pyramidal generation). *Let $\vec{t}$ be an input to* GEN_n. *We say that n is generated in a depth d pyramidal fashion by $\vec{t}$ if there is a mapping $m : Pyr_d \to [n]$ such that the following hold (recall that $a, b \vdash c$ means $t_{a,b,c} = 1$):*

$$\begin{aligned} &1, 1 \vdash m(d, j) && \text{for every } j \le d, \\ &m(i+1, j), m(i+1, j+1) \vdash m(i, j) && \text{for every } (i, j) \in Pyr_{d-1}, \\ &m(1, 1), m(1, 1) \vdash n. \end{aligned}$$

We can obtain an analogue of Theorem 3.14 also for the simpler case in which the generation is restricted to be only in a pyramidal form.

COROLLARY 3.16. *Every monotone real formula that outputs* 1 *on every input to* GEN_n *for which n is generated in a depth d pyramidal fashion, and outputs* 0 *on all inputs where* GEN_n *is* 0, *has to be of size $\Omega(2^{n^\epsilon})$ for some $\epsilon > 0$.*

Proof. To simplify, let $Pyrgen_n$ be any monotone boolean function that outputs 1 on every input to GEN_n for which n is generated in a depth d pyramidal fashion, and outputs 0 on all inputs where GEN_n is 0. Note that there are many such functions, since the output is not specified in the case where n can be generated, but not in a depth d pyramidal fashion. Observe that in Lemma 3.3, player I builds from his input an input for GEN_n which enforces a depth d pyramidal generation. So the proof of Lemma 3.4 also shows that $CC_{\mathbb{R}}(\text{PYRGEN}(m, d)) \le CC_{\mathbb{R}}(R_{Pyrgen_n})$. Lemma 3.2 and Theorem 3.4 then imply that $CC_{\mathbb{R}}(R_{Pyrgen_n}) \ge \Omega(n^\epsilon)$ for some $\epsilon > 0$. Finally, Lemma 2.1 gives the statement of the corollary. ∎

The other consequences drawn from Theorem 3.4 and Lemma 3.2 in [25] apply to monotone real circuits as well, e.g., we just state without proof the following result.

THEOREM 3.17. *There are constants $0 < \epsilon, \gamma < 1$ such that for every function $d(n) \le n^\epsilon$, there is a family of monotone functions $f_n : \{0,1\}^n \to \{0,1\}$ that can be computed by monotone boolean circuits of size $n^{O(1)}$ and depth $d(n)$, but cannot be computed by monotone real circuits of depth less than $\gamma \cdot d(n)$.*

The method also gives a simpler proof of the lower bounds in [18] in the same way that [25] simplifies the lower bound of [20].

4. Separation between tree-like and dag-like versions of resolution and cutting planes. We will define an unsatisfiable CNF formula $Gen(\vec{p}, \vec{q}) \wedge Col(\vec{p}, \vec{r})$ that fulfills the assumptions of Theorem 2.2, so any CP refutation of it can be transformed into a monotone real circuit, and any tree-like CP refutation into a monotone real formula. This circuit (or formula) is similar in size to the original CP refutation. We will show that it computes a boolean function related to GEN_n: It outputs 1 if n is generated in a pyramidal way, so the exponential size lower bound in Corollary 3.16 implies an exponential size lower bound for tree-like CP refutations of $Gen(\vec{p}, \vec{q}) \wedge Col(\vec{p}, \vec{r})$. Besides, we give a polynomial size resolution refutation of $Gen(\vec{p}, \vec{q}) \wedge Col(\vec{p}, \vec{r})$. As CP polynomially simulates resolution, we get the separation between tree-like CP and CP; in fact, we also get a separation of tree-like resolution from resolution.

Let n and d be natural numbers whose values are to be fixed. Recall that the set Pyr_d is $\{\, (i, j) \,:\, 1 \le j \le i \le d \,\}$. The vector $\vec{p}$, that is, the variables $p_{a,b,c}$ for $a, b, c \in [n]$, represent the input to GEN_n.

The set of clauses $Gen(\vec{p}, \vec{q})$ is designed to be satisfiable if in the input $\vec{p}$, n is generated in a depth d pyramidal fashion. To this end, the variables $q_{i,j,a}$ for $(i, j) \in Pyr_d$ and $a \in [n]$ encode a mapping $m : Pyr_d \to [n]$ as in the definition of pyramidal generation in section 3.15, where $q_{i,j,a}$ is intended to express that $m(i, j) = a$.

On the other hand, the set of clauses $Col(\vec{p},\vec{r})$ is designed to be satisfiable if for the input $\vec{p}$, $\textsc{Gen}_n(\vec{p}) = 0$. To achieve this, the variables r_a for $a \in [n]$ encode a coloring of the elements of $[n]$ such that element 1 is colored 0, element n is colored 1, and the elements colored 0 are closed under generation, i.e., if a and b are colored 0 and $a, b \vdash c$, then c is also colored 0.

The set $Gen(\vec{p},\vec{q})$ is given by (4.1)–(4.4), and $Col(\vec{p},\vec{r})$ by (4.5)–(4.7).

$$\bigvee_{1\le a\le n} q_{i,j,a} \qquad \text{for } (i,j) \in Pyr_d, \tag{4.1}$$

$$\bar{q}_{d,j,a} \vee p_{1,1,a} \qquad \text{for } 1 \le j \le d \text{ and } a \in [n], \tag{4.2}$$

$$\bar{q}_{1,1,a} \vee p_{a,a,n} \qquad \text{for } a \in [n], \tag{4.3}$$

$$\bar{q}_{i+1,j,a} \vee \bar{q}_{i+1,j+1,b} \vee \bar{q}_{i,j,c} \vee p_{a,b,c} \qquad \text{for } (i,j) \in Pyr_{d-1} \text{ and } a,b,c \in [n], \tag{4.4}$$

$$\bar{p}_{1,1,a} \vee \bar{r}_a \qquad \text{for } a \in [n], \tag{4.5}$$

$$\bar{p}_{a,a,n} \vee r_a \qquad \text{for } a \in [n], \tag{4.6}$$

$$r_a \vee r_b \vee \bar{p}_{a,b,c} \vee \bar{r}_c \qquad \text{for } a,b,c \in [n]. \tag{4.7}$$

Obviously, $Gen(\vec{p},\vec{q}) \wedge Col(\vec{p},\vec{r})$ is unsatisfiable. Observe that the variables $\vec{p}$ occur only positively in $Gen(\vec{p},\vec{q})$ and only negatively in $Col(\vec{p},\vec{r})$; thus Theorem 2.2 yields an interpolating monotone real formula $C(\vec{p})$.

Now if, for a assignment $\vec{t}$ to the variables $\vec{p}$, n is generated in a depth d pyramidal fashion, then $Gen(\vec{t},\vec{q})$ is satisfiable by setting the values of the variables $q_{i,j,a}$ according to the mapping m. Therefore, $Col(\vec{t},\vec{r})$ must be unsatisfiable, and thus $C(\vec{t}) = 1$.

If, on the other hand, $\textsc{Gen}_n(\vec{t}) = 0$, then $Col(\vec{t},\vec{r})$ can be satisfied by assigning the color 0 to precisely those elements that can be generated in $\vec{t}$. Therefore, $Gen(\vec{t},\vec{q})$ must be unsatisfiable, and so $C(\vec{t}) = 0$.

Thus $C(\vec{p})$ is a monotone real formula satisfying the assumptions of Corollary 3.16, and therefore it has to be of size $2^{\Omega(n^\epsilon)}$. Note that Theorem 2.2 gives no information about the behavior of $C(\vec{t})$ in the case where $Gen(\vec{t},\vec{q})$ and $Col(\vec{t},\vec{r})$ are both unsatisfiable; thus we need Corollary 3.16 in precisely the general form in which it is stated. From the size bounds in Theorem 2.2 we now obtain the following theorem.

THEOREM 4.1. *Every tree-like CP refutation of the clauses $Gen(\vec{p},\vec{q}) \cup Col(\vec{p},\vec{r})$ has to be of size $2^{\Omega(n^\epsilon)}$ for some $\epsilon > 0$.*

On the other hand, there are polynomial size dag-like resolution refutations of these clauses.

THEOREM 4.2. *There are (dag-like) resolution refutations of size $n^{O(1)}$ of the clauses $Gen(\vec{p},\vec{q}) \cup Col(\vec{p},\vec{r})$.*

Proof. First we resolve clauses (4.2) and (4.5) to get

$$\bar{q}_{d,j,c} \vee \bar{r}_c \tag{4.8}$$

for $1 \le j \le d$ and $1 \le c \le n$.

Now we want to derive $\bar{q}_{i,j,c} \vee \bar{r}_c$ for every $(i,j) \in Pyr_d$ and $1 \le c \le n$, by induction on i downward from d to 1. The induction base is just (4.8).

Now by induction we have

$$\bar{q}_{i+1,j,a} \vee \bar{r}_a \quad \text{and} \quad \bar{q}_{i+1,j+1,b} \vee \bar{r}_b\ .$$

We resolve them against (4.7) to get $\bar{q}_{i+1,j,a} \vee \bar{q}_{i+1,j+1,b} \vee \bar{p}_{a,b,c} \vee \bar{r}_c$ for $1 \le a,b,c \le n$ and then resolve them against (4.4) and get

$$\bar{q}_{i+1,j,a} \vee \bar{q}_{i+1,j+1,b} \vee \bar{q}_{i,j,c} \vee \bar{r}_c$$

for every $1 \le a, b \le n$. All of these are then resolved against two instances of (4.1), and we get the desired $\bar{q}_{i,j,c} \vee \bar{r}_c$ for every $1 \le c \le n$.

Finally, we have, in particular, $\bar{q}_{1,1,a} \vee \bar{r}_a$ for every $1 \le c \le n$. We resolve them with (4.6) and get $\bar{q}_{1,1,a} \vee \bar{p}_{a,a,n}$ for every $1 \le a \le n$. These are resolved with (4.3) to get $\bar{q}_{1,1,a}$ for every $1 \le a \le n$. Finally, this clause is resolved with another instance of (4.3) (the one with $i = j = 1$) to get the empty clause. □

It is easy to check that the above refutation is a negative resolution refutation. The following corollary is an easy consequence of the above theorems and known simulation results.

COROLLARY 4.3. *The clauses $Gen(\vec{p}, \vec{q}) \cup Col(\vec{p}, \vec{r})$ exponentially separate tree-like resolution from dag-like resolution; in fact, they separate tree-like resolution from dag-like negative resolution. They also separate tree-like cutting planes from dag-like cutting planes.*

The resolution refutation of $Gen(\vec{p}, \vec{q}) \cup Col(\vec{p}, \vec{r})$ that appears in the proof of Theorem 4.2 is not regular. We do not know whether $Gen(\vec{p}, \vec{q}) \cup Col(\vec{p}, \vec{r})$ has polynomial size regular resolution refutations. To obtain a separation between tree-like resolution and regular resolution we will modify the clauses $Col(\vec{p}, \vec{r})$.

4.1. Separation of tree-like *cp* from regular resolution. The clauses $Col(\vec{p}, \vec{r})$ are modified (and the modification is called $RCol(\vec{p}, \vec{r})$), so that $Gen(\vec{p}, \vec{q}) \cup RCol(\vec{p}, \vec{r})$ allow small regular resolutions, but in such a way that the lower bound proof still applies. We replace the variables r_a by $r_{a,i,D}$ for $a \in [n]$, $1 \le i \le d$, and $D \in \{L, R\}$, giving the coloring of element a, with auxiliary indices i being a row in the pyramid and D distinguishing whether an element is used as a left or right predecessor in the generation process.

The set $RCol(\vec{p}, \vec{r})$ is defined as follows:

$$\bar{p}_{1,1,a} \vee \bar{r}_{a,d,D} \qquad \text{for } a \in [n] \text{ and } D \in \{L, R\}, \tag{4.9}$$

$$\bar{p}_{a,a,n} \vee r_{a,1,D} \qquad \text{for } a \in [n] \text{ and } D \in \{L, R\}, \tag{4.10}$$

$$r_{a,i+1,L} \vee r_{b,i+1,R} \vee \bar{p}_{a,b,c} \vee \bar{r}_{c,i,D} \qquad \text{for } i < d,\ a, b, c \in [n],\ \text{and } D \in \{L, R\}, \tag{4.11}$$

$$\bar{r}_{a,i,D} \vee r_{a,i,\bar{D}} \qquad \text{for } 1 \le i \le d \text{ and } D \in \{L, R\}, \tag{4.12}$$

$$\bar{r}_{a,i,D} \vee r_{a,j,D} \qquad \text{for } 1 \le i, j \le d \text{ and } D \in \{L, R\}. \tag{4.13}$$

Due to the clauses (4.12) and (4.13), the variables $r_{a,i,D}$ are equivalent for all values of the auxiliary indices i, D. Hence a satisfying assignment for $RCol(\vec{p}, \vec{r})$ still codes a coloring of $[n]$ such that elements a with $1, 1 \vdash a$ are colored 0, the elements b with $b, b \vdash n$ are colored 1, and the 0-colored elements are closed under generation. Hence if $RCol(\vec{t}, \vec{r})$ is satisfiable, then $\text{GEN}(\vec{t}) = 0$.

Hence any interpolant for the clauses $Gen(\vec{p}, \vec{q}) \cup RCol(\vec{p}, \vec{r})$ satisfies the assumptions of Corollary 3.16, and we can conclude the following theorem.

THEOREM 4.4. *Tree-like CP refutations of the clauses $Gen(\vec{p}, \vec{q}) \cup RCol(\vec{p}, \vec{r})$ have to be of size $2^{\Omega(n^\epsilon)}$.*

On the other hand, we have the following upper bound on (dag-like) regular resolution refutations of these clauses.

THEOREM 4.5. *There are (dag-like) regular resolution refutations of the clauses $Gen(\vec{p}, \vec{q}) \cup RCol(\vec{p}, \vec{r})$ of size $n^{O(1)}$.*

Proof. First we resolve clauses (4.2) and (4.9) to get

$$\bar{q}_{d,j,a} \vee \bar{r}_{a,d,D} \tag{4.14}$$

for $1 \le j \le d$, $1 \le a \le n$, and $D \in \{L, R\}$. Next we resolve (4.3) and (4.10) to get

$$\bar{q}_{1,1,a} \vee r_{a,1,D} \tag{4.15}$$

for $1 \le a \le n$, and $D \in \{L, R\}$. Finally, from (4.4) and (4.11) we obtain

$$\bar{q}_{i+1,j,a} \vee \bar{q}_{i+1,j+1,b} \vee \bar{q}_{i,j,c} \vee r_{a,i+1,L} \vee r_{b,i+1,R} \vee \bar{r}_{c,i,D} \tag{4.16}$$

for $1 \le j \le i < d$, $1 \le a, b, c \le n$, and $D \in \{L, R\}$.

Now we want to derive $\bar{q}_{i,j,a} \vee \bar{r}_{a,i,D}$ for every $(i, j) \in Pyr_d$, $1 \le a \le n$, and $D \in \{L, R\}$, by induction on i downward from d to 1. The induction base is just (4.14).

For the inductive step, resolve (4.16) against the clauses

$$\bar{q}_{i+1,j,a} \vee \bar{r}_{a,i+1,L} \quad \text{and} \quad \bar{q}_{i+1,j+1,b} \vee \bar{r}_{b,i+1,R} \,,$$

which we have by induction to give

$$\bar{q}_{i+1,j,a} \vee \bar{q}_{i+1,j+1,b} \vee \bar{q}_{i,j,c} \vee \bar{r}_{c,i,D}$$

for every $1 \le a, b \le n$. All of these are then resolved against two instances of (4.1), and we get the desired $\bar{q}_{i,j,c} \vee \bar{r}_{c,i,D}$.

Finally, we have, in particular, $\bar{q}_{1,1,a} \vee \bar{r}_{a,1,L}$, which we resolve against (4.15) to get $\bar{q}_{1,1,a}$ for every $a \le n$. From these and an instance of (4.1) we get the empty clause. □

Note that the refutation given in the proof of Theorem 4.5 is actually an ordered refutation. It respects the following elimination order:

$$\begin{array}{l} p_{1,1,1} \cdots p_{n,n,n} \\ r_{1,d,L} \;\; r_{1,d,R} \;\; \cdots \;\; r_{n,d,L} \;\; r_{n,d,R} \\ q_{1,d,1} \cdots q_{1,d,n} \cdots q_{d,d,1} \cdots q_{d,d,n} \\ r_{1,d-1,L} \cdots r_{n,d-1,R} \;\; q_{1,d-1,1} \cdots q_{d-1,d-1,n} \\ \vdots \\ r_{1,1,L} \;\; r_{1,1,R} \;\; q_{1,1,1} \cdots q_{1,1,n} \,. \end{array}$$

COROLLARY 4.6. *The clauses $Gen(\vec{p}, \vec{q}) \cup RCol(\vec{p}, \vec{r})$ exponentially separate tree-like resolution from ordered resolution; therefore, they also separate exponentially tree-like resolution from regular resolution.*

5. Lower bound for ordered resolution. Goerdt [13] showed that ordered resolution is strictly weaker than unrestricted resolution by giving a superpolynomial lower bound (of the order $\Omega(n^{\log \log n})$) for ordered resolutions of a certain family of clauses, which, on the other hand, has polynomial size unrestricted resolution refutations. In this section we improve this separation to an exponential one; in fact, we give an exponential separation of ordered resolution from negative resolution.

To simplify the exposition, we apply the method of [13] to a set of clauses $SP_{n,m}$ expressing a combinatorial principle that we call the *string-of-pearls* principle. From a bag of m pearls, which are colored red and blue, n pearls are chosen and placed on a string. The string-of-pearls principle $SP_{n,m}$ says that if the first pearl is red and the last one is blue, then there must be a blue pearl next to a red pearl somewhere on the string.

$SP_{n,m}$ is given by an unsatisfiable set of clauses in variables $p_{i,j}$ and q_j for $i \in [n]$ and $j \in [m]$, where $p_{i,j}$ is intended to say that pearl j is at position i on the string, and q_j means that pearl j is colored blue. The clauses forming $SP_{n,m}$ are

$$\bigvee_{j=1}^{m} p_{i,j}, \qquad i \in [n], \tag{5.1}$$

$$\bar{p}_{i,j} \vee \bar{p}_{i,j'}, \qquad i \in [n],\ j, j' \in [m],\ j \neq j', \tag{5.2}$$

$$\bar{p}_{i,j} \vee \bar{p}_{i',j}, \qquad i, i' \in [n],\ j \in [m],\ i \neq i'. \tag{5.3}$$

These first three sets of clauses express that there is a unique pearl at each position.

$$\bar{p}_{1,j'} \vee \bar{q}_{j'}, \qquad j' \in [m], \tag{5.4}$$

$$\bar{p}_{n,j} \vee q_j, \qquad j \in [m], \tag{5.5}$$

$$\bar{p}_{i,j} \vee \bar{p}_{(i+1),j'} \vee q_j \vee \bar{q}_{j'}, \qquad 1 \leq i < n,\ j, j' \in [m],\ j \neq j'. \tag{5.6}$$

These last three sets of clauses express that the first pearl is red, the last one is blue, and that a pearl sitting next to a red pearl is also colored red. The clauses $SP_{n,m}$ are a modified and simplified version of the clauses related to the *st*-connectivity problem that were introduced by Clote and Setzer [8].

PROPOSITION 5.1. *The clauses $SP_{n,m}$ have negative resolution refutations of size $O(nm^2)$.*

Proof. For every $i \in [n]$, we will derive the clauses $\bar{p}_{i,j} \vee \bar{q}_j$ for $j \in [m]$ from $SP_{n,m}$ by a negative resolution derivation. For $i = 1$, these are the clauses (5.4) from $SP_{n,m}$. Inductively, assume we have derived $\bar{p}_{i,j'} \vee \bar{q}_{j'}$ for $j' \in [m]$, and we want to derive $\bar{p}_{(i+1),j} \vee \bar{q}_j$ from these.

Consider the clauses (5.6) of the form $\bar{p}_{i,j'} \vee \bar{p}_{(i+1),j} \vee q_{j'} \vee \bar{q}_j$ for $j' \in [m]$. Using the inductive assumption, we derive from these the clauses $\bar{p}_{i,j'} \vee \bar{p}_{(i+1),j} \vee \bar{q}_j$ for $j' \in [m]$. Note that these are negative clauses.

By a derivation of length m, we obtain $\bar{p}_{(i+1),j} \vee \bar{q}_j$ from these and the clause $\bigvee_{j' \in [m]} p_{i,j'}$ from $SP_{n,m}$. The whole derivation is of length $O(m)$, and we need m of them, giving a total length of $O(m^2)$ for the induction step.

We end up with a derivation of the clauses $\bar{p}_{n,j} \vee \bar{q}_j$ for $j \in [m]$ of length $O(nm^2)$. In another m steps we resolve these with the initial clauses (5.5), obtaining the singleton clauses $\bar{p}_{n,j}$ for $j \in [m]$. Finally, we derive a contradiction from these and the clauses $\bigvee_{j \in [m]} p_{n,j}$. □

The above refutation of $SP_{n,m}$ is not ordered, since it is not even regular: the variables q_j for every pearl j are eliminated at every stage of the induction. Nevertheless, we are unable to show that there are no short ordered refutations of $SP_{n,m}$. In order to obtain a lower bound for ordered resolution refutations, we shall modify the clauses $SP_{n,m}$. The lower bound is then proved by a bottleneck counting argument similar to that used in [13], which is based on the original argument of Haken [16]. Note that the clauses (5.1)–(5.3) are similar to the clauses expressing the pigeonhole principle, which makes the bottleneck counting technique applicable in our situation.

We call the pearls numbered 1 through $\frac{n}{4}$ (we assume $\frac{n}{4}$ to be an integer, for simplicity) the *special* pearls. The positions 1 to $\frac{n}{2}$ on the string are called the *left half*, and the positions $\frac{n}{2} + 1$ to n are called the *right half* of the string.

For each special pearl j placed on the string, an associated position $\hat{\imath} = \hat{\imath}(j)$ is defined, depending on where on the string j is placed. If j is placed in the left half,

then $\hat{\imath}$ is in the right half; say, $\hat{\imath} = \frac{n}{2} + 2j - 1$ for definiteness, and if j is placed in the right half, then $\hat{\imath}$ is in the left half, say, $\hat{\imath} = 2j$.

The set $SP'_{n,m}$ is obtained from $SP_{n,m}$ by adding additional literals to those clauses that restrict the coloring of the special pearls placed on the string. First, the clauses (5.4) and (5.6) for $1 \leq i < \frac{n}{2}$, where $j' \leq \frac{n}{4}$ is special, are replaced by m clauses each, namely,

$$\bar{p}_{\hat{\imath},\ell} \vee \bar{p}_{1,j'} \vee \bar{q}_{j'}, \tag{5.7}$$

$$\bar{p}_{\hat{\imath},\ell} \vee \bar{p}_{i,j} \vee \bar{p}_{(i+1),j'} \vee q_j \vee \bar{q}_{j'} \tag{5.8}$$

for every $\ell \in [m]$, where $\hat{\imath} := \frac{n}{2} + 2j' - 1$, since j' is placed in the left half. Similarly, the clauses (5.5) and (5.6) for $\frac{n}{2} < i < n$ and special $j \leq \frac{n}{4}$ are replaced by

$$\bar{p}_{\hat{\imath},\ell} \vee \bar{p}_{n,j} \vee q_j, \tag{5.9}$$

$$\bar{p}_{\hat{\imath},\ell} \vee \bar{p}_{i,j} \vee \bar{p}_{(i+1),j'} \vee q_j \vee \bar{q}_{j'} \tag{5.10}$$

for every $\ell \in [m]$, where now $\hat{\imath} := 2j$, since j is placed in the right half. All other clauses remain unchanged. The modified clauses $SP'_{n,m}$ do not have an intuitive combinatorial interpretation different from the meaning of the original clauses $SP_{n,m}$. The added literals only serve to make the clauses hard for ordered refutations. The idea is that for the clauses (5.7)–(5.10) to be used as one would use the original (5.4)–(5.6) in the natural short, inductive proof above, the additional literals $\bar{p}_{\hat{\imath},\ell}$ have to be removed first. The positions $\hat{\imath}$ are chosen in such a way that this cannot be done in a manner consistent with a global ordering of the variables.

THEOREM 5.2. *The clauses $SP'_{n,m}$ have negative resolution refutations of size $O(nm^2)$.*

Proof. We modify the refutation of $SP_{n,m}$ given above for the modified clauses $SP'_{n,m}$. First, note that the original clauses (5.4) can be obtained from (5.7) by a negative derivation of length m.

Next, we modify those places in the inductive step where the clauses (5.6) are used that have been modified. First, we resolve the modified clauses (5.8), respectively, (5.10) with the inductive assumption, yielding the negative clauses

$$\bar{p}_{\hat{\imath},\ell} \vee \bar{p}_{i,j} \vee \bar{p}_{(i+1),j'} \vee \bar{q}_{j'}$$

for $\ell \in [m]$. These are then resolved with the clause $\bigvee_{j=1}^{m} p_{\hat{\imath},j}$, after which we can continue as in the original refutation.

In the places where the clauses (5.5) are used in the original refutation, we first resolve (5.9) with the clauses $\bar{p}_{n,j} \vee \bar{q}_j$, yielding $\bar{p}_{\hat{\imath},\ell} \vee \bar{p}_{n,j}$, which can be resolved with $\bigvee_{j=1}^{m} p_{\hat{\imath},j}$ to get the singleton clauses $\bar{p}_{n,j}$ as in the original refutation. ∎

In particular, there are polynomial size unrestricted resolution refutations of the clauses $SP'_{n,m}$. The next theorem gives a lower bound for ordered resolution refutations of these clauses.

THEOREM 5.3. *For sufficiently large n and $m \geq \frac{9}{8}n$, every ordered resolution refutation of the clauses $SP'_{n,m}$ contains at least $2^{k(\log n - 5)}$ clauses.*

For the sake of simplicity, let n be divisible by 8, say, $n = 8k$. Let $N := nm + m$ be the number of variables, and let an ordering $x_1, x_2, \ldots, x_N$ of the variables be given, i.e., each x_ν is one of the variables $p_{i,j}$ or q_j. Let R be an ordered resolution refutation of $SP'_{n,m}$ respecting this elimination ordering, i.e., on every path through R the variables are eliminated in the prescribed order. We shall show that R contains at least $k!$ different clauses, which is at least $2^{\frac{n}{8}(\log n - 5)}$ for large n.

For a position $i \in [n]$ and $\nu \leq N$, let $S(i,\nu)$ be the set of special pearls $j \leq 2k = \frac{n}{4}$ such that $p_{i,j}$ is among the first ν eliminated variables, i.e.,

$$S(i,\nu) := \{\, j \leq 2k \;:\; p_{i,j} \in \{x_1, \ldots, x_\nu\}\,\} \;.$$

Let ν_0 be the smallest index such that $|S(i,\nu_0)| = k$ for some position i, and call this position i_0. It follows that for all $i \neq i_0$, $|S(i,\nu_0)| < k$. In other words, i_0 is the first position for which k of the variables $p_{i_0,j}$ with $j \leq 2k$ special are eliminated.

Let the elements of $S(i_0,\nu_0)$ be denoted by $j_1, \ldots, j_k$, enumerated in increasing order for definiteness. For each $1 \leq \mu \leq k$, let i_μ be the position $\hat{\imath}(j_\mu)$ associated to j_μ when j_μ is placed on the string at position i_0, i.e.,

$$i_\mu := \begin{cases} \frac{n}{2} + 2j_\mu - 1 & \text{if } i_0 \leq \frac{n}{2}, \\ 2j_\mu & \text{if } i_0 > \frac{n}{2}. \end{cases}$$

Further, we define for the set $R_\mu := [2k] \setminus S(i_\mu,\nu_0)$, i.e., R_μ is the set of special pearls j with the property that on every path in the refutation, the variable $p_{i_\mu,j}$ is eliminated only after all the variables p_{i_0,j_κ} for $1 \leq \kappa \leq k$ have been eliminated. Note that by the definition of ν_0, $|S(i_\mu,\nu_0)| < k$ and therefore $|R_\mu| \geq k$ for all $1 \leq \mu \leq k$.

DEFINITION 5.4. *A* critical assignment *is an assignment that satisfies all the clauses of $SP'_{n,m}$ except for exactly one of the clauses* (5.1)*. From a critical assignment α, we define the following data.*

- *The unique position $i_\alpha \in [n]$ such that no pearl is placed at position i_α by α, i.e., $\alpha(p_{i_\alpha,j}) = 0$ for every $j \in [m]$. We call i_α the* gap *of α.*
- *A* 1-1 *mapping $m_\alpha : [n] \setminus \{i_\alpha\} \to [m]$, where for every $i \neq i_\alpha$, $m_\alpha(i)$ is the pearl placed at position i by α, i.e., the unique $j \in [m]$ such that $\alpha(p_{i,j}) = 1$.*

For every $j \in [m]$, we refer to the value $\alpha(q_j)$ as the color *of j, where we identify the value* 0 *with red and* 1 *with blue.*

A critical assignment α is called 0-critical *if the gap is $i_\alpha = i_0$ and $m_\alpha(i_\mu) \in R_\mu$ for each $1 \leq \mu \leq k$, and, moreover,*

- *if i_0 is in the left half, then $j_1, \ldots, j_k$ are colored blue (i.e., $\alpha(q_{j_1}) = \cdots = \alpha(q_{j_k}) = 1$), and*
- *if i_0 is in the right half, then $j_1, \ldots, j_k$ are colored red (i.e., $\alpha(q_{j_1}) = \cdots = \alpha(q_{j_k}) = 0$).*

Note that the positions $i_0, i_1, \ldots, i_k$ and the pearls $j_1, \ldots, j_k$, and thus the notion of 0-critical assignment, only depend on the elimination order and not on the refutation R.

As in other bottleneck counting arguments, the lower bound will now be proved in two steps. First, we show that there are many 0-critical assignments. Second, we will map each 0-critical assignment α to a certain clause C_α in R, and then show that not too many different assignments α can be mapped to the same clause C_α, and thus that there must be many of the clauses C_α.

The first goal, showing there are many 0-critical assignments, is attained with the following claim.

CLAIM 5.5. *For every choice of pairwise distinct pearls $b_1, \ldots, b_k$ with $b_\mu \in R_\mu$ for $1 \leq \mu \leq k$, there is a 0-critical assignment α with $m_\alpha(i_\mu) = b_\mu$ for $1 \leq \mu \leq k$. In particular, there are at least $k!$ 0-critical assignments that disagree on the values $m_\alpha(i_\mu)$ for $1 \leq \mu \leq k$.*

Proof (proof of Claim 5.5). For those positions i such that $m_\alpha(i)$ is not defined yet, i.e., $i \notin \{i_0, i_1, \ldots, i_k\}$, assign pearls $m_\alpha(i) \in [m] \setminus \{j_1, \ldots, j_k\}$ arbitrarily but

consistently, i.e., choose an arbitrary 1-1 mapping from $[n] \setminus \{i_0, i_1, \dots, i_k\}$ to $[m] \setminus \{b_1, \dots b_k, j_1, \dots, j_k\}$. This is always possible, since by assumption $m \geq 9k$.

Finally, color those pearls that are assigned to positions to the left of the gap red, and those that are assigned to positions to the right of the gap blue, i.e., set $\alpha(q_{m_\alpha(i)}) = 0$ for $i < i_0$ and $\alpha(q_{m_\alpha(i)}) = 1$ for $i > i_0$. The pearls $j_1, \dots, j_k$ are colored according to the requirement in the definition of a 0-critical assignment.

This coloring of the pearls is well defined even if some of the pearls $b_1, \dots b_k$ are among the $j_1, \dots, j_k$, because the positions $i_1, \dots, i_k$ and i_0 are in opposing halves of the string: if i_0 is in the left half, then every i_μ is in the right half, and, in particular, $i_\mu > i_0$. Similarly, if i_0 is in the right half, then $i_\mu < i_0$, so in both cases, the pearls $j_1, \dots, j_k$ get the same color as $b_1, \dots, b_k$. The remaining pearls can be colored arbitrarily. □

Now we map 0-critical assignments to certain clauses in R. For a 0-critical assignment α, let C_α be the first clause in R such that α does not satisfy C_α, and

$$\{\, j \,:\, p_{i_0,j} \text{ occurs in } C_\alpha \,\} = [m] \setminus \{j_1, \dots, j_k\} \,.$$

This clause exists because α determines a path through R from the clause $\bigvee_{j \in [m]} p_{i_0,j}$ to the empty clause, such that α does not satisfy any clause on this path. The variables $p_{i_0,j}$ with $j \leq 2k$ are eliminated along that path, and $p_{i_0,j_1}, \dots p_{i_0,j_k}$ are the first among them in the elimination order.

Claim 5.6. *Let α be a 0-critical assignment. For every $1 \leq \mu \leq k$, the literal $\bar{p}_{i_\mu,\ell_\mu}$, where $\ell_\mu := m_\alpha(i_\mu)$, occurs in C_α.*

Proof (proof of Claim 5.6). Let α' be the assignment defined by $\alpha'(p_{i_0,j_\mu}) := 1$ and $\alpha'(x) := \alpha(x)$ for all other variables x. As p_{i_0,j_μ} does not occur in C_α, α' does not satisfy C_α either.

There is exactly one clause in $SP'_{n,m}$ that is not satisfied by α', depending on where the gap i_0 is; this clause is

$$\begin{aligned}
i_0 = 1 \;:&\quad \bar{p}_{i_\mu,\ell_\mu} \vee \bar{p}_{1,j_\mu} \vee \bar{q}_{j_\mu}, \\
1 < i_0 \leq \frac{n}{2} \;:&\quad \bar{p}_{i_\mu,\ell_\mu} \vee \bar{p}_{i_0-1,h} \vee \bar{p}_{i_0,j_\mu} \vee q_h \vee \bar{q}_{j_\mu}, \qquad \text{where } h = m_\alpha(i_0 - 1), \\
\frac{n}{2} < i_0 < n \;:&\quad \bar{p}_{i_\mu,\ell_\mu} \vee \bar{p}_{i_0,j_\mu} \vee \bar{p}_{i_0+1,h} \vee q_{j_\mu} \vee \bar{q}_h, \qquad \text{where } h = m_\alpha(i_0 + 1), \\
i_0 = n \;:&\quad \bar{p}_{i_\mu,\ell_\mu} \vee \bar{p}_{n,j_\mu} \vee q_{j_\mu}.
\end{aligned}$$

The requirement for the coloring of the j_μ in the definition of a 0-critical assignment entails that these clauses are not satisfied by α' and that all other clauses are satisfied by α'.

In any case, the literal $\bar{p}_{i_\mu,\ell_\mu}$ occurs in this clause, and there is a path through R leading from the clause in question to C_α, such that α' does not satisfy any clause on that path. The variable that is eliminated in the last inference on that path must be one of the p_{i_0,j_κ} for $1 \leq \kappa \leq k$, by the definition of C_α. Since $\ell_\mu \in R_\mu$, the variable p_{i_μ,ℓ_μ} appears after p_{i_0,j_κ} in the elimination order, by the definition of R_μ . Therefore, p_{i_μ,ℓ_μ} cannot have been eliminated on that path, so $\bar{p}_{i_\mu,\ell_\mu}$ still occurs in C_α. □

Finally, we are ready to finish the proof of the theorem. Let α, β be two 0-critical assignments such that $\ell_\mu := m_\alpha(i_\mu) \neq m_\beta(i_\mu)$ for some $1 \leq \mu \leq k$, so that $\beta(p_{i_\mu,\ell_\mu}) = 0$. By Claim 5.6, the literal $\bar{p}_{i_\mu,\ell_\mu}$ occurs in C_α; therefore, β satisfies C_α, and hence $C_\beta \neq C_\alpha$.

By Claim 5.5, there are at least $k!$ 0-critical assignments α that disagree on at least one of the values $m_\alpha(i_\mu)$. Thus R contains at least $k!$ distinct clauses of the form C_α. □

The following corollary is a direct consequence of Theorems 5.3 and 5.2.

COROLLARY 5.7. *The clauses $SP'_{n,m}$ for $m \geq \frac{9}{8}n$ exponentially separate ordered resolution from unrestricted resolution and negative resolution.*

A modification similar to the one that transforms $SP_{n,m}$ into $SP'_{n,m}$ can also be applied to the clauses $Gen(\vec{p}, \vec{q})$, yielding a set $DPGen(\vec{p}, \vec{q})$. Then for the clauses $DPGen(\vec{p}, \vec{q}) \cup Col(\vec{p}, \vec{r})$, an exponential lower bound for ordered resolutions can be proved by the method of Theorem 5.3 (this was presented in the conference version [5] of this paper). Also the negative resolution proofs of Theorem 4.2 can be modified for these clauses. Thus the clauses $DPGen(\vec{p}, \vec{q}) \cup Col(\vec{p}, \vec{r})$ exponentially separate ordered from negative resolution as well.

6. Open problems. We would like to conclude by stating some open problems related to the topics of this paper.

1. For boolean circuits (monotone as well as general), circuit depth and formula size are essentially the same complexity measure, as they are exponentially related by the well-known Brent–Spira theorem. Is there an analogous theorem for monotone real circuits, i.e., is $d_{\mathbb{R}}(f) = \Theta(\log s_{\mathbb{R}}(f))$ for every monotone function f? This would be implied by the converse to Lemma 2.1, i.e., $d_{\mathbb{R}}(f) \leq CC_{\mathbb{R}}(R_f)$. Does this hold for every monotone function f?
2. The separation between tree-like and dag-like resolution was recently improved to a strongly exponential one, with a lower bound of the form $2^{n/\log n}$ [3, 4, 24]. Can we prove the same strong separation between tree-like and dag-like CP?
3. A solution for the previous problem would follow from a strongly exponential separation of monotone real formula size from monotone circuit size. Such a strong separation is not even known for monotone *boolean* circuits.
4. Can the superpolynomial separations of regular and negative resolution from unrestricted resolution [14, 15] be improved to exponential as well? And is there an exponential speed-up of regular over ordered resolution?

Acknowledgments. We would like to thank Ran Raz for reading a previous version of this work and discovering an error, Andreas Goerdt for sending us copies of his papers, Sam Buss for helpful discussions, and, finally, Peter Clote for suggesting to us to work on resolution separations.

REFERENCES

[1] P. BEAME, R. KARP, T. PITASSI, AND M. SAKS, *The efficiency of resolution and Davis-Putnam procedures*, submitted, 1999.

[2] P. BEAME AND T. PITASSI, *Simplified and improved resolution lower bounds*, in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 274–282.

[3] E. BEN-SASSON AND A. WIGDERSON, *Short proofs are narrow—resolution made simple*, in Proceedings of the 31st ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 517–526.

[4] E. BEN-SASSON, R. IMPAGLIAZZO, AND A. WIGDERSON, *Near-Optimal Separation of Treelike and General Resolution*, Electronic Colloquium on Computational Complexity TR 00-005, http://www.eccc.uni-trier.de/eccc-local/Lists/TR-2000.html.

[5] M. L. BONET, J. L. ESTEBAN, N. GALESI, AND J. JOHANNSEN, *Exponential separations between restricted resolution and cutting planes proof systems*, in Proceedings of the 39th Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1998, pp. 638–647.

[6] M. L. Bonet, T. Pitassi, and R. Raz, *Lower bounds for cutting planes proofs with small coefficients*, J. Symbolic Logic, 62 (1997), pp. 708–728. Preliminary version in Proceedings of the 27th ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 575–584.
[7] V. Chvátal and E. Szemerédi, *Many hard examples for resolution*, J. ACM, 35 (1988), pp. 759–768.
[8] P. Clote and A. Setzer, *On PHP, st-connectivity and odd charged graphs*, in Proof Complexity and Feasible Arithmetics, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 39, P. Beame and S. R. Buss, eds., AMS, Providence, RI, 1998, pp. 93–117.
[9] S. Cook and A. Haken, *An exponential lower bound for the size of monotone real circuits*, J. Comput. System Sci., 58 (1999), pp. 326–335.
[10] S. A. Cook and R. A. Reckhow, *The relative efficiency of propositional proof systems*, J. Symbolic Logic, 44 (1979), pp. 36–50.
[11] W. Cook, C. Coullard, and G. Turán, *On the complexity of cutting plane proofs*, Discrete Appl. Math., 18 (1987), pp. 25–38.
[12] X. Fu, *Lower bounds on sizes of cutting planes proofs for modular coloring principles*, in Proof Complexity and Feasible Arithmetics, DIMACS Ser. Discrete Math. Theoret. Comput. Sci. 39, P. Beame and S. R. Buss, eds., AMS, Providence, RI, 1998, pp. 135–148.
[13] A. Goerdt, *Davis-Putnam resolution versus unrestricted resolution*, Ann. Math. Artificial Intelligence, 6 (1992), pp. 169–184.
[14] A. Goerdt, *Unrestricted resolution versus N-resolution*, Theoret. Comput. Sci., 93 (1992), pp. 159–167.
[15] A. Goerdt, *Regular resolution versus unrestricted resolution*, SIAM J. Comput., 22 (1993), pp. 661–683.
[16] A. Haken, *The intractability of resolution*, Theoret. Comput. Sci., 39 (1985), pp. 297–308.
[17] R. Impagliazzo, T. Pitassi, and A. Urquhart, *Upper and lower bounds for tree-like cutting planes proofs*, in Proceedings of the 9th IEEE Symposium on Logic in Computer Science, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 220–228.
[18] J. Johannsen, *Lower bounds for monotone real circuit depth and formula size and tree-like cutting planes*, Inform. Process Lett., 67 (1998), pp. 37–41.
[19] S. Jukna, *Combinatorics of monotone computations*, Combinatorica, 19 (1999), pp. 65–85. Preliminary version available as Electronic Colloquium on Computational Complexity TR98-041, 1998, http://www.eccc.uni-trier.de/eccc-local/Lists/TR-1999.html.
[20] M. Karchmer and A. Wigderson, *Monotone circuits for connectivity require super-logarithmic depth*, SIAM J. Discrete Math., 3 (1990), pp. 255–265. Preliminary version in Proceedings of the 20th ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 539–550.
[21] J. Krajíček, *Interpolation theorems, lower bounds for proof systems and independence results for bounded arithmetic*, J. Symbolic Logic, 62 (1997), pp. 457–486.
[22] J. Krajíček, *Interpolation by a game*, MLQ Math. Log. Q., 44 (1998), pp. 450–458.
[23] P. Pudlák, *Lower bounds for resolution and cutting plane proofs and monotone computations*, J. Symbolic Logic, 62 (1997), pp. 981–998.
[24] P. Pudlák and R. Impagliazzo, *A lower bound for DLL algorithms for k-SAT*, in Proceedings of the ACM Symposium on Discrete Algorithms, ACM, New York, 2000, pp. 128–136.
[25] R. Raz and P. McKenzie, *Separation of the monotone NC hierarchy*, Combinatorica, 19 (1999), pp. 403–435. Preliminary version in Proceedings of the 38th Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1997, pp. 234–243.
[26] A. Rosenbloom, *Monotone real circuits are more powerful than monotone boolean circuits*, Inform. Process Lett., 61 (1997), pp. 161–164.
[27] U. Schöning, *Logic for Computer Scientists*, Birkhäuser, Basel, 1989.
[28] A. Urquhart, *Hard examples for resolution*, J. ACM, 34 (1987), pp. 209–219.
[29] A. Urquhart, *The complexity of propositional proofs*, Bull. Symbolic Logic, 1 (1995), pp. 425–467.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1485–1501

RANDOMNESS IS HARD*

HARRY BUHRMAN† AND LEEN TORENVLIET‡

Abstract. We study the set of incompressible strings for various resource bounded versions of Kolmogorov complexity. The resource bounded versions of Kolmogorov complexity we study are polynomial time *CD* complexity defined by Sipser, the nondeterministic variant *CND* due to Buhrman and Fortnow, and the polynomial space bounded Kolmogorov complexity *CS* introduced by Hartmanis. For all of these measures we define the set of random strings R_t^{CD}, R_t^{CND}, and R_t^{CS} as the set of strings x such that $CD^t(x)$, $CND^t(x)$, and $CS^s(x)$ is greater than or equal to the length of x for s and t polynomials. We show the following:

- $\mathrm{MA} \subseteq \mathrm{NP}^{\mathrm{R}_t^{CD}}$, where MA is the class of Merlin–Arthur games defined by Babai.
- $\mathrm{AM} \subseteq \mathrm{NP}^{\mathrm{R}_t^{CND}}$, where AM is the class of Arthur–Merlin games.
- $\mathrm{PSPACE} \subseteq \mathrm{NP}^{\mathrm{cR}_s^{CS}}$.

In the last item cR_s^{CS} is the set of pairs $\langle x, y\rangle$ so that x is random given y. These results show that the set of random strings for various resource bounds is hard for complexity classes under *nondeterministic* reductions.

This paper contrasts the earlier work of Buhrman and Mayordomo where they show that for polynomial time *deterministic* reductions the set of exponential time Kolmogorov random strings is not complete for EXP.

Key words. complexity classes, *CD* complexity, *CND* complexity, interactive proofs, Kolmogorov complexity, Merlin–Arthur, Arthur–Merlin, randomness, relativization

AMS subject classifications. 03D15, 68Q10, 68Q15, 68Q17, 68Q19

PII. S0097539799360148

1. Introduction.

The holy grail of complexity theory is the separation of complexity classes like P, NP, and PSPACE. It is well known that all of these classes possess complete sets and that it is thus sufficient for a separation to show that a complete set of one class is not contained in the other. Therefore lots of effort was put into the study of complete sets. (See [11].)

Kolmogorov [19], however, suggested focusing attention on sets which are *not* complete. His intuition was that complete sets possess a lot of "structure" that hinders a possible lower bound proof. He suggested to look at the set of time bounded Kolmogorov random strings. In this paper we will continue this line of research and study variants of this set.

Kolmogorov complexity measures the "amount" of regularity in a string. Informally the Kolmogorov complexity of a string x, denoted as $C(x)$, is the size of the smallest program that prints x and then stops. For any string x, $C(x)$ is less than or equal to the length of x (up to some additive constant). Those strings for which it holds that $C(x)$ is greater than or equal to the length of x are called *incompressible* or *random*. A simple counting argument shows that random strings exist.

*Received by the editors August 17, 1999; accepted for publication (in revised form) February 8, 2000; published electronically November 8, 2000.

http://www.siam.org/journals/sicomp/30-5/36014.html

†CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (buhrman@cwi.nl). The work of this author was partially supported by the Dutch foundation for scientific research (NWO) by SION project 612-34-002, the European Union through NeuroCOLT ESPRIT Working Group 8556, HC&M grant ERB4050PL93-0516, and the EU fifth framework program QAIP, IST-1999-11234.

‡Department of Computer Science, University of Amsterdam, 24 Plantage Muidergracht, 1018 TV Amsterdam, The Netherlands (leen@wins.uva.nl).

In the sixties, when the theory of Kolmogorov complexity was developed, Martin [23] showed that the coRE set of Kolmogorov random strings is complete with respect to (resource unbounded) Turing reductions. Kummer [18] has shown that this can be strengthened to show that this set is also truth-table complete.

The resource bounded version of the set of random strings was first studied by Ko [17]. The *polynomial* time bounded Kolmogorov complexity $C^p(x)$ for p a polynomial is the smallest program that prints x in $p(|x|)$ steps [14]. Ko showed that there exists an oracle such that the set of random strings with respect to this time bounded Kolmogorov complexity is complete for coNP under strong nondeterministic polynomial time reductions. He also constructed an oracle where this set is not complete for coNP under deterministic polynomial time Turing reductions.

Buhrman and Mayordomo [10] considered the *exponential* time Kolmogorov random strings. The exponential time Kolmogorov complexity $C^t(x)$ is the smallest program that prints x in $t(|x|)$ steps for functions $t(n) = 2^{n^k}$. They showed that the set of $t(n)$ random strings is *not* deterministic polynomial time Turing hard for EXP. They showed that the class of sets that reduce to this set has p measure 0 and hence that this set is not even weakly hard for EXP.

The results in this paper contrast those from Buhrman and Mayordomo. We show that the set of random strings is hard for various complexity classes under *nondeterministic* polynomial time reductions.

We consider three well-studied measures of Kolmogorov complexity that lie in between $C^p(x)$ and $C^t(x)$ for p a polynomial and $t(n) = 2^{n^k}$. We consider the distinguishing complexity as introduced by Sipser [25]. The distinguishing complexity, $CD^t(x)$, is the size of the smallest program that runs in time $t(n)$ and accepts x and nothing else. We show that the set of random strings $\mathrm{R}_t^{CD} = \{x \mid CD^t(x) \geq |x|\}$ for t a fixed polynomial is hard for MA under nondeterministic reductions. MA is the class of Merlin–Arthur games introduced by Babai [1]. As an immediate consequence we obtain that BPP and $\mathrm{NP}^{\mathrm{BPP}}$ are in $\mathrm{NP}^{\mathrm{R}_t^{CD}}$.

Next we shift our attention to nondeterministic distinguishing complexity [6], $CND^t(x)$, which is defined as the size of the smallest *nondeterministic* algorithm that runs in time $t(n)$ and accepts only x. We define $\mathrm{R}_t^{CND} = \{x : CND^t(x) \geq |x|\}$ for t a fixed polynomial. We show that $\mathrm{AM} \subseteq \mathrm{NP}^{\mathrm{R}_t^{CND}}$, where AM is the class of Arthur–Merlin games [1]. It follows that the complement of the graph isomorphism problem, $\overline{GI}$, is in $\mathrm{NP}^{\mathrm{R}_t^{CND}}$ and that if for some polynomial t, $\mathrm{R}_t^{CND} \in \mathrm{NP} \cap \mathrm{coNP}$, then $GI \in \mathrm{NP} \cap \mathrm{coNP}$.

The $s(n)$ *space* bounded Kolmogorov complexity $CS^s(x|y)$ is defined as the size of the smallest program that prints x, given y, and uses at most $s(|x| + |y|)$ tape cells [14]. Likewise we define $\mathrm{cR}_s^{CS} = \{\langle x, y\rangle : CS^s(x|y) \geq |x|\}$ for $s(n)$ a polynomial. We show that $\mathrm{PSPACE} \subseteq \mathrm{NP}^{\mathrm{cR}_s^{CS}}$.

For the first two results we use the oblivious sampler construction of Zuckerman [29], a lemma [6] that measures the size of sets in terms of CD complexity, and we prove a lemma that shows that the first bits of a random string are in a sense more random than the whole string. For the last result we make use of the interactive protocol [22, 24] for quantified boolean formulas (QBFs).

To show optimality of our results for relativizing techniques, we construct an oracle world where our first result cannot be improved to deterministic reductions. We show that there is an oracle such that $\mathrm{BPP} \not\subseteq \mathrm{P}^{\mathrm{R}_t^{CD}}$ for any polynomial t. The construction of the oracle is an extension of the techniques developed by Beigel, Buhrman, and Fortnow [4].

2. Definitions and notations. We assume the reader is familiar with standard notions in complexity theory as can be found, e.g., in [2]. Strings are elements of Σ^*, where $\Sigma = \{0,1\}$. For a string s and integers $n, m \leq |s|$ we use the notation $s[n..m]$ for the string consisting of the nth through mth bit of s. We use λ for the empty string. We also need the notion of an *oblivious sampler* from [29].

DEFINITION 2.1. *A universal $(r, d, m, \epsilon, \gamma)$-oblivious sampler is a deterministic algorithm which on input a uniformly random r-bit string outputs a sequence of points $z_1, \ldots, z_d \in \{0,1\}^m$ such that for any collection of d functions $f_1, \ldots, f_d : \{0,1\}^m \mapsto [0,1]$ it is the case that*

$$\Pr\left[\left|\frac{1}{d}\sum_{i=1}^{d}(f_i(z_i) - Ef_i)\right| \leq \epsilon\right] \geq 1 - \gamma$$

(where $Ef_i = 2^{-m}\sum_{z\in\{0,1\}^m} f_i(z)$).

In our application of this definition, we will always use a single function f.

Fix a universal Turing machine U and a nondeterministic universal machine U_{N}. All our results are independent of the particular choice of universal machine. For the definition of Kolmogorov complexity we need the fact that the universal machine can, on input p, y, halt and output a string x. For the definition of distinguishing complexity below we need the fact that the universal machine on input p, x, y can either accept or reject. We also need resource bounded versions of this property.

We define the Kolmogorov complexity function $C(x|y)$ (see [20]) by $C(x|y) = \min\{|p| : U(p,y) = x\}$. We define unconditional Kolmogorov complexity by $C(x) = C(x|\lambda)$. Hartmanis [14] defined a time bounded version of Kolmogorov complexity, but resource bounded versions of Kolmogorov complexity date back as far as [3]. (See also [20].) Sipser [25] defined the distinguishing complexity CD^t.

We will need the following versions of resource bounded Kolmogorov complexity and distinguishing complexity.

- $CS^s(x|y) = \min\left\{|p| : \begin{array}{ll}(1) & U(p,y) = x;\\ (2) & U(p,y) \text{ uses at most } s(|x|+|y|) \text{ tape cells}\end{array}\right\}$.
  (See [14].)
- $CD^t(x|y) = \min\left\{|p| : \begin{array}{ll}(1) & U(p,x,y) \text{ accepts};\\ (2) & (\forall z \neq x)U(p,z,y) \text{ rejects};\\ (3) & (\forall z \in \Sigma^*)U(p,z,y) \text{ runs in at most } t(|u|+|y|) \text{ steps}\end{array}\right\}$.
  (See [25].)
- $CND^t(x|y) = \min\left\{|p| : \begin{array}{ll}(1) & U_{\mathrm{N}}(p,x,y) \text{ accepts};\\ (2) & (\forall z \neq x)U_{\mathrm{N}}(p,z,y) \text{ rejects};\\ (3) & (\forall z \in \Sigma^*)U_{\mathrm{N}}(p,z,y) \text{ runs in at most } t(|z|+|y|) \text{ steps}\end{array}\right\}$.
  (See [6].)

For $0 < \epsilon \leq 1$ we define the following sets of strings of "maximal" CD^p and CND^p complexity.

- $\mathrm{R}^{CD}_{t,\epsilon} = \{x : CD^t(x|\lambda) \geq \epsilon|x|\}$.
- $\mathrm{R}^{CND}_{t,\epsilon} = \{x : CND^t(x|\lambda) \geq \epsilon|x|\}$.

Note that for $\epsilon = 1$ these sets are the sets mentioned in the introduction. In this case we will omit the ϵ and use R^{CD}_t and R^{CND}_t. We also define the set of strings of

maximal space bounded complexity.

$$\mathrm{cR}_s^{CS} = \{\langle x, y\rangle : C\mathrm{S}^s(x|y) \geq |x|\}.$$

The c in the notation is to emphasize that randomness is conditional. Also, cR_s^{CS} technically is a set of pairs rather than a set of strings. The unconditional space bounded random strings would be

$$\mathrm{R}_s^{CS} = \{x : \langle x, \lambda\rangle \in \mathrm{cR}_s^{CS}\}.$$

We have no theorems concerning this set.

The C complexity of a string is always upperbounded by its length plus some constant depending only on the choice of the universal machine. The *CD* and *CND* complexities of a string are always upperbounded by the C complexity of that string plus some constant depending again only on the particular choice of universal machine. All quantifiers used in this paper are polynomially bounded. Often the particular polynomial is not important for what follows, or it is clear from the context and is omitted. Sometimes we need explicit bounds. Then the particular bound is given as a superscript to the quantifier. For example, we use $\exists^m y$ to denote "there exists a y with $|y| \leq m$," or $\forall^{=n} x$ to denote "for all x of length n."

The classes MA and AM are defined as follows.

DEFINITION 2.2. *$L \in \mathrm{MA}$ iff there exists a $|x|^c$ time bounded machine M such that*

1. $x \in L \implies \exists y \Pr[M(x, y, r) = 1] > 2/3$;
2. $x \notin L \implies \forall y \Pr[M(x, y, r) = 1] < 1/3$,

where r is chosen uniformly at random in $\{0,1\}^{|x|^c}$.

$L \in \mathrm{AM}$ iff there exists a $|x|^c$ time bounded machine M such that

1. $x \in L \implies \Pr[\exists y M(x, y, r) = 1] > 2/3$;
2. $x \notin L \implies \Pr[\exists y M(x, y, r) = 1] < 1/3$,

where r is chosen uniformly at random in $\{0,1\}^{|x|^c}$.

It is known that $\mathrm{NP} \cup \mathrm{BPP} \subseteq \mathrm{MA} \subseteq \mathrm{AM} \subseteq \mathrm{PSPACE}$ [1].

Let $\#M(x)$ represent the number of accepting computations of a nondeterministic Turing machine M on input x. A language L is in $\oplus\mathrm{P}$ if there exists a polynomial time bounded nondeterministic Turing machine M such that for all x

- $x \in L \Rightarrow \#M(x)$ is odd;
- $x \notin L \Rightarrow \#M(x)$ is even.

Let g be any function. We say that advice function f is g-bounded if for all n it holds that $|f(n)| \leq g(n)$. In this paper we will be interested only in functions g that are polynomial.

The notation $\leq_T^{sn}$ is used for *strong nondeterministic Turing reductions*, which are defined by $A \leq_T^{sn} B$ iff $A \in \mathrm{NP}^B \cup \mathrm{CoNP}^B$.

3. Distinguishing complexity for derandomization. In this section we prove hardness of R_t^{CD} and R_t^{CND} for AM and MA games, respectively, under NP-reductions.

THEOREM 3.1. *For any t with $t(n) \in \omega(n \log n)$, $\mathrm{MA} \subseteq \mathrm{NP}^{\mathrm{R}_t^{CD}}$.*

THEOREM 3.2. *For any t with $t(n) \in \omega(n)$, $\mathrm{AM} \subseteq \mathrm{NP}^{\mathrm{R}_t^{CND}}$.*

The proof of both theorems is roughly as follows. First guess a string of high CD^{poly} complexity, respectively, CND^{poly} complexity. Next, we use the nondeterministic reductions once more to play the role of Merlin and use the random string to derandomize Arthur. Note that this is not as straightforward as it might look. The

randomness used by Arthur in interactive protocols is used for hiding and cannot in general be substituted by computational randomness.

The idea of using strings of high CD complexity and Zuckerman's sampler derandomization stems from [7, section 8], which is the full version of [6]. Though they do not explicitly define the set R_t^{CD}, they use the same approach to derandomize BPP computations there.

The proof needs a string of high CD^p, respectively, CND^p complexity for p some polynomial. We first show that we can nondeterministically extract such a string from a longer string with high CD^t complexity (respectively, CND^t complexity) for any fixed t with $t(n) \in \omega(n \log n)$.

LEMMA 3.3. *Let f be such that $f(n) < n$, and let t, t', and T be such that $T(n) = (t'(f(n)) + n - f(n))$, $\lim_{n\mapsto\infty} \frac{T(n)\log T(n)}{t(n)} = 0$. Then for all sufficiently large s with $CD^t(s) > |s|$, it holds that $CD^{t'}(s[1..f(|s|)]) \geq f(|s|) - 2\log|f(|s|)| - O(1)$.*

Proof. Suppose for a contradiction that for any constant d_0 and infinitely many s with $CD^t(s) \geq |s|$, it holds that $CD^{t'}(s[1..f(|s|)]) < f(|s|) - 2\log|f(|s|)| - d_0$. Then for any such s there exists a program p_s that runs in $t'(f(|s|))$ and recognizes only $s[1..f(|s|)]$ where $|p_s| < f(|s|) - 2\log|f(|s|)| - d_0$. The following program then recognizes s and no other string.

> Input y.
> Check that the first $f(|s|)$ bits of y equal $s[1..f(|s|)]$, using p_s. (Assume $|f(|s|)|$ is stored in the program for a cost of $\log|f(|s|)|$ bits.)
> Check that the last $|s| - f(|s|)$ bits of y equal $s[f(|s|) + 1..|s|]$. (These bits are also stored in the program.)

This program runs in time $T(|s|) = t'(f(|s|)) + |s| - f(|s|)$. Therefore it takes at most $t(|s|)$ steps on U for all sufficiently large s [16]. We lose the $\log n$ factor here because our algorithm must run on a fixed machine and the simulation is deterministic.

The program's length is $|p_s| + |s| - f(|s|) + \log|f(|s|)| + d_1 < f(|s|) - 2\log|f(|s|)| - d_0 + |s| - f(|s|) + \log|f(|s|)| + d_1$, which is less than $|s|$ for almost all s. Hence $CD^t(s) < |s|$, which contradicts the assumption. □

COROLLARY 3.4. *For every polynomial n^c, $t \in \omega(n \log n)$ and sufficiently large string s with $CD^t(s) \geq |s|$, if $m = |s|^{\frac{1}{c}}$ and $s' = s[1..m]$, then $CD^{n^c}(s') \geq |s'| - 2\log|s'| - O(1)$.*

Proof. Take $t'(n) = n^c$, $f(n) = n^{\frac{1}{c}}$ and apply Lemma 3.3. □

Lemma 3.3 and Corollary 3.4 have the following nondeterministic analogon.

LEMMA 3.5. *For every polynomial n^c, $t \in \omega(n)$ and sufficiently large string s with $CND^t(s) \geq |s|$, if $m = |s|^{\frac{1}{c}}$ and $s' = s[1..m]$, then $CND^{n^c}(s') \geq |s'| - 2\log|s'| - O(1)$.*

Proof. The same proof applies, with a lemma similar to Lemma 3.3. However, in the nondeterministic case the simulation costs only linear time [5]. □

Before we can proceed with the proof of the theorems, we also need some earlier results. We first need the following theorem from Zuckerman.

THEOREM 3.6 (see [29]). *There is a constant c such that for $\gamma = \gamma(m)$, $\epsilon = \epsilon(m)$, and $\alpha = \alpha(m)$ with $m^{-1/2\log^* m} \leq \alpha \leq 1/2$ and $\epsilon \geq \exp(-\alpha^{2\log^* m} m)$, there exists a universal $(r, d, m, \epsilon, \gamma)$-oblivious sampler which runs in polynomial time and uses only $r = (1 + \alpha)(m + \log\gamma^{-1})$ random bits and outputs $d = ((m + log\gamma^{-1})/\epsilon)^{c_\alpha}$ sample points, where $c_\alpha = c(\log\alpha^{-1})/\alpha$.*

We also need the following lemma by Buhrman and Fortnow.

LEMMA 3.7 (see [6]). *Let A be a set in* P. *For each string $x \in A^{=n}$ it holds that $CD^p(x) \leq 2\log(\|A^{=n}\|) + O(\log n)$ for some polynomial p.*

As noted in [6], an analogous lemma holds for CND^p and NP.

LEMMA 3.8 (see [6]). *Let A be a set in* NP. *For each string $x \in A^{=n}$ it holds that $CND^p(x) \leq 2\log(\|A^{=n}\|) + O(\log n)$ for some polynomial p.*

From these results we can prove the theorems. If we want to prove, for Theorem 3.1, that an NP machine oracle with oracle R_t^{CD} can recognize a set A in MA, then the positive side of the proof is easy: If x is in A, then there exists a machine M and a string y such that a 2/3 fraction of the strings r of length $|x|^c$ makes $M(x, y, r)$ accept. So an NP machine can certainly guess one such pair x, y as a "proof" for $x \in A$. The negative side is harder. We will show that if $x \notin A$ and we substitute for r a string of high enough CD complexity (CND complexity for Theorem 3.2), then *no* y can make $M(x, y, r)$ accept.

To grasp the intuition behind the proof let us look at the much simplified example of a BPP machine M having a 1/3 error probability on input x and a string r of maximal unbounded Kolmogorov complexity. There are $2^{|x|^c}$ possible computations on input x, where $|x|^c$ is the runtime of M. Suppose that M must accept x. Then at most a 1/3 fraction, i.e., at most $2^{|x|^c}/3$ of these computations reject x. Each rejecting computation consists of a deterministic part described by M and x and a set of $|x|^c$ coin flips. Identify such a set of coin flips with a binary string and we have that each rejecting computation uniquely identifies a string of length $|x|^c$. Call this set B. We would like to show by contradiction that a random string cannot be a member of this set, and hence that any random string, used as a sequence of coin flips, leads to a correct result. Any string in B is described by M, x, and an index in B, which has length $\log \|B\| \leq |x|^c - \log 3$. So far there are no grounds for a contradiction since a description consisting of these elements can have length greater than $|x|^c$. However, we can *amplify* the computation of M on input x by repetition and taking majority. Suppose we repeat the computation $|x|^2$ times. This will increase the number of incorrect computations to (at most) $(\frac{8}{9})^{|x|^2/2} 2^{|x|^{c+2}}$. An index in this set has length $|x|^{c+2} - (|x|^2/2)\log\frac{9}{8}$. However, $|x| + |x|^{c+2} - (|x|^2/2)\log\frac{9}{8}$ cannot describe a random string of length $|x|^{c+2}$, which is the length of such a computation.

Unfortunately, in our case the situation is a bit more complicated. The factor 2 in Lemma 3.7 renders standard amplifaction of randomized computation useless. Fortunately, Theorem 3.6 allows for a different type of amplification using much less random bits, so that the same type of argument *can* be used. We will now proceed to show how to fit the amplification given by Theorem 3.6 to our situation.

LEMMA 3.9.

1. *Let L be a language in* MA. *For any constant k and any constant $0 < \alpha \leq \frac{1}{2}$, there exists a deterministic polynomial time bounded machine M such that*

(a) $x \in L \implies \exists^m y \Pr[M(x, y, r) = 1] = 1$;

(b) $x \notin L \implies \forall^m y \Pr[M(x, y, r) = 1] < 2^{-km}$,

where $m = |x|^c$ and r is chosen uniformly at random in $\{0,1\}^{(1+\alpha)(1+k)m}$.

2. *Let L be a language in* AM. *For any constant k and any constant $0 < \alpha \leq \frac{1}{2}$, there exists a deterministic polynomial time bounded machine M such that*

(a) $x \in L \implies \Pr[\exists y M(x, y, r) = 1] = 1$;

(b) $x \notin L \implies \Pr[\exists y M(x, y, r) = 1] < 2^{-km}$,

where $m = |x|^c$ and r is chosen uniformly at random in $\{0,1\}^{(1+\alpha)(1+k)m}$.

Proof.

1. Fürer et al. showed that the fraction 2/3 (see Definition 2.2) can be replaced by 1 in [13]. Now let M_L be the deterministic polynomial time machine corresponding to L in Definition 2.2, adapted so that it can accept with probability 1 if $x \in L$. Assume

M_L runs in time n^c (where $n = |x|$). This means that for M_L the $\exists y$ and $\forall y$ in the definition can be assumed to be $\exists^{n^c} y$ and $\forall^{n^c} y$, respectively. Also, the random string may be assumed to be drawn uniformly at random from $\{0,1\}^{n^c}$.

To obtain the value 2^{-km} in the second item, we use Theorem 3.6 with $\gamma = 2^{-km}$, and $\epsilon = 1/6$. For given x and y let f_{xy} be the FP function that on input z computes $M_L(x,y,z)$. If $|y| = |z| = n^c = m$, then $f_{xy} : \{0,1\}^m \mapsto [0,1]$. We use the oblivious sampler to get a good estimate for Ef_{xy}. That is, we feed a random string of length $(1+\alpha)(1+k)m$ in the oblivious sampler and it returns $d = ((1+k)m/\epsilon)^{c_\alpha}$ sample points $z_1, \ldots, z_d$ on which we compute $\frac{1}{d}\sum_{i=1}^{d} f_{xy}(z_i)$. M is the machine that computes this sum on input x, y, and r and accepts iff its value is greater than $1/2$.

If $x \in L$, there is a y such that $\Pr[M_L(x,y,r) = 1] = 1$. This means $\frac{1}{d}\sum_{i=1}^{d} f_{xy}(z_i) = 1$ no matter which sample points are returned by the oblivious sampler. If $x \notin L$, then $(\forall y)[Ef_{xy} < 1/3]$. With probability $1-\gamma$ the sample points returned by the oblivious sampler are such that $|\frac{1}{d}\sum_{i=1}^{d} f_{xy}(z_i) - Ef_{xy}| \leq \epsilon$, so $\frac{1}{d}\sum_{i=1}^{d} f_{xy}(z_i) > \frac{1}{2}$ with probability $\leq 2^{-km}$.

2. The proof is analogous to the proof of part 1. We just explain the differences. For the 1 in the first item of the claim we can again refer to [13]. In this part M_L is the deterministic polynomial time machine corresponding to the AM-language L and we define the function $f_x : \{0,1\}^m \mapsto [0,1]$ as the function that on input z computes $\exists^{n^c} y M_L(x,y,z) = 1$. Now f_x is an $\mathrm{FP}^{\mathrm{NP}}$ computable function. The sample points $z_1, \ldots, z_d$ that are returned in this case have the following properties. If $x \in L$, then $f_x(z_i) = 1$ no matter which string is returned as z_i. That is, for every possible sample point z_i, there is a y_i such that $M_L(x,y_i,z_i) = 1$. So for any set of sample points $z_1, \ldots, z_d$ that the sampler may return, there exists a $y = \langle y_1, \ldots, y_d \rangle$ such that $M_L(x,y_i,z_i) = 1\ \forall i$. If $x \notin L$, then $f_x(z_i) = 1$ for less than half of the sample points with probability $1-\gamma$. That is,

$$\Pr\left[(\exists y = y_1 \ldots y_d)\left[\frac{1}{d}\sum_{i=1}^{d} M_L(x,y_i,z_i) > \frac{1}{2}\right]\right]$$

is less than 2^{-km}. So if we let $M(x,y,r)$ be the deterministic polynomial time machine that uses r to generate d sample points and then interprets y as $\langle y_1, \ldots, y_d \rangle$ and counts the number of accepts of $M_L(x,y_i,z_i)$ and accepts if this number is greater than $\frac{1}{2}d$, we get exactly the desired result. □

In the next lemma we show that a string of high enough CD^{poly} (CND^{poly}) can be used to *derandomize* an MA (AM) protocol.

LEMMA 3.10.

1. *Let L be a language in* MA *and $0 < \epsilon \leq 1$. There exists a deterministic polynomial time bounded machine M, a polynomial q, $\alpha > 0$, and integers k and c such that for almost all n and every r with $|r| = (1+\alpha)(1+k)n^c$ and $CD^q(r) \geq \epsilon|r|$, $\forall^{=n} x[x \in L \iff \exists y M(x,y,r) = 1]$.*

2. *Let L be a language in* AM *and $0 < \epsilon \leq 1$. There exists a deterministic polynomial time bounded machine M a polynomial q, $\alpha > 0$, and integers k and c such that for almost all n and every r with $|r| = (1+\alpha)(1+k)n^c$ and $CND^q(r) \geq \epsilon|r|$, $\forall^{=n} x[x \in L \iff \exists y M(x,y,r) = 1]$.*

Proof.

1. Choose $\alpha < \frac{\epsilon}{2}$ and $k > \frac{6}{\epsilon - 2\alpha}$. Let M be the deterministic polynomial time bounded machine corresponding to L, k, and α of Lemma 3.9(1). The polynomial n^c will be the time bound of the machine witnessing $L \in$ MA of that same lemma. We

will determine q later, but assume for now that r is a string of length $(1+\alpha)(1+k)n^c$ such that $CD^q(r) \geq \epsilon|r|$, and for ease of notation set $m = n^c$.

Suppose $x \in L$. Then it follows that there exists a y such that for all s of length $(1+\alpha)(1+k)n^c$, $M(x,y,s) = 1$. So in particular it holds that $M(x,y,r) = 1$.

Suppose $x \notin L$. We have to show that $(\forall y)[M(x,y,r) = 0]$. Suppose that this is not true and let y_0 be such that $M(x,y_0,r) = 1$. Define

$$A_{x,y_0} = \{s : M(x,y_0,s) = 1\}.$$

It follows that $A_{x,y_0} \in \mathrm{P}$ by essentially a program that simulates M and has x and y_0 hardwired. (Although A_{x,y_0} is finite and therefore trivially in P it is crucial here that the size of the polynomial program is roughly $|M| + |x| + |y_0|$.) Because of the amplification of the MA protocol we have that

$$\|A_{x,y_0}\| \leq 2^{(1+\alpha)(1+k)m - km}.$$

Since $r \in A_{x,y_0}$ it follows by Lemma 3.7 that there is a polynomial p such that

$$\begin{aligned} CD^p(r) &\leq 2[(1+\alpha)(1+k)m - km] + |x| \\ &\quad + |y_0| + O(\log m) \\ &\leq 2\alpha m + 2\alpha km + 5m. \end{aligned}$$

On the other hand, we chose r such that

$$\begin{aligned} CD^q(r) &\geq \epsilon|r| \\ &= (1+\alpha)(1+k)m\epsilon \\ &> 2\alpha m + 2\alpha km + 5m, \end{aligned}$$

which gives a contradiction for $q \geq p$.

2. Choose $\alpha < \frac{\epsilon}{2}$ and $k > \frac{5}{\epsilon - 2\alpha}$. Let M be the deterministic polynomial time bounded machine corresponding to L, α, and k of Lemma 3.9(2). Again, n^c will be the time bound of the machine now witnessing $L \in \mathrm{AM}$, $m = n^c$, and q will be determined later. Assume for now that r is a string of length $(1+\alpha)(1+k)n^c$ such that $CND^q(r) \geq \epsilon|r|$. Suppose $x \in L$. Then it follows that for all s there exists a y such that $M(x,y,s) = 1$. So in particular there is a y_r such that $M(x,y_r,r) = 1$. Suppose $x \notin L$. We have to show that $\forall y M(x,y,r) = 0$. Suppose that this is not true. Define $A_x = \{s : \exists y M(x,y,s) = 1\}$. Then $A_x \in \mathrm{NP}$ by a program that has x hardwired, guesses a y, and simulates M. Because of the amplification of the AM protocol we have that $\|A_x\| \leq 2^{(1+\alpha)(1+k)m - km}$. Since $r \in A_x$ it follows by Lemma 3.8 that there exists a polynomial p such that

$$\begin{aligned} CND^p(r) &\leq 2[(1+\alpha)(1+k)m - km] + |x| + O(\log m) \\ &\leq 2\alpha m + 2\alpha km + 4m. \end{aligned}$$

On the other hand, we chose r such that

$$\begin{aligned} CND^q(r) &\geq \epsilon|r| \\ &= (1+\alpha)(1+k)m\epsilon \\ &> 2\alpha m + 2\alpha km + 4m, \end{aligned}$$

which gives a contradiction whenever $q \geq p$. □

The following corollary shows that a string of high enough CD^{poly} complexity can be used to derandomize a BPP machine (see also [7, Theorem 8.2]).

COROLLARY 3.11. *Let A be a set in* BPP. *For any $\epsilon > 0$ there exists a polynomial time Turing machine M and a polynomial q such that if $CD^q(r) \geq \epsilon|r|$ with $|r| = q(n)$, then for all x of length n it holds that $x \in A \iff M(x, r) = 1$.*

Proof of Theorem 3.1. Let A be a language in MA. Let q, M, and $q'(n) = (1+\alpha)(1+k)q(n)$ be as in Lemma 3.10(1). The nondeterministic reduction behaves as follows on input x of length n. First guess an s of size $q(q'(n))$ and check that $s \in \mathrm{R}_t^{CD}$. Set $r = s[1..q'(n)]$ and accept iff there exists a y such that $M(x, y, r) = 1$. By Corollary 3.4 it follows that $CD^q(r) \geq |r|/2$ and the correctness of the reductions follows directly from Lemma 3.10(1) with $\epsilon = 1/2$. □

Proof of Theorem 3.2. This follows directly from Lemma 3.10(2). The NP-algorithm is analogous to the one above. □

COROLLARY 3.12. *For $t \in \omega(n \log n)$*

1. BPP *and* $\mathrm{NP}^{\mathrm{BPP}}$ *are included in* $\mathrm{NP}^{\mathrm{R}_t^{CD}}$;
2. $\overline{GI} \in \mathrm{NP}^{\mathrm{R}_t^{CND}}$.

It follows that if $\mathrm{R}_t^{CND} \in \mathrm{NP} \cap \mathrm{coNP}$, then the graph isomorphism (GI) problem is in $\mathrm{NP} \cap \mathrm{coNP}$.

4. Limitations. In the previous section we showed that the set R_t^{CD} is hard for MA under NP reductions. One might wonder whether R_t^{CD} is also hard for MA under a stronger reduction like the deterministic polynomial time Turing reduction. In this section we show that, if true, this will need a nonrelativizing proof. We will derive the following theorem.

THEOREM 4.1. *There is a relativized world where for every polynomial t and $0 < \epsilon \leq 1$,* $\mathrm{BPP} \not\subset \mathrm{P}^{\mathrm{R}_{t,\epsilon}^{CD}}$.

The proof of this theorem is given in Lemma 4.2, which says that the statement of Theorem 4.1 is true in any world where $\mathrm{P}^A = \oplus\mathrm{P}^A$ and $\mathrm{EXP}^{\mathrm{NP}^A} \subseteq \mathrm{NP}^A/poly$, and in Theorem 4.3, which precisely shows the existence of such a world.

LEMMA 4.2. *For any oracle A and $0 < \epsilon \leq 1$ it holds that if* $\mathrm{EXP}^{\mathrm{NP}^A} \subseteq \mathrm{NP}^A/poly$ *and* $\oplus\mathrm{P}^A = \mathrm{P}^A$, *then* $\mathrm{BPP}^A \not\subset P^{\mathrm{R}_{t,\epsilon}^{CD\,A}}$.

Proof. Suppose for a contradiction that the lemma is not true. If $\mathrm{EXP}^{\mathrm{NP}} \subseteq \mathrm{NP}/poly$, then $\mathrm{EXP} \subseteq \mathrm{NP}/poly$, so $\mathrm{EXP} \subseteq \mathrm{PH}$ [27]. Furthermore, if $\mathrm{EXP}^{\mathrm{NP}} \subseteq \mathrm{NP}/poly$, then certainly $\mathrm{EXP}^{\mathrm{NP}} \subseteq \mathrm{EXP}/poly$. It then follows from [9] that $\mathrm{EXP}^{\mathrm{NP}} = \mathrm{EXP}$, so $\mathrm{EXP}^{\mathrm{NP}} \subseteq \mathrm{PH}$.

If $\oplus\mathrm{P} = \mathrm{P}$, then unique-SAT (see [8] for a definition) is in P. Then $\mathrm{NP} = \mathrm{R}$ by [26] and so $\mathrm{NP} \subseteq \mathrm{BPP}$ which implies $\mathrm{PH} \subseteq \mathrm{BPP}$ by [28].

Finally, the fact that unique-SAT is in P is equivalent to the following: for all x and y, $C^{poly}(x|y) \leq CD^{poly}(x|y) + O(1)$, as shown in [12]. We can use the proof of [12] to show that unique-SAT in P also implies that $\mathrm{R}_{t,\epsilon}^{CD} \in \mathrm{coNP}$ for a particular universal machine. (Note that we need only contradict the assumption for one particular type of universal machine.) This then in its turn implies by assumption that BPP and hence $\mathrm{EXP}^{\mathrm{NP}}$ are in P^{NP}. This, however, contradicts the hierarchy theorem for relativized Turing machines [15]. As all parts of this proof relativize, we get the result for any oracle. There's one caveat here. Though $\mathrm{R}_{t,\epsilon}^{CD^A}$ clearly has a meaningful interpretation, to talk about $\mathrm{P}^{\mathrm{R}_{t,\epsilon}^{CD\,A}}$ one must of course allow P to have access to the oracle. It is not clear that P can ask any question if the machine can only ask a question about the random strings. Therefore, one might argue that $\mathrm{P}^{\mathrm{R}_{t,\epsilon}^{CD\,A} \oplus A}$ should

actually be in the statement of the lemma. This does not affect the proof.

Our universal machine, say, U_S, is the following. On input p, x, y, U_S uses the Cook–Levin reduction to produce a formula f on $|x|$ variables with the property that x satisfies f iff p accepts x. Then U_S uses the self-reducibility of f and the assumed polynomial time algorithm for unique-SAT to make acceptance of x unique. That is, first if the number of variables is not equal $|y|$, it rejects. Then, using the well-known substitute and reduce algorithm for SAT, it verifies for $i = 1, \ldots, |x|$ and assignments $x_j = v_j$ successively obtained from the algorithm that the algorithm for unique-SAT precisely accepts $f(v_1 \ldots v_i)$ or rejects if this algorithm accepts both $f(v_1 \ldots v_i)$ and $f(v_1 \ldots (1 - v_i))$. Using this universal machine every program accepts at most one string and therefore $\mathrm{R}^{CD}_{t,\epsilon} \in$ coNP via an obvious predicate. As argued above, this gives us our contradiction. □

Now we proceed to construct the oracle.

THEOREM 4.3. *There exists an oracle A such that* $\mathrm{EXP}^{\mathrm{NP}^A} \subset \mathrm{NP}^A/poly \wedge \oplus\mathrm{P}^A = \mathrm{P}^A$.

Proof. The proof parallels the construction from Beigel, Buhrman, and Fortnow [4], who construct an oracle such that $\mathrm{P}^A = \oplus\mathrm{P}^A$ and $\mathrm{NEXP}^A = \mathrm{NP}^A$. We will use a similar setup.

Let M^A be a nondeterministic *linear* time Turing machine such that the language L^A defined by

$$w \in L^A \Leftrightarrow \#M^A(w) \bmod 2 = 1$$

is $\oplus P^A$ complete for every A.

For every oracle A, let K^A be the linear time computable complete set for NP^A. Let N^{K^A} be a deterministic machine that runs in time 2^n and for every A accepts a language H^A that is complete for $\mathrm{EXP}^{\mathrm{NP}^A}$. We will construct A such that there exists a n^2 bounded advice function f such that for all w

$$\begin{array}{lcll} w \in L^A & \Leftrightarrow & \langle 0, w, 1^{|w|^2}\rangle \in A & \text{(Condition 0),} \\ w \in H^A & \Leftrightarrow & \exists v\ |v| = |w|^2 \text{ and} & \\ & & \langle 1, f(|w|), w, v\rangle \in A & \text{(Condition 1).} \end{array}$$

Condition 0 will guarantee that $\mathrm{P} = \oplus\mathrm{P}$, and Condition 1 will guarantee that $\mathrm{EXP}^{\mathrm{NP}} \subset \mathrm{NP/poly}$.

We use the term 0-strings for the strings of the form $\langle 0, w, 1^{|w|^2}\rangle$ and 1-strings for the strings of the form $\langle 1, z, w, v\rangle$ with $|z| = |v| = |w|^2$. All other strings we immediately put in $\overline{A}$.

First we give some intuition for the proof. M is a linear time Turing machine. Therefore setting the 1-strings forces the setting of the 0-strings. Condition 0 will be automatically fulfilled by just describing how we set the 1-strings because they force the 0-strings as defined by Condition 0.

Fulfilling Condition 1 requires a bit more care since $N^{K^A}(x)$ can query exponentially long and *double* exponentially many 0- and 1-strings. We consider each 1-string $\langle 1, z, w, v\rangle$ as a 0-1 valued variable $y_{\langle z,w,v\rangle}$ whose value determines whether $\langle 1, z, w, v\rangle$ is in A. The construction of A will force a 1-1 correspondence between the computation of $N^{K^A}(x)$ and a low-degree polynomial over variables with values in $GF[2]$. To encode the computation properly we use the fact that the OR function has high degree.

We will assign a polynomial p_z over GF[2] to all of the 0-strings and 1-strings z. We ensure that for all z

1. if $p_z = 1$, then z is in A;
2. if $p_z = 0$, then z is not in A.

First for each 1-string $z = \langle 1, z, w, v\rangle$ we let p_z be the single variable polynomial $y_{\langle z,w,v\rangle}$.

We assign polynomials to the 0-strings recursively. Note that $M^A(x)$ can only query 0-strings with $|w| \leq \sqrt{|x|}$. Consider an accepting computation path π of $M(x)$ (assuming the oracle queries are guessed correctly). Let $q_{\pi,1}, \ldots, q_{\pi,m}$ be the queries on this path and $b_{\pi,1}, \ldots, b_{\pi,m}$ be the query answers with $b_{\pi,i} = 1$ if the query was guessed in A, and $b_{\pi,i} = 0$ otherwise. Note that $m \leq n = |x|$.

Let $\mathcal{P}$ be the set of accepting computation paths of $M(x)$. We then define the polynomial p_z for $z = \langle 0, x, 1^{|x|^2}\rangle$ as follows:

$$p_z = \sum_{\pi\in\mathcal{P}} \prod_{1\leq i\leq m} (p_{q_{\pi,i}} + b_{\pi,i} + 1). \tag{1}$$

Remember that we are working over GF[2] so addition is parity.

Setting the variables $y_{\langle z,w,v\rangle}$ (and thus the 1-strings) forces the values of p_z for the 0-strings. We have set things up properly so the following lemma is straightforward.

LEMMA 4.4. *For each* 0*-string* $z = \langle 0, x, 1^{|x|^2}\rangle$ *we have* $p_z = \#M^A(x) \bmod 2$ *and Condition* 0 *can be satisfied. The polynomial* p_z *has degree at most* $|x|^2$.

Proof. The proof is simple by induction on $|x|$. □

The construction will be done in stages. At stage n we will code all the strings of length n of H^A into A setting some of the 1-strings and automatically the 0-strings and thus fulfilling both Conditions 0 and 1 for this stage.

We will need to know the degree of the multivariate multilinear polynomials representing the *OR* and the *AND* function.

LEMMA 4.5. *The representation of the function* $OR(u_1, \ldots, u_m)$ *and the function* $AND(u_1, \ldots, u_m)$ *as multivariate multilinear polynomials over* GF[2] *requires degree exactly* m.

Proof. Every function over GF[2] has a unique representation as a multivariate multilinear polynomial.

Note that *AND* is just the product and by using De Morgan's laws we can write *OR* as

$$OR(u_1, \ldots, u_m) = 1 + \prod_{1\leq i\leq m} (1 + u_i). \qquad \square$$

The construction of the oracle now treats all strings of length n in lexicographic order. First, in a *forcing phase* in which the oracle is set so that all computations of N^{K^A} remain fixed for future extensions of the oracle, and then in a *coding phase* in which first an advice string is picked and then the computations just forced are coded in the oracle in such a way that they can be retrieved by an NP machine with this advice string. Great care has of course to be taken so that the two phases don't disturb each other and do not disturb earlier stages of the construction.

We first describe the *forcing phase*. Without loss of generality, we will assume that machine N queries only strings of the form $q \in K^A$. Note that since N runs in time 2^n it may query exponentially long strings to K^A.

Let x_1 be the first string of length n. When we examine the computation of $N(x_1)$ we encounter the first query q_1 to K^A. We will try to extend the oracle A to $A' \supseteq A$ such that $q_1 \in K^{A'}$. If such an extension does not exist we may assume that q_1 will

never be in K^A no matter how we extend A in the future. We must, however, take care that we will not disturb previous queries that were forced to be in K^A. To this end we will build a set S containing all the previously encountered queries that were forced to be in K^A. We will only extend A such that $\forall q \in S$ it holds that $q \in K^{A'}$. We will call such an extension an *S-consistent* extension of A.

Returning to the computation of $N(x_1)$ and q_1 we ask whether there is an S-consistent extension of A such that $q_1 \in K^{A'}$. If such an extension exists, we will choose the S-consistent extension of A which adds a minimal number of strings to A and puts q_1 in S. Next we continue the computation of $N^{K^A}(x_1)$ with q_1 answered yes, and otherwise we continue with q_1 answered no. The next lemma shows that a minimal extension of A will never add more than 2^{3n} strings to A.

LEMMA 4.6. *Let S be as above and q be any query to K^A and suppose we are in stage n. If there exists an S-consistent extension of A such that $q \in K^{A'}$, then there exists one that adds at most 2^{3n} strings to A.*

Proof. Let M_K be a machine that accepts K^A when given oracle A and consider the computation of machine $M_K^A(q)$. Let $o_1, \ldots, o_l$ be the smallest set of strings such that adding them to A is an S-consistent extension of A such that $M_K^{A'}(q)$ accepts. ($A' = A \cup \{o_1, \ldots, o_l\}$.) Consider the leftmost accepting path of $M_K^{A'}(q)$ and let $q_1, \ldots, q_{2^n}$ be the queries (both 0- and 1-queries) on that path. Moreover let b_i be 1 iff $q_i \in A'$. Define for q the following polynomial:

$$P_q = \prod_{1 \le i \le 2^n} (p_{q_i} + b_i + 1). \tag{2}$$

After adding the strings $o_1, \ldots, o_l$ to A we have that $P_q = 1$. Moreover by Lemma 4.4 the degree of each p_{q_i} is at most 2^{2n} and hence the degree of P_q is at most 2^{3n}. Now consider what happens when we take out any number of the strings $o_1, \ldots, o_l$ of A' resulting in A''. Since this was a minimal extension of A it follows that $M_K^{A''}(q)$ rejects and that $P_q = 0$. So P_q computes the *AND* on the l strings $o_1, \ldots, o_l$. Since by Lemma 4.5 the degree of the unique multivariate multilinear polynomial that computes the *AND* over l variables over GF[2] is l, it follows that $l \le 2^{3n}$. □

After we have dealt with all the queries encountered on $N^{K^A}(x_1)$ we continue this process with the other strings of length n in lexicographic order. Note that since we only extend A S-consistently we will never disturb any computation of N^{K^A} on lexicographic smaller strings. This follows since the queries that are forced to be yes will remain yes, and the queries that could not be forced with an S-consistent extension will never be forced by any S'-consistent extension of A for $S \subset S'$. After we have finished this process we have to code all the computations of N on the strings of length n. It is easy to see that $\|S\| \le 2^{2n}$ and that at this point by Lemma 4.6 at most 2^{5n} strings have been added to A at this stage. Closing the *forcing phase* we can now pick an advice string and proceed to the *coding phase*. A standard counting argument shows that there is a string z of length n^2 such that no strings of the form $\langle 1, z, w, v\rangle$ have been added to A. This string z will be the advice for strings of length n.

Now we have to show that we can code every string x of length n correctly in A to fulfill Condition 1. We will do this in lexicographic order. Suppose we have coded all strings x_j (for $j < i$) correctly and that we want to code x_i. There are two cases.

Case 1. $N^{K^A}(x_i) = 0$. In this case we put all the strings $\langle 1, z, x_i, w\rangle$ in $\overline{A}$ and thus set all these variables to 0. Since this does not change the oracle it is an S-consistent extension.

Case 2. $N^{K^A}(x_i) = 1$. We properly extend A S-consistently adding only strings of the form $\langle 1, z, x_i, w\rangle$ to A. The following lemma shows that this can always be done. A *proper* extension of A is one that adds one or more strings to A.

LEMMA 4.7. *Let* $\|S\| \leq 2^{2n}$ *be as above. Suppose that* $N^{K^A}(x_i) = 1$. *There exists a proper* S*-consistent extension of* A *adding only strings of the form* $\langle 1, z, x_i, w\rangle$ *with* $|w| = n^2$.

Proof. Suppose that no such proper S-consistent extension of A exists. Consider the following polynomial:

$$Q_{x_i} = 1 - \prod_{q \in S}(P_q), \tag{3}$$

where P_q is defined as in Lemma 4.6, equation (2). Initially $Q_{x_i} = 0$ and the degree of $Q_{x_i} \leq 2^{5n}$. Since every extension of A with strings of the form $\langle 1, z, x_i, w\rangle$ is not S-consistent it follows that Q_{x_i} computes the OR of the variables $y_{\langle z, x_i, w\rangle}$. Since there are 2^{n^2} many of those variables we have by Lemma 4.5 a contradiction with the degree of Q_{x_i}. Hence there exists a proper S-consistent extension of A adding only strings of the form $\langle 1, z, x_i, w\rangle$, and x_i is properly coded into A. □

Stage n ends after coding all the strings of length n.

This completes the proof of Theorem 4.3. □

Theorem 4.3 together with the proof of Lemma 4.2 also gives the following corollary.

COROLLARY 4.8. *There exists a relativized world where* $\mathrm{EXP}^{\mathrm{NP}}$ *is in* BPP *and* $\oplus\mathrm{P} = \mathrm{P}$.

Our oracle also extends the oracle of Ko [17] to CD^{poly} complexity as follows.

COROLLARY 4.9. *There exists an oracle such that* $\overline{\mathrm{R}^{CD}_{t,\epsilon}}$ *for any* $t \in \omega(n\log(n))$ *and* $\epsilon > 0$ *is complete for* NP *under strong nondeterministic reductions and* $\mathrm{P}^{\mathrm{NP}} \neq \Sigma^p_2$.

Proof. The relativized world constructed in the proof of Theorem 4.3 is a world where coNP $\subseteq$ BPP and $C^{poly}(x|y) = CD^{poly}(x|y) + O(1)$. Hence it follows that $\overline{\mathrm{R}^{CD}_{t,\epsilon}} \in$ NP. Moreover Corollary 3.12 relativizes so by item 1 we have that BPP $\subseteq \mathrm{NP}^{\overline{\mathrm{R}^{CD}_{t,\epsilon}}}$. □

As a by-product our oracle shows the following.

COROLLARY 4.10. $\exists A$ *unique-SAT* $^A \in \mathrm{P}^A$ *and* $\mathrm{P}^{\mathrm{NP}^A} \neq \Sigma^{p,A}_2$.

This corollary indicates that the current proof that shows that if unique-SAT $\in$ P, then PH $= \Sigma^p_2$ cannot be improved to yield a collapse to P^{NP} using relativizing techniques.

5. PSPACE and cR^{CS}_s. In this section we further study the connection between cR^{CS}_s and interactive proofs. So far we have established that strings that have sufficiently high CND^{poly} complexity can be used to derandomize an IP protocol that has two rounds in such a way that the role of both the prover and the verifier can be played by an NP oracle machine. Here we will see that this is also true for IP itself provided that the random strings have high enough space bounded Kolmogorov complexity. The set of QBFs is defined as the closure of the set of boolean variables x_i and their negations $\overline{x_i}$ under the operations $\wedge$, $\vee$, $\forall x_i$, and $\exists x_i$. A QBF in which all the variables are quantified is called *closed*. Other QBFs are called open. We need the following definitions and theorems from [24].

DEFINITION 5.1 (see [24]). *A QBF B is called* simple *if in the given syntactic representation every occurrence of each variable is separated from its point of quantification by at most one universal quantifier (and arbitrarily many other symbols).*

For technical reasons we also assume that (simple) QBFs can contain negated variables, but no other negations. This is no loss of generality since negations can be pushed all the way down to variables.

DEFINITION 5.2 (see [24]). *The arithmetization of a (simple) QBF B is an arithmetic expression obtained from B by replacing every positive occurrence of x_i by variable z_i, every negated occurrence of x_i by $(1 - z_i)$, every $\wedge$ by $\times$, every $\vee$ by $+$, every $\forall x_i$ by $\prod_{z_i \in \{0,1\}}$, and every $\exists x_i$ by $\sum_{z_i \in \{0,1\}}$.*

It follows that the arithmetization of a (simple) QBF in closed form has an integer value, whereas the arithmetization of an open QBF is equivalent to a (possibly multivariate) function.

DEFINITION 5.3 (see [24]). *The functional form of a simple closed QBF is the univariate function that is obtained by removing from the arithmetization of B either $\sum_{z_i \in \{0,1\}}$ or $\prod_{z_i \in \{0,1\}}$ where i is the least index of a variable for which this is possible.*

Notation. Let B be a (simple) QBF with quantifiers $Q_1, \ldots, Q_k$. For $i \leq k$ we let $*_i = +$ if $Q_i = \exists$ and $*_i = \times$ if $Q_i = \forall$. Let B be a QBF. Let B' be the boolean formula obtained from B by removing all its quantifiers. We denote by $\tilde{B}$ the arithmetization of B'. It is well known that the language of all true QBFs is complete for PSPACE. The restriction of true QBFs to simple QBFs remains complete.

THEOREM 5.4 (see [24]). *The language of all closed simple true QBFs is complete for* PSPACE *(under polynomial time many-one reductions).*

It is straightforward that the arithmetization of a QBF takes on a positive value iff the QBF is true. This fact also holds relative a not-too-large prime.

THEOREM 5.5 (see [24]). *A simple closed QBF B is true iff there exists a prime number P of size polynomial in $|B|$ such that the value of the arithmetization of B is positive modulo P. Moreover if B is false, then the value of the arithmetization of B is* 0 *modulo* any *such prime.*

THEOREM 5.6 (see [24]). *The functional form of every simple QBF can be represented by a univariate polynomial of degree at most* 3.

THEOREM 5.7 (see [24]). *For every simple QBF there exists an interactive protocol with prover P and polynomial time bounded verifier V such that*

1. *when B is true and P is honest, V always accepts the proof;*
2. *when B is false, V accepts the proof with negligible probability.*

The proof of Theorem 5.7 essentially uses Theorem 5.6 to translate a simple QBF to a polynomial in the following way. First, the arithmetization of a simple QBF B in closed form is an integer value V which is positive iff B is true. Then B's functional form F (recall that this is arithmetization of the QBF that is obtained from B by deleting the first quantifier) is a univariate polynomial p_1 of degree at most 3 which has the property that $p_1(0) *_1 p_1(1) = V$. Substituting any value r_1 in p_1 gives a new integer value V_1, which is of course the same value that we get when we substitute r_1 in F. However, $F(r_1)$ can again be converted to a (low-degree) polynomial by deleting its first $\sum$ or $\prod$ sign, and the above game can be repeated. Thus, we obtain a sequence of polynomials. From the first polynomial in this sequence V can be computed. The last polynomial p_n has the property that $p_n(r_1, \ldots, r_n) = \tilde{B}(r_1, \ldots, r_n)$. Two more things are needed: First, if any *other* sequence of polynomials $q_1, \ldots, q_n$ has the property that $q_1(0) *_1 q_1(1) \neq V$, $q_{i+1}(0) *_{i+1} q_{i+1}(1) = q_i(r_i)$, and $q_n(r_n) = \tilde{B}(r_1, \ldots, r_n)$,

then there has to be some i where $q_i(r_i) = p_i(r_i)$, yet $q_i \neq p_i$. That is, r_i is an intersection point of p_i and q_i. Second, all calculations can be done modulo some prime number of polynomial size (Theorem 5.5). We summarize this in the following observation, which is actually a skeleton of the proof of Theorem 5.7.

OBSERVATION 5.8 (see [24, 22]). *Let B be a closed simple QBF wherein the quantifiers are $Q_1, \ldots Q_n$ if read from left to right in its syntactic representation. Let A be its arithmetization, and let V be the value of A. There exist a prime number P of size polynomial in $|B|$ such that for any sequence $r_1, \ldots, r_n$ of numbers taken from $[1..P]$ there is a sequence of polynomials of degree at most 3 and size polynomial in $|B|$ such that*

1. *$p_1(0) *_1 p_1(1) = V$ and $V > 0$ iff B is true;*
2. *$p_{i+1}(0) *_{i+1} p_{i+1}(1) = p_i(r_i)$;*
3. *$p_n(r_n) = \tilde{B}(r_1, \ldots, r_n)$;*
4. *for any sequence of univariate polynomials $q_1, \ldots, q_n$ such that*
   (a) *$p_1(0) *_1 p_1(1) \neq q_1(0) *_1 q_1(1)$ and*
   (b) *$q_{i+1}(0) *_{i+1} q_{i+1}(1) = q_i(r_i)$ and*
   (c) *$q_n(r_n) = \tilde{B}(r_1, \ldots, r_n)$,*
   there is a minimal i such that $p_i \neq q_i$, yet $p_i(r_i) = q_i(r_i)$. That is, r_i is an intersection point of p_i and q_i.

Where all (in)equalities hold modulo P and hold modulo any *prime of polynomial size if B is false. Moreover, p_i can be computed in space $(|B| + |P|)^2$ from B, P, and $r_1, \ldots, r_{i-1}$.*

From this reformulation of Theorem 5.7 we obtain that for any sequence of univariate polynomials $q_1, \ldots, q_n$ and sequence of values $r_1, \ldots, r_n$ that satisfy items 2 and 3 in Observation 5.8 it holds that either $q_1(0) *_1 q_1(1)$ is the true value of the arithmetization of B, or there is some polynomial q_i in this sequence such that r_i is an intersection point of p_i and q_i (where p_i is as in Observation 5.8). As p_i can be computed in quadratic space from B, P, and $r_1, \ldots, r_{i-1}$ it follows that in the latter case r_i cannot have high space bounded Kolmogorov complexity relative to B, P, $q_1, \ldots, q_i$, $r_1, \ldots, r_{i-1}$. Hence, if r_i *does* have high space bounded Kolmogorov complexity, then r_i is *not* an intersection point, so the first case must hold (i.e., the value computed from q_1 is the true value of the arithmetization of B). The following lemma makes this precise.

LEMMA 5.9. *Assume the following for B, P, n, $q_1, \ldots, q_n$, $r_1, \ldots, r_n$, and $y_1, \ldots, y_n$.*

1. *B is a simple* false *closed QBF on n variables.*
2. *P is a prime number $\geq 2^{|B|}$ of size polynomial in $|B|$.*
3. *$q_1 \ldots q_n$ is a sequence of polynomials of degree 3 with coefficients in $[1..P]$.*
4. *$r_1, \ldots, r_n$ are numbers in $[1..P]$.*
5. *$y_1 = B\#P\#q_1\# \ldots \#q_n$ and $y_{i+1} = y_i\#r_i$.*
6. *$CS^{n^2}(r_i \mid y_i) \geq |P|$.*
7. *$(\forall i \geq 2)[q_{i-1}(r_{i-1}) = q_i(0) *_i q_i(1) \mod P]$.*
8. *$\tilde{B}(r_1, \ldots, r_n) = q_n(r_n) \mod P$.*

Then $q_1(0) *_1 q_1(1) = 0 \mod P$.

Proof. Take all calculations modulo P. Suppose $q_1(0) *_1 q_1(1) \neq 0$. It follows from Observation 5.8 that there exists a sequence $p_1, \ldots, p_n$ satisfying items 1 through 3 of that lemma. Furthermore since B is false $p_1(0) *_1 p_1(1) = 0$ modulo *any* prime, so $p_1(0) *_1 p_1(1) \neq q_1(0) *_1 q_1(1)$. It follows that there must be a minimal i such that $p_i \neq q_i$ and r_i is an intersection point of p_i and q_i. However, p_i can be computed in space $(|B|+|P|)^2$ from B, P, and $r_1, \ldots, r_{i-1}$. As both p_i and q_i have degree at most

3, it follows that $CS^{n^2}(r_i \mid y_i)$ is bounded by a constant—a contradiction. □

This suffices for the main theorem of this section. Let s be any polynomial.

THEOREM 5.10. PSPACE $\subseteq \mathrm{NP}^{\mathrm{cR}_s^{CS}}$.

Proof. We prove the lemma for $s(n) = n^2$, but the proof can by padding be extended to any polynomial. There exists an NP oracle machine that accepts the language of all simple closed true QBFs as follows. On input B first check that B is simple. Guess a prime number $P \geq 2^{|B|}$ of size polynomial in $|B|$, a sequence of polynomials $p_1, \ldots, p_n$ of degree at most 3 and with coefficients in $[1..P]$. Finally guess a sequence of numbers $r_1, \ldots, r_n$ all of size $|P|$. Check that

1. $p_1(0) *_1 p_1(1) > 0$ and
2. $p_{i+1}(0) *_{i+1} p_{i+1}(1) = p_i(r_i)$ and
3. $p_n(r_n) = B(r_1, \ldots, r_n)$ and
4. finally that $(\forall i \leq n)[CS^{n^2}(r_i|y_i) \geq |P|]$.

If B is true, Lemma 5.8 guarantees that these items can be guessed such that all tests are passed. If B is false and no other test fails, then Lemma 5.9 guarantees that $p_1(0) *_1 p_1(1) = 0$, so the first check must fail. □

By the fact that PSPACE is closed under complement and the fact that cR_s^{CS} is also in PSPACE Theorem 5.10 gives that cR_s^{CS} is complete for PSPACE under strong nondeterministic reductions [21].

COROLLARY 5.11. cR_s^{CS} *is complete for* PSPACE *under strong nondeterministic reductions.*

Buhrman and Mayordomo [10] showed that for $t(n) = 2^{n^k}$, the set $R_t^C = \{x : C^t(x) \geq |x|\}$ is *not* hard for EXP under deterministic Turing reductions. In Theorem 5.10 we made use of the relativized Kolmogorov complexity (i.e., $CS^s(x|y)$). Using exactly the same proof as in [10] one can prove that the set $\mathrm{cR}_t^C = \{\langle x, y\rangle : C^t(x|y) \geq |x|\}$ is not hard for EXP under Turing reductions. On the other hand the proof of Theorem 5.10 also works for this set: PSPACE $\subseteq \mathrm{NP}^{\mathrm{cR}_t^C}$. We suspect that it is possible to extend this to show that EXP $\subseteq \mathrm{NP}^{\mathrm{cR}_t^C}$. So far, we have been unable to prove this.

Acknowledgments. We thank Paul Vitányi for interesting discussions and providing the title of this paper. We also thank two anonymous referees, who helped with a number of technical issues that cleared up much of the proofs and who pointed us to more correct references. One of the referees also pointed out Corollary 4.8.

REFERENCES

[1] L. BABAI, *Trading group theory for randomness*, in Proceedings of the 17th ACM Symposium on Theory of Computing, Providence, RI, 1985, pp. 421–429.

[2] J. BALCÁZAR, J. DÍAZ, AND J. GABARRÓ, *Structural Complexity* I, Springer-Verlag, Berlin, 1988.

[3] J. M. BARZDIN, *Complexity of programs to determine whether natural numbers not greater than n belong to a recursively enumerable set*, Dokl. Akad. Nauk SSSR, 9 (1968), pp. 1251–1254.

[4] R. BEIGEL, H. BUHRMAN, AND L. FORTNOW, *NP might not be as easy as detecting unique solutions*, in Proceedings of the 30th ACM Symposium on Theory of Computing, Dallas, TX, ACM, New York, 1998, pp. 203–208.

[5] R. BOOK, S. GREIBACH, AND B. WEGBREIT, *Time- and tape-bound Turing acceptors and afl's*, J. Comput. System Sci., 4 (1970), pp. 606–621.

[6] H. BUHRMAN AND L. FORTNOW, *Resource bounded Kolmogorov complexity revisited*, in Proceedings of the 14th Annual Symposium on Theoretical Computer Science, Lecture Notes in Comput. Sci. 1200, Springer-Verlag, Berlin, 1997, pp. 105–116.

[7] H. BUHRMAN AND L. FORTNOW, *Resource Bounded Kolmogorov Complexity Revisited*, manuscript, available from http://www.neci.nj.nec.com/homepages/fortnow/.

[8] H. Buhrman, L. Fortnow, and L. Torenvliet, *Six hypotheses in search of a theorem*, in Proceedings of the 12th Annual IEEE Conference on Computational Complexity, Ulm, Germany, 1997, pp. 2–12.

[9] H. Buhrman and S. Homer, *Superpolynomial circuits, almost sparse oracles and the exponential hierarchy*, in Proceedings of the 12th Conference on the Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 652, R. Shyamasundar, ed., Springer-Verlag, Berlin, 1992, pp. 116–127.

[10] H. Buhrman and E. Mayordomo, *An excursion to the kolmogorov random strings*, in Proceedings of the 10th Annual Conference on Structure in Complexity Theory, Minneapolis, MN, 1995, IEEE Computer Society Press, Los Alamitos, CA, pp. 197–205.

[11] H. Buhrman and L. Torenvliet, *Complete sets and structure in subrecursive classes*, in Lecture Notes Logic 12, Springer-Verlag, Berlin, 1998, pp. 45–78.

[12] L. Fortnow and M. Kummer, *Resource-bounded instance complexity*, Theoret. Comput. Sci. A, 161 (1996), pp. 123–140.

[13] M. Fürer, O. Goldreich, Y. Mansour, M. Sipser, and S. Zachos, *On completeness and soundness in interactive proof systems*, in Randomness and Computation, Advances in Computing Research 5, S. Micali, ed., JAI Press, Greenwich, CT, 1989, pp. 429–442.

[14] J. Hartmanis, *Generalized Kolmogorov complexity and the structure of feasible computations*, in Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, Tucson, AZ, 1983, pp. 439–445.

[15] J. Hartmanis and R. Stearns, *On the computational complexity of algorithms*, Trans. Amer. Math. Soc., 117 (1965), pp. 285–306.

[16] F. Hennie and R. Stearns, *Two tape simulation of multitape Turing machines*, J. ACM, 13 (1966), pp. 533–546.

[17] K.-I Ko, *On the complexity of learning minimum time-bounded turing machines*, SIAM J. Comput., 20 (1991), pp. 962–986.

[18] M. Kummer, *On the complexity of random strings (extended abstract)*, in Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1046, Springer-Verlag, Berlin, 1996, pp. 25–36.

[19] L. Levin, *personal communication*, 1994.

[20] M. Li and P. Vitányi, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed., Grad. Texts Comput. Sci., Springer-Verlag, Berlin, 1997.

[21] T. Long, *Strong nondeterministic polynomial-time reducibilities*, Theoret. Comput. Sci., 21 (1982), pp. 1–25.

[22] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, *Algebraic methods for interactive proof systems*, J. ACM, 39 (1992), pp. 859–868.

[23] D. Martin, *Completeness, the recursion theorem and effectively simple sets*, Proc. Amer. Math. Soc., 17 (1966), pp. 838–842.

[24] A. Shamir, *IP = PSPACE*, J. ACM, 4 (1992), pp. 869–877.

[25] M. Sipser, *A complexity theoretic approach to randomness*, in Proceedings of the 15th ACM Symposium on Theory of Computing, Boston, MA, 1983, pp. 330–335.

[26] L. Valiant and V. Vazirani, *NP is as easy as detecting unique solutions*, Theoret. Comput. Sci., 47 (1986), pp. 85–93.

[27] C. K. Yap, *Some consequences of non-uniform conditions on uniform classes*, Theoret. Comput. Sci., 26 (1983), pp. 287–300.

[28] S. Zachos, *Probabilistic quantifiers and games*, J. Comput. System Sci., 36 (1988), pp. 433–451.

[29] D. Zuckerman, *Randomness-optimal sampling, extractors, and constructive leader election*, in Proceedings of the 28th ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 286–295.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1502–1531

ON THE DETERMINIZATION OF WEIGHTED FINITE AUTOMATA*

ADAM L. BUCHSBAUM†, RAFFAELE GIANCARLO‡, AND JEFFERY R. WESTBROOK§

Abstract. We study the problem of constructing the deterministic equivalent of a nondeterministic weighted finite-state automaton (WFA). Determinization of WFAs has important applications in automatic speech recognition (ASR). We provide the first polynomial-time algorithm to test for the *twins* property, which determines if a WFA admits a deterministic equivalent. We also give upper bounds on the size of the deterministic equivalent; the bound is tight in the case of acyclic WFAs. Previously, Mohri presented a superpolynomial-time algorithm to test for the twins property, and he also gave an algorithm to determinize WFAs. He showed that the latter runs in time linear in the size of the output when a deterministic equivalent exists; otherwise, it does not terminate. Our bounds imply an upper bound on the running time of this algorithm.

Given that WFAs can expand exponentially in size when determinized, we explore why those that occur in ASR tend to *shrink* when determinized. According to ASR folklore, this phenomenon is attributable solely to the fact that ASR WFAs have simple topology, in particular, that they are acyclic and layered. We introduce a very simple class of WFAs with this structure, but we show that the expansion under determinization depends on the transition weights: some weightings cause them to shrink, while others, including random weightings, cause them to expand exponentially. We provide experimental evidence that ASR WFAs exhibit this weight dependence. That they shrink when determinized, therefore, is a result of favorable weightings in addition to special topology. These analyses and observations have been used to design a new, approximate WFA determinization algorithm, reported in a separate paper along with experimental results showing that it achieves significant WFA size reduction with negligible impact on ASR performance.

Key words. algorithms, rational functions and power series, speech recognition, weighted automata

AMS subject classifications. 20M35, 68Q25, 68Q45, 68T10

PII. S0097539798346676

1. Introduction.

Finite-state machines and their relation to rational functions and power series have been extensively studied [2, 3, 13, 19] and widely applied in fields ranging from image compression [10, 11, 12, 17] to natural language processing [20, 21, 22, 28, 30]. A subclass of finite-state machines, the weighted finite-state automata (henceforth simply weighted finite automata (WFAs)), has recently assumed new importance, because WFAs provide powerful method for representing and manipulating models of human language in automatic speech recognition (ASR) systems [23, 24]. This new research direction also raises a number of challenging algorithmic questions [5].

A WFA is a nondeterministic finite automaton (NFA), A, that has both an alpha-

*Received by the editors November 6, 1998; accepted for publication (in revised form) March 23, 2000; published electronically November 8, 2000. An extended abstract of this paper appeared in *Proceedings of the* 25*th International Conference on Automata, Languages, and Programming*, Aalborg, Denmark, 1998, Lecture Notes in Comput. Sci. 1443, Springer-Verlag, New York, 1998, pp. 482–493.

http://www.siam.org/journals/sicomp/30-5/34667.html

†AT&T Labs, Shannon Laboratory, 180 Park Avenue, Florham Park, NJ 07932 (alb@research.att.com).

‡Dipartimento di Matematica ed Applicazioni, Universitá di Palermo, Via Archirafi 34, 90123 Palermo, Italy (raffaele@altair.math.unipa.it). The work of this author was supported by AT&T Labs.

§20th Century Television, Los Angeles, CA 90025 (jwestbrook@acm.org). The work of this author was completed while he was a member of AT&T Labs.

bet symbol and a weight, from some set K, on each transition. Let $R = (K, \oplus, \otimes, \overline{0}, \overline{1})$ be a semiring. Then A together with R generates a partial function from strings to K: the value of an accepted string is the semiring sum over accepting paths of the semiring product of the weights along each accepting path. A partial function that can be generated this way is a *rational power series* [29]. An example important to ASR is the set of WFAs with the *min-sum semiring*, $(\Re^+ \cup \{0, \infty\}, \min, +, \infty, 0)$, which compute for each accepted string the minimum cost accepting path.

In this paper, we study problems related to the determinization of WFAs. A *deterministic*, or *sequential*, WFA has at most one transition with a given input symbol out of each state. Not all rational power series can be generated by deterministic WFAs. A *determinization algorithm* takes as input a WFA and produces a deterministic WFA that generates the same rational power series, if such a deterministic WFA exists. The importance of determinization to ASR is well established [20, 23, 24].

To the best of our knowledge, Mohri [20] presented the first determinization procedure for WFAs, extending the seminal ideas of Choffrut [8, 9] and Weber and Klemm [31] regarding string-to-string transducers. Mohri gives a determinization procedure with three phases. First, A is converted to an equivalent unambiguous, trim WFA A_t, using an algorithm analogous to one for NFAs [13]. (We define *unambiguous* and *trim* below.) Mohri then gives an algorithm, **TT**, that determines if A_t has the *twins property* (also defined below). If A_t does not have the twins property, then there is no deterministic equivalent of A. If A_t has the twins property, a second algorithm of Mohri's, **DTA**, can be applied to A_t to yield A', a deterministic equivalent of A. Algorithm **TT** runs in $O(m^{4n^2})$ time, where m is the number of transitions and n the number of states in A_t. Algorithm **DTA** runs in time linear in the size of A'. In practice, **DTA** is run directly on A, which is assumed to admit a deterministic equivalent; conversion to A_t and testing for twins are theoretical steps needed to make the procedure well defined. Mohri observes that A' can be exponentially larger than A, because WFAs include classical NFAs. He gives no upper bound on the worst-case state-space expansion, however, and because of the weights on transitions, the classical NFA upper bound does not apply. Finally, Mohri gives an algorithm that takes a deterministic WFA and outputs the minimum-size equivalent, deterministic WFA.

We present several results related to the determinization of WFAs. In section 3 we give the first polynomial-time algorithm to test whether an unambiguous, trim WFA satisfies the twins property. It runs in $O(m^2 n^6)$ time. We then provide a worst-case time complexity analysis of **DTA**. The number of states in the output deterministic WFA is at most $2^{n(2\log n + n^2 \log|\Sigma| + 1)}$, where Σ is the input alphabet. If the weights are rational, this bound becomes $2^{n(2\log n + 1 + \min(n^2 \log|\Sigma|, \rho))}$, where ρ is the maximum bit-size of a weight. When the input WFA is acyclic, the bound becomes $2^{n \log|\Sigma|}$. The acyclic bound holds for real weights, and it is tight (up to constant factors) for any alphabet size. It remains open whether there exists a polynomial-time procedure to determine whether an arbitrary WFA admits a deterministic equivalent, because the determinization process above requires converting a WFA to an unambiguous equivalent prior to testing for twins.

In sections 4–6 we study questions motivated by the use of WFA determinization in ASR [23, 24]. Although determinization causes exponential state-space expansion in the worst case, in ASR systems the determinized WFAs are often *smaller* than the input WFAs [20], and they are seldom very large. This is fortunate, because the performance of ASR systems depends directly on WFA size [23, 24]. Folklore within the ASR community credits this phenomenon entirely to the special topology of ASR

WFAs. (The topology of a WFA is its underlying directed graph and labeling by input symbols, ignoring weights.) ASR WFAs tend to be acyclic and layered. Such a WFA always admits a deterministic equivalent. The role that the transition weights might play in controlling expansion under determinization has not been considered.

In section 4 we study the role of topology in expansion under determinization. We exhibit a class of layered, acyclic WFAs whose minimum equivalent deterministic WFAs are exponentially larger regardless of weighting. The languages accepted by these WFAs are quite unnatural, however.

In section 5 we study the role of transition weights in expansion under determinization. We introduce a class of nondeterministic WFAs, RG. Each WFA in this class has an extremely simple multipartite, acyclic topology, accepts a very trivial language, and in the absence of weights (i.e., with all weights set to zero) has a smaller deterministic equivalent. We show, however, that for any $A \in RG$ and any $i \leq n$, there exists an assignment of weights to the transitions of A such that the minimal equivalent deterministic WFA has $\Theta(2^{i \log |\Sigma|})$ states. This gives a lower bound to match the upper bounds of section 3. Using ideas from universal hashing, we also show that similar results hold when the weights are random i-bit numbers.

This motivates us to examine experimentally the effect of varying weights on actual WFAs from ASR applications. In section 6 we give the results of these experiments. We call a WFA *weight-dependent* if its expansion under determinization is strongly determined by its weights. Most of the examples from ASR were weight-dependent. These experimental results together with the theory we develop show that the folklore explanation is insufficient: ASR WFAs shrink under determinization because both the topology and weighting tend to be favorable.

Some of our results help explain the nature of WFAs from the algorithmic point of view, i.e., how weights assigned to the transitions of a WFA can affect the performance of algorithms manipulating it. Others relate directly to the theory of weighted automata. We have used our results to design an approximate variant of Mohri's determinization algorithm. We describe this algorithm separately [6], along with experimental results showing that it achieves size reductions in ASR language models that significantly exceed those of previous methods, with negligible effects on ASR performance (time and accuracy).

2. Definitions and terminology. Given a semiring $(K, \oplus, \otimes, \overline{0}, \overline{1})$, a *WFA* is a tuple $G = (Q, \bar{q}, \Sigma, \delta, Q_f)$. Q is the set of states, $\bar{q} \in Q$ is the initial state, Σ is the set of symbols, $\delta \subseteq Q \times \Sigma \times K \times Q$ is the set of transitions, and $Q_f \subseteq Q$ is the set of final states. We assume that $|\Sigma| > 1$. A *deterministic*, or *sequential*, WFA has at most one transition $t = (q_1, \sigma, \nu, q_2)$ for any pair (q_1, σ); a *nondeterministic* WFA can have multiple transitions on a pair (q_1, σ), differing in target state q_2. The problems examined in this paper are motivated primarily by ASR applications, which work with the *min-sum semiring*, $(\Re^+ \cup \{0, \infty\}, \min, +, \infty, 0)$, and we therefore limit further discussion to the min-sum semiring. (Some of the algorithms considered use subtraction. To be well defined, therefore, they require a skew field. The min-sum semiring is indeed embedded in a skew field [16].)

Let $\vec{t} = (t_1, \ldots, t_\ell)$ be some sequence of transitions, such that $t_i = (q_{i-1}, \sigma_i, \nu_i, q_i)$; $\vec{t}$ *induces* string $w = \sigma_1 \cdots \sigma_\ell$. String w is *accepted by* $\vec{t}$ if $q_0 = \bar{q}$ and $q_\ell \in Q_f$; w is *accepted by* G if some $\vec{t}$ accepts w. Let $c(t_i) = \nu_i$ be the *weight* of t_i. Then the *weight*

of $\vec{t}$ is

$$c(\vec{t}) = \sum_{i=1}^{\ell} c(t_i).$$

Let $T(w)$ be the set of all sequences of transitions that accept string w. Then the *weight of* w is

$$c(w) = \min_{\vec{t} \in T(w)} c(\vec{t}).$$

The *weighted language* of G is the set $L(G) = \{(w, c(w)) \mid w \text{ is accepted by } G\}$; i.e., the weighted strings accepted by G. Intuitively, the weight on a transition of G can be seen as the "confidence" one has in taking that transition. The weights need not, however, satisfy stochastic constraints, as do the probabilistic automata introduced by Rabin [26].

Fix two states q and q' and a string $v \in \Sigma^*$. Let $c(q, v, q')$ be the minimum of $c(\vec{t})$, taken over all transition sequences $\vec{t}$ from q to q' inducing v. We refer to $c(q, v, q')$ as the *optimal cost* of inducing v from q to q'. We generally abuse notation so that $\delta(q, w)$ can represent the set of states reachable from state $q \in Q$ on string $w \in \Sigma^*$. We extend the function δ to strings in the usual way: $q' \in \delta(q, v), v \in \Sigma^+$, means that there is a sequence of transitions from q to q' inducing v.

The *topology* of G is the projection $\pi_{Q \times \Sigma \times Q}(\delta)$; i.e., the transitions of G without respect to the weights. We also refer to the topology of G as the *graph* underlying G.

A WFA is *trim* if every state appears in an accepting path for some string and no transition is weighted $\overline{0}$ (∞ in the min-sum semiring). A WFA is *unambiguous* if there is exactly one accepting path for each accepted string.

Determinization of G is the problem of computing a deterministic WFA G' such that $L(G') = L(G)$, if such a G' exists. We denote the output of algorithm **DTA** by *dta*(G). We denote the minimal deterministic WFA accepting $L(G)$ by *min*(G), if one exists. We say that G *expands* if *dta*(G) has more states and/or transitions than G.

Let $n = |Q|$ and $m = |\delta|$, and let the *size* of G be $|G| = n + m$. We also use $\#G$ to mean $|Q|$, the number of states of G. We assume that each transition is labeled with exactly one symbol, so $|\Sigma| \leq m$. Recall that the weights of G are nonnegative real numbers. Let C be the maximum weight. In the *general case*, weights are incommensurable real numbers, requiring "infinite precision." In the *integer case*, weights can be represented with $\rho = \lceil \lg C \rceil$ bits. We denote the integral range $[a, b]$ by $[a, b]_Z$. The integer case extends to the case in which the weights are rationals requiring ρ bits. We assume that in the integer and rational cases, weights are normalized to remove excess least-significant zero bits.

For our analyses, we use the RAM model of computation as follows. In the general case, we charge constant time for each arithmetic-logic operation involving weights (which are real numbers). We refer to this model as the $\Re$-RAM [25]. The relevant parameters for our analyses are n, m, and $|\Sigma|$. In the integer case, we also use a RAM, except that each arithmetic-logic operation now takes $O(\rho)$ time. We refer to this model as the $\mathcal{CO}$-RAM [1]. The relevant parameters for the analyses are n, m, $|\Sigma|$, and ρ.

3. Determinization of WFAs.

3.1. An algorithm for testing the twins property.

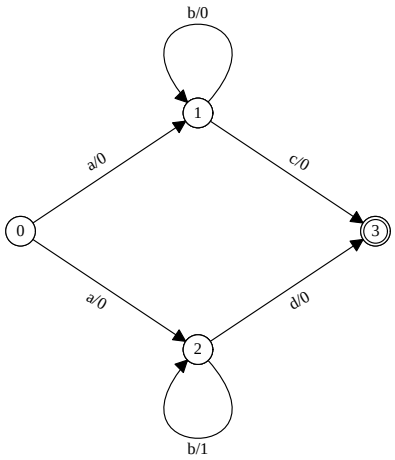


FIG. 1. *A nondeterministic, trim, unambiguous WFA G. Arcs labeled σ/w correspond to transitions labeled σ with weight w. For this and succeeding figures, the start state is the unique source, and final states are denoted by double circles. G accepts the language $\{(ab^n c, 0),\ (ab^n d, n) \mid n \geq 0\}$. States 1 and 2 do not have the twins property: each is reachable from state 0 via string a, yet the costs of the cycles labeled b at each differ. It is easily shown that no deterministic WFA can accept $L(G)$.*

DEFINITION 3.1. *Two states, q and q', of a WFA G are* twins *if $\forall u, v \in \Sigma^*$ such that $q \in \delta(\bar{q}, u)$, $q' \in \delta(\bar{q}, u)$, $q \in \delta(q, v)$, and $q' \in \delta(q', v)$, the following holds: $c(q, v, q) = c(q', v, q')$. G has the* twins property *if all pairs $q, q' \in Q$ are twins.*

That is, if states q and q' are reachable from $\bar{q}$ by a common string, then q and q' are twins only if any string that induces a cycle at each induces cycles of equal optimal cost. Note that two states having no cycle on a common string are twins. Mohri [20, Theorems 11 and 12] proves that any WFA G over the min-sum semiring is determinizable if it has the twins property; furthermore, if G is trim and unambiguous, the twins property becomes a necessary and sufficient condition. For an example of a nondeterminizable WFA, see Figure 1.

The twins property for WFAs is analogous to that defined by Choffrut [8, 9] and (in different terms) by Weber and Klemm [31] to identify necessary and sufficient conditions for a string-to-string transducer to admit a sequential transducer realizing the same rational transduction. In spite of the strong analogy, the proof techniques used for WFAs differ from those used to obtain analogous results for string-to-string transducers. In particular, the efficient algorithm we derive here to test a WFA for twins is not related to the polynomial-time algorithm of Weber and Klemm [31] for testing twins in string-to-string transducers. We reduce the problem of testing the twins property to that of computing shortest paths on some suitably defined graphs, which we introduce next.

Let $T_{\bar{q},\bar{q}}$ be the multipartite acyclic, labeled, weighted graph having $2n^2$ layers and inductively defined as follows. The root vertex $\hat{r}$ is at layer zero and corresponds to $(\bar{q}, \bar{q})$. The vertices at layer one correspond to a subset of $Q \times Q$ obtained as follows: $\hat{r}$ is connected to a vertex u, corresponding to (q_1, q_2), if and only if there are two distinct transitions $t = (\bar{q}, a, c_1, q_1)$ and $t' = (\bar{q}, a, c_2, q_2)$ in G. The arc connecting $\hat{r}$ to u is labeled with $a \in \Sigma$ and has cost $c = c_1 - c_2$. Assume that we have the vertices at layer $i - 1$. The vertices at layer i are obtained as follows. Let u be the vertex

at layer $i-1$ corresponding to $(q_1, q_2) \in Q \times Q$; u is connected to u', corresponding to (q_1', q_2'), at layer i if and only if there are two distinct transitions $t = (q_1, a, c_1, q_1')$ and $t' = (q_2, a, c_2, q_2')$ in G. The arc connecting u to u' is labeled with $a \in \Sigma$ and has cost $c = c_1 - c_2$. This graph has $O(n^4)$ vertices and $O(m^2n^4)$ arcs. Let $(q, q')_i$ denote the vertex corresponding to $(q, q') \in Q \times Q$ at layer i of $T_{\bar{q},\bar{q}}$, if any. Let $RT \subseteq \{(q, q') \mid q \neq q'\}$ be the set of pairs of distinct states of G that are reachable from $(\bar{q}, \bar{q})_0$ in $T_{\bar{q},\bar{q}}$. For each $(q, q') \in RT$, define $T_{q,q'}$ analogously to $T_{\bar{q},\bar{q}}$. Notice that $T_{q,q'}$ has $O(n^4)$ vertices and $O(m^2n^4)$ arcs. We need the following.

LEMMA 3.2. *Fix two distinct states q and q' of G. They can be reached from the initial state $\bar{q}$ of G by the same string $z \in \Sigma^+$ if and only if there exists some string $z' \in \Sigma^i$ for some $1 \leq i \leq 2n^2 - 1$ such that q and q' are both reached from $\bar{q}$ using z'. In that case, there is at least one path in $T_{\bar{q},\bar{q}}$ that goes from $(\bar{q}, \bar{q})_0$ to $(q, q')_i$.*

Proof. Fix a string $z \in \Sigma^+$, and assume that q and q' can be reached from $\bar{q}$ by z. Assume that $|z| > 2n^2 - 1$, or else we are done. Since there are only n^2 distinct pairs of states of G and $|z| > 2n^2 - 1$, there must exist two states q_1 and q_2 and a string $v \in \Sigma^+$ such that (a) $z = xvu$; (b) q_1 (resp., q_2) is on a path from $\bar{q}$ to q (resp., q') inducing z; and (c) $q_1 \in \delta(q_1, v)$ (resp., $q_2 \in \delta(q_2, v)$). But then, $z' = xu$ also reaches both q and q' from $\bar{q}$. If $|z'| \leq 2n^2 - 1$, we are done; otherwise we iterate the argument. The second part of the lemma follows by construction of $T_{\bar{q},\bar{q}}$. □

LEMMA 3.3. *Let G be trim and unambiguous. Fix a string $y \in \Sigma^i$, $1 \leq i \leq 2n^2-1$, and two distinct states q and q' of G. Then $q \in \delta(q, y)$ and $q' \in \delta(q', y)$ if and only if there is exactly one path $\mathbf{p}$ in $T_{q,q'}$ that starts at $(q, q')_0$, ends at $(q, q')_{|y|}$, and induces y. Moreover, the cost of $\mathbf{p}$ is $c(q, y, q') - c(q', y, q')$.*

Proof. We prove the sufficient case; the necessary case should be clear from the construction of $T_{q,q'}$.

First observe that, since G is trim and unambiguous, the following holds: for each string $y \in \Sigma^+$ such that $q \in \delta(q, y)$, there is exactly one cycle starting and ending at q and inducing y.

Let $(q = q_0, q_1, q_2, \ldots, q_{|y|} = q)$ be the unique sequence of states of G originating in q and inducing y in G. Therefore, $c(q, y, q)$ is the sum of the weights on the transitions in that sequence. Similarly define $(q' = q_0', q_1', q_2', \ldots, q_{|y|}' = q')$. By the above construction, there exists a path $\mathbf{p}$ in $T_{q,q'}$, consisting of the vertices $((q, q')_0, (q_1, q_1')_1, \ldots, (q, q')_{|y|})$ and inducing y. This path must be unique, and its cost is $c(q, y, q) - c(q', y, q')$. □

LEMMA 3.4 (see [20, Lemma 2]). *Let G be a trim, unambiguous WFA. G has the twins property if and only if $\forall u, v \in \Sigma^*$, such that $|uv| \leq 2n^2 - 1$, the following holds: when there exist two states q and q' such that* (i) $\{q, q'\} \subseteq \delta(\bar{q}, u)$ *and* (ii) $q \in \delta(q, v)$ *and* $q' \in \delta(q', v)$, *then* (iii) $c(q, v, q) = c(q', v, q')$ *must follow.*

Fix two distinct states q and q' of G. Let $(q, q')_{i_1}, (q, q')_{i_2}, \ldots, (q, q')_{i_s}$, $0 < i_1 < i_2 < \cdots < i_s$, be all the occurrences of (q, q') in $T_{q,q'}$, excluding $(q, q')_0$. This sequence may be empty. A symmetric sequence can be extracted from $T_{q',q}$. We refer to these sequences as the *common cycles sequences* of (q, q'). We say that q and q' satisfy the *local twins property* if and only if (1) their common cycles sequences are empty, or (2) zero is the cost of any shortest path from $(q, q')_0$ to $(q, q')_{i_j}$ in $T_{q,q'}$ and from $(q', q)_0$ to $(q', q)_{i_j}$ in $T_{q'q}$ $\forall\, 1 \leq j \leq s$.

LEMMA 3.5. *Let G be a trim, unambiguous WFA. G satisfies the twins property if and only if* (i) *RT is empty, or* (ii) *all $(q, q') \in RT$ satisfy the local twins property.*

Proof.

⇒) Assume that G satisfies the twins property. If RT is empty, we are done.

Assume then that $RT \neq \emptyset$. The proof is by contradiction. Assume that some $(q, q') \in RT$ does not satisfy the local twins property. The common cycles sequences of (q, q') cannot be empty, or else they would satisfy the local twins property. By assumption, there exists some j for which the cost of some shortest path from $(q, q')_0$ to $(q, q')_{i_j}$ in $T_{q,q'}$ is not zero, while the cost of a shortest path from $(q', q)_0$ to $(q', q)_{i_j}$ in $T_{q'q}$ may be any value, including zero (or vice versa). Fix any such shortest path $\mathbf{p}$ in $T_{q,q'}$. According to Lemma 3.3, $\mathbf{p}$ corresponds to cycles around q and q' that each induce the same string y for some $y \in \Sigma^{i_j}$. Moreover, we must have $c(q, y, q) - c(q', y, q') \neq 0$. By definition of RT, q and q' are each reachable by some string u from the initial state of G. Therefore, G does not satisfy the twins property, which is a contradiction.

$\Leftarrow$) Assume that RT is empty. Then, by Lemma 3.2, no two distinct states q, q' of G can both be reached by some string $z \in \Sigma^+$ from the initial state $\bar{q}$. Therefore, G satisfies the twins property. Assume now that RT is not empty. We have two subcases.

Subcase A. Assume that all states in RT satisfy the local twins property because their common cycles sequences are empty. This implies that all pairs of distinct states reachable from the initial state of G through the same string $z \in \Sigma^+$ do not have any cycles in common inducing identical strings. Thus, G satisfies the twins property.

Subcase B. Assume that some states in RT satisfy the local twins property and their common cycles sequences are not empty. Let RT' be such a set. Assume that G does not satisfy the twins property. We derive a contradiction. Since RT' is not empty, we have that the set of pairs of states for which (i) and (ii) are satisfied in Lemma 3.4 is not empty. But since G does not satisfy the twins property, there must exist two distinct states q and q' and a string $uv \in \Sigma^*$, $|uv| \leq 2n^2 - 1$, such that (i) both q and q' can be reached from the initial state of G through string u; (ii) $q \in \delta(q, v)$ and $q' \in \delta(q', v)$; and (iii) $c(q, v, q) \neq c(q', v, q')$. We now argue that (q, q') must be in RT'. Because $q \neq q'$ and G has only one initial state, we have that $|u| \geq 1$. Thus, $1 \leq |u| \leq 2n^2 - 1$, implying that $(q, q') \in RT$. v cannot be the empty string ϵ because $c(q, \epsilon, q) = c(q', \epsilon, q') = 0$. Since $|uv| \leq 2n^2 - 1$, we have that $1 \leq |v| \leq 2n^2 - 1$. But then, by Lemma 3.3 and (ii) above, we have that $(q, q')_{|v|}$ can be reached from $(q, q')_0$ in $T_{q,q'}$ through the nonempty string v. Therefore, the common cycles sequences of (q, q') cannot be empty, implying that $(q, q') \in RT'$. Without loss of generality, assume that $c(q, v, q) - c(q', v, q') < 0$. Since $1 \leq |v| \leq 2n^2 - 1$, we have by Lemma 3.3 that there is exactly one path $\mathbf{p}$ in $T_{q,q'}$ starting at $(q, q)_0$, ending in $(q, q')_{|v|}$, inducing v, and with cost $c(q, v, q) - c(q', v, q') < 0$. Since $\mathbf{p}$ has negative cost, the cost of the shortest path from $(q, q')_0$ to $(q, q')_{|v|}$ in $T_{q,q'}$ cannot be zero, which contradicts that q and q' satisfy the local twins property and have nonempty common cycles sequences. □

Our algorithm for testing whether a trim, unambiguous WFA has the twins property works as follows. First, compute $T_{\bar{q},\bar{q}}$ and the set RT. Then, for each pair of states $(q, q') \in RT$ that have not been processed yet, compute $T_{q,q'}$ and $T_{q',q}$, extract the common cycles sequences, and compute the single source (from the root) shortest paths to vertices in $T_{q,q'}$ and $T_{q',q}$.

THEOREM 3.6. *Let G be a trim unambiguous WFA. In the general case, whether G satisfies the twins property can be checked in $O(m^2n^6)$ time using the $\Re$-RAM model of computation. In the integer case, the bound becomes $O(\rho m^2 n^6)$ using the $\mathcal{CO}$-RAM model of computation.*

Proof. Lemma 3.5 implies correctness. We now analyze the algorithm, starting with the general case. Recall that each arithmetic-logic operation can be done in

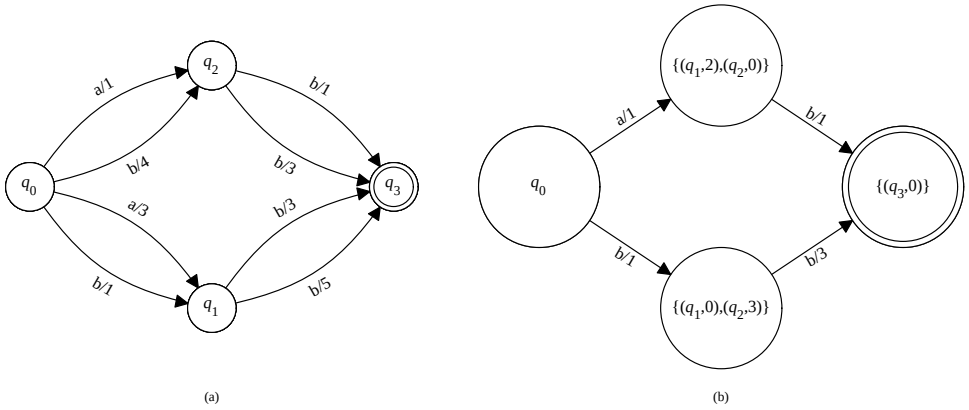


FIG. 2. (a) *A nondeterministic weighted automaton, A. Arcs labeled σ/w correspond to transitions labeled σ with weight w.* (b) *The result of applying* **DTA** *to A. This is derived from Figures* 11 *and* 12 *of Mohri* [20].

constant time. $T_{\bar{q},\bar{q}}$ can be easily obtained in $O(m^2n^4)$ time by visiting the automaton G. Now, visiting $T_{\bar{q},\bar{q}}$, we can obtain the set RT in the same amount of time.

Fix a pair of distinct states q and q' of G. It is sufficient to discuss how to compute shortest paths from the root vertex of $T_{q,q'}$ to the other vertices in the graph. Notice that the edges of $T_{q,q'}$ may have negative cost. However, $T_{q,q'}$ is a multipartite acyclic graph. In that case, it is a simple exercise to show how to perform the required computation in time linear in the size of $T_{q,q'}$, i.e., $O(m^2n^4)$ time. Since $|RT| = O(n^2)$, the total time of the algorithm is $O(m^2n^6)$.

For the integer case, we multiply the above bound by ρ. □

We also mention, omitting the details, that the exponential-time algorithm for testing the twins property originally devised by Mohri [20] can be simplified and implemented to run in pseudopolynomial time in the integer case. The algorithm we devise here is weakly polynomial in the integer case.

3.2. The DTA algorithm. Mohri [20] describes a determinization algorithm for a finite-state automaton with weights drawn from a general semiring. What we refer to as **DTA** is that algorithm restricted to the min-sum semiring. **DTA** is a generalization of the classic power-set construction for finite automata. We describe the algorithm, starting with an example.

Consider the weighted automaton, A, in Figure 2(a). While A is not unambiguous, it has the twins property, and so we can apply **DTA** directly to it, proceeding as follows. From the initial state q_0, we can reach states q_1 and q_2 using the input symbol a. Analogously to the determinization of finite-state automata, we establish a new state $\{q_1, q_2\}$ in A', reachable from q_0 with input symbol a. The transitions to q_1 and q_2, however, have different weights in A. **DTA** selects the smaller weight to be the weight of the transition to $\{q_1, q_2\}$ and records the difference between the two weights in the new state. In the example, the weight of the $q_0 \to q_1$ transition is 3, and that of the $q_0 \to q_2$ transition is 1. Therefore, the new transition to $\{q_1, q_2\}$ gets weight 1, and the difference of $2 = 3 - 1$ is assigned as a *remainder* to component q_1. For completeness, a remainder of $0 = 1 - 1$ is assigned to q_2. The new state is thus encoded as $\{(q_1, 2), (q_2, 0)\}$ in A'. Similarly, from state q_0 in A, we can reach states q_1 and q_2 via symbol b. Again the minimum weight among these transitions is

1, so we assign this weight to the new arc and encode the remainder weights (0 and 3, respectively) in the new state $\{(q_1, 0), (q_2, 3)\}$ in A'.

In general, the states in A' are of the form $\hat{q} = \{(q_{i_1}, r_{i_1}), \ldots, (q_{i_\ell}, r_{i_\ell})\}$. The q_i's are states from A, and the r_i's are called *remainders*. Each such $\hat{q}$ is interpreted as follows. Consider any string $w \in \Sigma^*$ such that there is a (single) path inducing w from the start state, q_0, to $\hat{q}$. As in classical automata determinization, there is at least one path inducing w from q_0 to each q_{i_j} in the nondeterministic input, A. Let c_j be the weight of the minimum weight path inducing w from q_0 to q_{i_j} in A. Let c be the weight of the path from q_0 to $\hat{q}$ in A'. The remainders are constructed so that $r_{i_j} = c_j - c$. In this way, all necessary path length information is encoded into the transition weights and remainders in A'.

Returning to the example, consider state $\{(q_1, 2), (q_2, 0)\}$ in A' and the input symbol b. In A, we can reach state q_3 from both q_1 and q_2. Recalling the above discussion of remainders, we consider the sum of the weight of the transition in A (3 for the $q_1 \to q_3$ transition and 1 for the $q_2 \to q_3$ transition) plus the remainder associated with the original source state encoded in state $\{(q_1, 2), (q_2, 0)\}$ in A'. That is, we consider the sums $3 + 2 = 5$ and $1 + 0 = 1$. We take the minimum among those values, i.e., 1, as the weight of the transition from $\{(q_1, 2), (q_2, 0)\}$ to $\{(q_3, r)\}$ (r to be determined) in A'. Since there is only one destination state (q_3) in A, the remainder r is 0, so we encode the new destination state as $\{(q_3, 0)\}$. Similarly, we construct an arc with weight 3 on symbol b from $\{(q_1, 0), (q_2, 3)\}$ to $\{(q_3, 0)\}$. ($3+0 = 3$, $1+3 = 4$, and we take the minimum, which is 3.) The end result is shown in Figure 2(b).

Generalizing to an arbitrary WFA $G = (Q, \bar{q}, \Sigma, \delta, Q_f)$, the deterministic WFA G' is obtained as follows. The start state of G' is $\{(\bar{q}, 0)\}$, which forms an initial set P. While $P \neq \emptyset$, we remove any state $q = \{(q_1, r_1), \ldots, (q_n, r_n)\}$ from P, where $q_i \in Q$ and $r_i \in \Re^+ \cup \{0, \infty\}$. The remainders encode path length information, as described above. For each $\sigma \in \Sigma$, let $\{q'_1, \ldots, q'_m\}$ be the set of states reachable by σ-transitions out of all the q_i. For $1 \leq j \leq m$, let

$$\rho_j = \min_{1 \leq i \leq n; (q_i, \sigma, \nu, q'_j) \in \delta} \{r_i + \nu\}$$

be the minimum of the weights of σ-transitions into q'_j from the q_i plus the respective r_i. Let $\rho = \min_{1 \leq j \leq m}\{\rho_j\}$. Let $q' = \{(q'_1, s_1), \ldots, (q'_m, s_m)\}$, where $s_j = \rho_j - \rho$ for $1 \leq j \leq m$. If q' is a new state, we add it to P. We add transition (q, σ, ρ, q') to G'. This is the only σ-transition out of state q, so G' is deterministic.

Let $T_G(w)$ be the set of sequences of transitions in G that accept a string $w \in \Sigma^*$; let $\vec{t}_{G'}(w)$ be the (one) sequence of transitions in G' that accepts the same string. Mohri [20] shows that

$$c(\vec{t}_{G'}(w)) = \min_{\vec{t} \in T_G(w)} \{c(\vec{t})\},$$

and thus $L(G') = L(G)$. Moreover, let $T_G(w, q)$ be the set of sequences of transitions in G from state $\bar{q}$ to state q that induce string w. Again, let $\vec{t}_{G'}(w)$ be the (one) sequence of transitions in G' that induces the same string; $\vec{t}_{G'}(w)$ ends at some state $\{(q_1, r_1), \ldots, (q_n, r_n)\}$ in G' such that some $q_i = q$. Mohri [20] shows that

$$c(\vec{t}_{G'}(w)) + r_i = \min_{\vec{t} \in T_G(w,q)} \{c(\vec{t})\}.$$

Thus each remainder r_i encodes the difference between the weight of the shortest path to some state that induces w in A and the weight of the path inducing w in A', as described above. Hence at least one remainder in each state is zero.

3.3. An analysis. We first bound $\#dta(G)$, the number of states in $dta(G)$. The results of section 5 show that our upper bound is tight to within polynomial factors.

LEMMA 3.7. *Assume that G satisfies the twins property. Let $\tilde{R}$ be the set of remainders generated by* **DTA** *when computing $dta(G)$. Let R be the set of remainders r for which the following holds: $\exists w \in \Sigma^*$, $|w| \leq n^2 - 1$, and two states q_1 and q_2, such that $r = |c(\bar{q}, w, q_2) - c(\bar{q}, w, q_1)|$. Then $\tilde{R} \subseteq R$.*

Proof. Let R' be the set of remainders r' such that $\exists w' \in \Sigma^*$ and two states q_1 and q_2 such that $r' = |c(\bar{q}, w', q_2) - c(\bar{q}, w', q_1)|$. Consider a state-remainder tuple in $dta(G)$ reached by w' from the initial state, and assume that q_1 is the optimal state in that tuple, i.e., the one with zero remainder. Then the remainder associated to q_2 is r'. Thus, $\tilde{R} \subseteq R'$. We next show that $R = R'$.

Clearly $R \subseteq R'$. To prove the other inclusion we only need to show that the remainder r generated by any string of length at least n^2 is generated by a string of length at most $n^2 - 1$. Let $\mathbf{p_1}$ and $\mathbf{p_2}$ be the paths of minimum cost in G, starting at $\bar{q}$, ending at q_1 and q_2, respectively, and each inducing u. Because $|u| \geq n^2$ and there are only n^2 distinct pairs of states in G, there exist two (not necessarily distinct) states, q_1' and q_2', in $\mathbf{p_1}$ and $\mathbf{p_2}$, respectively, and a partition of $u = xvz$, $v \in \Sigma^+$, such that $\{q_1', q_2'\} \subseteq \delta(\bar{q}, x)$, $q_1' \in \delta(q_1', v)$ and $q_2' \in \delta(q_2', v)$ (there are cycles at q_1' and q_2' inducing v), and, finally, $q_1 \in \delta(q_1', z)$ and $q_2 \in \delta(q_2', z)$. Since q_1' and q_2' are twins, we have that $|c(\bar{q}, u, q_1) - c(\bar{q}, u, q_2)| = |c(\bar{q}, \bar{u}, q_1) - c(\bar{q}, \bar{u}, q_2)|$, where $\bar{u} = xz$ is in Σ^+ and $|\bar{u}| < |u|$. If $\bar{u}$ is of the required length, we are finished; otherwise, we iterate the argument. □

THEOREM 3.8. *Let G be a WFA satisfying the twins property. In the general case, $\#dta(G) < 2^{n(2\log n + n^2 \log|\Sigma| + 1)}$; in the integer (or rational) case, $\#dta(G) < 2^{n(2\log n + 1 + \min(n^2 \log|\Sigma|, \rho))}$; and if G is acyclic, $\#dta(G) < 2^{n \log|\Sigma|}$ independent of any assumptions on weights. The acyclic bound is tight (up to constant factors) for any alphabet.*

Proof. Let $\tilde{R}$ be the set of remainders in $dta(G)$. Each state in $dta(G)$ is an i-tuple of states from G with a corresponding i-tuple of remainders. In the worst case, each i-state tuple from G will appear in $dta(G)$, and there are $|\tilde{R}|^i$ distinct i-tuples of remainders it can assume. (This over-counts by including tuples without any zero remainders.) Therefore,

$$\#dta(G) \leq \sum_{i=1}^{n} \binom{n}{i} |\tilde{R}|^i \leq (2|\tilde{R}|)^n.$$

Let R be the set of remainders r for which the following holds: $\exists w \in \Sigma^*$, $|w| \leq n^2 - 1$, and two states q_1 and q_2, such that $r = |c(\bar{q}, w, q_2) - c(\bar{q}, w, q_1)|$. By Lemma 3.7, $\tilde{R} \subseteq R$, so we can bound $|\tilde{R}|$ in different settings by bounding $|R|$.

General case. The weights on the transitions of G are incommensurable real numbers, i.e., they require "infinite precision" as binary numbers. Since each string induced by G corresponds to at least one path in G, we have by definition of R that the cardinality of this set is bounded by the number of distinct pairs of paths of length at most $n^2 - 1$. There are at most $\sum_{i=1}^{n^2-1} m^i < m^{n^2}$ such paths, where m is the number of edges in G. Therefore $|R| < m^{2n^2}$. On the other hand, the number of strings of length at most $n^2 - 1$ is bounded by $|\Sigma|^{n^2}$. Since each of those strings can reach a pair of (not necessarily distinct) states in G, we have that $|R| < n^2|\Sigma|^{n^2}$. But $|\Sigma| \leq m$, so $n^2|\Sigma|^{n^2}$ is a tighter bound on $|R|$. Our first estimate follows.

Integer case. The weights are nonnegative integers. Fix a state q and a string w that reaches q from the initial state. Then $c(\bar{q}, w, q)$ is in $[0, (n^2-1)C]_Z$. Therefore, the remainders in R must also be in that range. It follows that $(2|R|)^n < (2n^2C)^n = 2^{n(2\log n+\rho+1)}$. Since the topological bound on $|R|$ we derived for the general case does not depend on the magnitude of weights and it holds also for the case we are considering, we have that $(2|R|)^n < 2^{n(2\log n+1+\min(n^2\log|\Sigma|,\rho))}$. Our second estimate follows. Notice that this result also holds for the case in which the weights are rational numbers represented by ρ bits.

Acyclic case. The graph underlying G is acyclic. Thus, each string induced by G is of length at most $n-1$. There are $|\Sigma|^n = 2^{n\log|\Sigma|}$ such strings. Each of the strings induced by G will reach exactly one state in $dta(G)$ (which is a deterministic automaton). Therefore, the number of states of $dta(G)$ is bounded by $2^{n\log|\Sigma|}$. Tightness follows from Theorem 5.10. □

Processing each tuple of state-remainders generated by **DTA** takes $O(|\Sigma|(n+m))$ time, excluding the cost of arithmetic and min operations involving two weights and/or remainders, yielding the following.

THEOREM 3.9. *Let G be a WFA satisfying the twins property.* **DTA** *takes $O(|\Sigma|(n+m)\cdot \#dta(G))$ time using the $\Re$-RAM and $O(\rho|\Sigma|(n+m)\cdot \#dta(G))$ time using the $\mathcal{CO}$-RAM. For the general case, using the $\Re$-RAM, the time is $O(|\Sigma|(n+m)2^{n(2\log n+n^2\log|\Sigma|+1)})$. For the (rational or) integer case, using the $\mathcal{CO}$-RAM, the time is $O(\rho|\Sigma|(n+m)2^{n(2\log n+1+\min(n^2\log|\Sigma|,\rho))})$. For the acyclic case, the time is $O(|\Sigma|(n+m)2^{n\log|\Sigma|})$ using the $\Re$-RAM and $O(\rho|\Sigma|(n+m)2^{n\log|\Sigma|})$ using the $\mathcal{CO}$-RAM.*

Theorems 3.8 and 3.9 do not require G to be unambiguous. **DTA** terminates within the stated resource bounds on any WFA that has the twins property. Consider in the integer case the interplay between the growth of G when determinized, the time complexity of the algorithm, and the magnitude of the weights.

In the acyclic case first, we have that $\#G \le \mathcal{S} \le 2^{n\log|\Sigma|}$, where $\mathcal{S}$ is the number of distinct strings accepted by G. In some sense, $\mathcal{S}$ gives the "expressive power" of G, i.e., how much information is compactly stored in G with the aid of nondeterminism. For small weights, i.e., $\rho \le n\log|\Sigma|$, the worst-case time complexity of the algorithm is dominated by the number of strings accepted by G. Therefore, we can actually "uncompact" some or all of the information contained in G by eliminating nondeterminism. On the other hand, when $\rho > n\log|\Sigma|$, the bigger weights add no information and actually slow down the algorithm to the point that, for very large weights, the arithmetic and logic operations dominate the cost of the entire algorithm.

For the cyclic case, the situation is analogous, with weights playing an even more prominent role. Let $\rho_{max} = n^2\log|\Sigma|$. For $\rho < \rho_{max}$, the estimate of $\#dta(G)$ depends on ρ, although we do not know how tight that estimate is. For $\rho \gg \rho_{max}$, the expansion of G depends only on its topology, but the large weights slow down the algorithm.

3.4. Computing a worst-case weighting. The results of section 3.3 can be used to generate hard instances for any determinization algorithm. Let G be a WFA. A *reweighting function* (or simply *reweighting*) f is such that, when applied to G, it preserves the topology and labeling of G, but possibly changes the weights on its transitions. We want to determine a reweighting f such that $min(f(G))$ exists and $\#min(f(G))$ is maximized among reweightings for which $min(f(G))$ exists. We restrict attention to the integer case and, without loss of generality, we assume that G is trim and unambiguous.

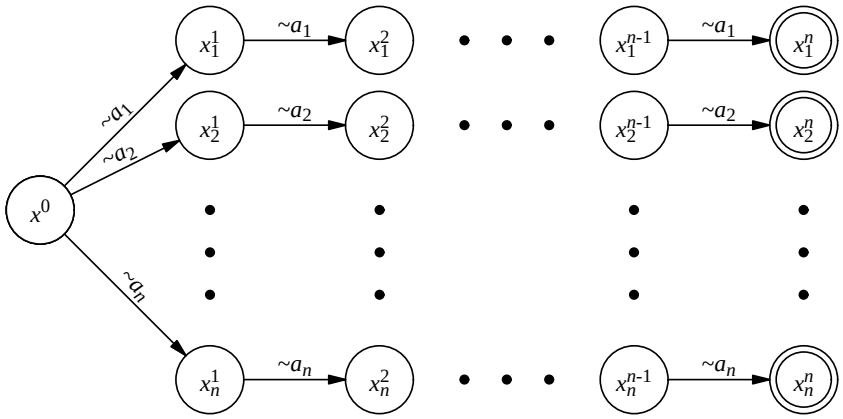


FIG. 3. *A nondeterministic finite-state automaton accepting language* $L = \bigcup_{i=1}^{n} (\Sigma - \{a_i\})^n$. *Arcs labeled* $\tilde{}a_i$ *denote transitions on all symbols in* $\Sigma - \{a_i\}$.

Theorem 3.8 shows that for weights to have an effect on the growth of $dta(G)$, it must be that $\rho \leq n^2 \log |\Sigma|$. Set $\rho_{max} = n^2 \log |\Sigma|$. To find the required reweighting, we simply consider all possible weight assignments to G satisfying the twins property and requiring at most ρ_{max} bits, choosing the one that leads to the minimum deterministic equivalent of maximum size. There are $(2^{\rho_{max}})^m = 2^{m\rho_{max}}$ possible reweightings, and it takes $2^{O(n(2\log n + (n^2 \log |\Sigma|)))}$ time to compute the size expansion or decide that the resulting machine cannot be determinized. The total time is thus bounded by $2^{O(n(2\log n + (n^2 \log |\Sigma|)) + m\rho_{max})}$.

4. Hot automata. This section provides a family of acyclic, multipartite WFAs that are *hot*: when determinized, they expand independently of the weights on their transitions. Given some alphabet $\Sigma = \{a_1, \ldots, a_n\}$, consider the language

$$L = \bigcup_{i=1}^{n} (\Sigma - \{a_i\})^n ;$$

i.e., the set of all n-length strings that do not include all symbols from Σ. It is simple to obtain an acyclic, multipartite NFA H of poly(n) size that accepts L. (See Figure 3.) One can also show that the minimal DFA accepting L has $\Theta(2^{n+\log n})$ states. Furthermore, we can construct H so that these bounds hold for a *binary* alphabet: encode the symbols in Σ as binary strings of length $\log n$, and replace arcs in the above NFA with n-vertex, $(\log n)$-depth trees appropriately. H corresponds to a WFA with all arcs weighted identically. Since acyclic WFAs satisfy the twins property, they can always be determinized. Altering the weights can only increase the expansion.

Continuing, Kintala and Wotschke [18] provide a set of NFAs that produces a hierarchy of expansion factors when determinized. Consider the set of languages

$$L_{h,k} = \{x1y \mid x, y \in \{0,1\}^*; |x| \leq k-1; |y| = k; x \text{ has at most } h \text{ 1's in it}\}$$

for $k \geq 1$, $h < k$. They show that for each $L_{h,k}$, there is an $O(k^2)$-state acyclic (but not multipartite) NFA that accepts $L_{h,k}$, yet any DFA accepting $L_{h,k}$ must have at least $\sum_{i=0}^{2\log h} \binom{k}{i}$ states. These provide additional examples of hot WFAs.

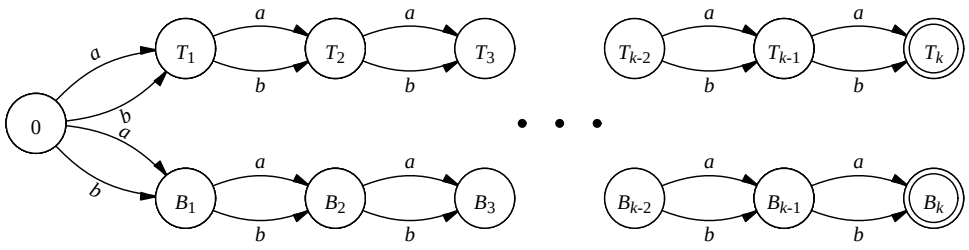


FIG. 4. *Topology of the k-layer rail graph.*

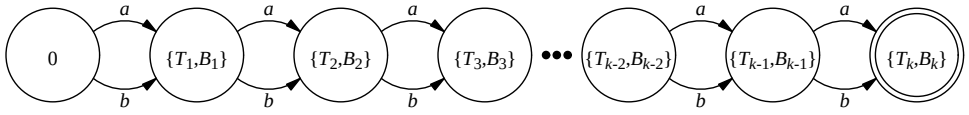


FIG. 5. *The result of determinizing $RG(k)$ when all arc weights are* 0*. In the result, all arcs are again weighted* 0*, and the remainders in the vertices are all* 0*; these values are omitted from the figure.*

5. Weight-dependent automata. In this section we address the effect of weights on the size of the deterministic equivalent of an input WFA. We study a simple family of WFAs with multipartite, acyclic topology. When the arcs are all weighted zero, all WFAs in this family shrink when determinized. We show, however, that even though the topology is by itself very benign, certain weightings can cause exponential increases in size when the WFA is determinized. This study is related in spirit to works that measure amounts of nondeterminism and ambiguity in finite automata [14, 15, 18]. We first discuss the case of a binary alphabet and then generalize to arbitrary alphabets. In this section, we use the terms automaton and graph interchangeably.

5.1. The rail graph. We denote by $RG(k)$ the *k-layer rail graph*. See Figure 4. $RG(k)$ has $2k+1$ vertices, which we denote by $\{0, T_1, B_1, \ldots, T_k, B_k\}$. There are arcs $(0, T_1, a)$, $(0, T_1, b)$, $(0, B_1, a)$, $(0, B_1, b)$, and then, for $1 \leq i < k$, arcs (T_i, T_{i+1}, a), (T_i, T_{i+1}, b), (B_i, B_{i+1}, a), and (B_i, B_{i+1}, b). (It should be clear from Figure 4 that T stands for "top" and B for "bottom.")

Note that $RG(k)$ is $(k+1)$-partite and also has fixed in- and out-degrees. (All vertices have in- and out-degrees 2, except the root, which has in-degree 0 and out-degree 4, and the vertices T_k and B_k, which have out-degree 0.) If we consider the strings induced by paths from 0 to either T_k or B_k, then the language of $RG(k)$ is the set of strings $L_{RG}(k) = \{a, b\}^k$. The only nondeterministic choice is at the state 0, where either the top or bottom rail may be selected. Hence a string w can be accepted by one of two paths: one following the top rail and the other the bottom rail.

Technically, the rail graph is ambiguous. We can easily disambiguate $RG(k)$ by adding transitions from T_k and B_k, each on a distinct symbol, to a new final state. Our results extend to this case. For clarity of presentation, however, we discuss the ambiguous rail graph.

The rail graph is weight-dependent. In section 5.2 we provide weightings such that **DTA** produces a trivial $(k+1)$-vertex series-parallel graph. (See Figure 5 for an example.) On the other hand, in section 5.3 we exhibit weightings for the rail graph such that, when input to **DTA**, we get an exponential increase in the number of states. (See Figures 6 and 7 for an example.) Notice that we cannot get more

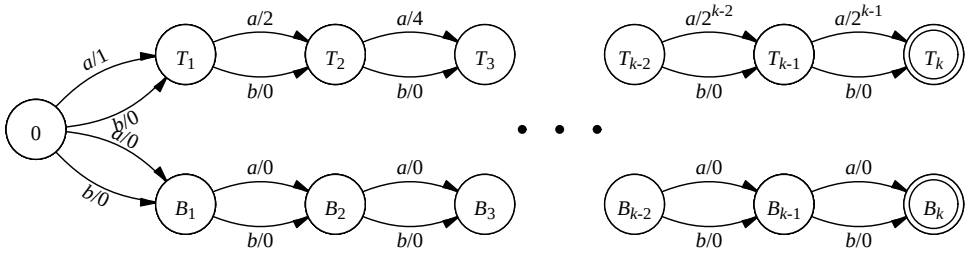


FIG. 6. *"Worst-case" weighting of $RG(k)$. Arc label σ/w means the arc is labeled with symbol σ and has weight w.*

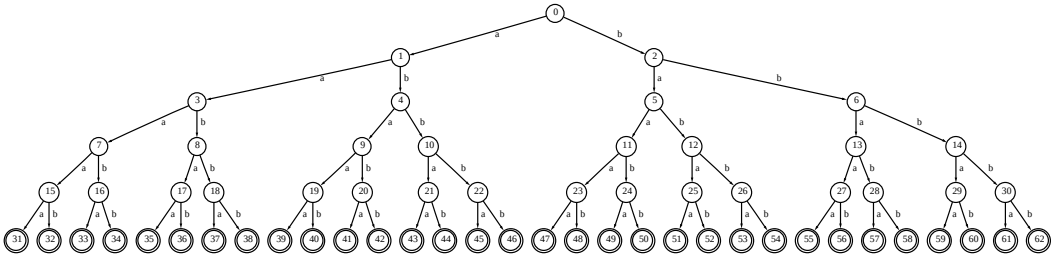


FIG. 7. *Result of determinizing $RG(5)$, weighted as in Figure* 6. *States have been renamed. All arcs are weighted* 0. *The remainders are not shown.*

than 2^k vertices, one per string in $L_{RG}(k)$, in the last layer of the determinized graph, and thus the weighting in Figure 6 is in some sense worst case. In that section we also explore the relationship between the magnitude of the weights and the amount of expansion that is possible. In section 5.4, we show that random weightings induce the behavior of worst-case weightings. We discuss variants of the rail graph in section 5.5, and finally, in section 5.6 we generalize the rail graph to arbitrary alphabets.

5.2. A framework for examining weightings of $RG(k)$. Consider determinizing $RG(k)$ with **DTA**. The set of states reachable on any string $w = \sigma_1 \cdots \sigma_j$ of length $j \leq k$ is $\{T_j, B_j\}$. For a given weighting function c, let $c_T(w)$ denote the cost of accepting string w if the top path is taken; i.e.,

$$c_T(w) = c(0, \sigma_1, T_1) + \sum_{i=1}^{j-1} c(T_i, \sigma_{i+1}, T_{i+1}).$$

Analogously define $c_B(w)$ to be the corresponding cost along the bottom path. Let $R(w)$ be the *remainder vector* for w, which is a pair of the form $(0, c_B(w) - c_T(w))$ or $(c_T(w) - c_B(w), 0)$. A state at layer $0 < i \leq k$ in the determinized WFA is labeled $(\{T_i, B_i\}/R(w))$ for any string w leading to that state; i.e., all strings leading to a particular state induce the same remainder vector. Two strings w_1 and w_2 of identical length lead to distinct states in the determinized version of the rail graph if and only if $R(w_1) \neq R(w_2)$.

It is convenient simply to write $R(w) = c_T(w) - c_B(w)$. The sign of $R(w)$ then determines which of the two forms $(0, x)$ or $(x, 0)$ of the remainder vector occurs.

Suppose that w has length j and can be written $w'\sigma$, where $\sigma \in \{a, b\}$. Let $r_i^T(\sigma)$ denote the weight on the (top) arc labeled σ into vertex T_i and $r_i^B(\sigma)$ denote the weight on the (bottom) arc labeled σ into vertex B_i. Then we can write $R(w) =$

$R(w') + r_j^T(\sigma) - r_j^B(\sigma)$. Abbreviating $r_i^T(\sigma) - r_i^B(\sigma)$ by $\delta_i(\sigma)$, we have

$$R(w) = \sum_{i=1}^{j} \delta_i(\sigma_i).$$

The rail graph with a specific weighting can also be regarded as a function that hashes a k-bit string w into a number $R(w)$. Define symbol a to be 0 and symbol b to be 1, so that a string w can be viewed as a sequence of bits $b_1, \ldots, b_k$. We can write

$$R(b_1, \ldots, b_k) = R(b_1, \ldots, b_{k-1}) + \delta_k(b_k).$$

Also, we can write $\delta_k(b_k) = b_k \cdot \delta_k(1) + (1 - b_k)\delta_k(0)$. Rearranging gives $\delta_k(b_k) = \delta_k(0) + b_k(\delta_k(1) - \delta_k(0))$. Summing over all i gives

$$R(w) = \sum_{i=1}^{k} (\delta_i(0) + b_i(\delta_i(1) - \delta_i(0))).$$

Alternatively,

$$R(w) = R_a + \sum_{i=1}^{k} b_i(\delta_i(1) - \delta_i(0)), \tag{1}$$

where $R_a = \sum_{i=1}^{k} \delta_k(0)$ is fixed for a given weighting function on $RG(k)$.

THEOREM 5.1. *There is a reweighting f such that both dta($f(RG(k))$) and min($f(RG(k))$) realize the topology of the $(k+1)$-vertex trivial series-parallel graph (exemplified in Figure 5).*

Proof. Any weighting in which $\delta_i(a) = \delta_i(b)$ for $i = 1$ to k suffices, since in this case $R(w_1) = R(w_2)$ for all pairs of strings $\{w_1, w_2\}$. In particular, giving zero weights suffices. □

5.3. Worst-case weightings of $RG(k)$. See Figures 6 and 7.

THEOREM 5.2. *For any $j \in [0, k]_Z$ there exists a reweighting f such that dta($f(RG(k))$) has the following form: Layers 0 through j form the complete binary tree on $2^{j+1} - 1$ vertices, and the remaining layers $j+1$ through k consist of trivial series-parallel graphs, each rooted at a leaf of the tree.*

Proof. Choose any weighting such that $\delta_i(a) = 2^{i-1}$ and $\delta_i(b) = 0$ for $1 \le i \le j$, and let $\delta_i(a) = \delta_i(b) = 0$ for $j < i \le k$. Consider a pair of strings w_1, w_2 of identical length that differ in position $i' \le j$. Let $\sigma_i^1 = 1$ if the ith symbol of w_1 is a and $\sigma_i^1 = 0$ otherwise; similarly define σ_i^2 with respect to w_2. Then we can write $R(w_1) = \sum_{i=1}^{j} \sigma_i^1 2^{i-1}$ and $R(w_2) = \sum_{i=1}^{j} \sigma_i^2 2^{i-1}$. Since $\sigma_{i'}^1 \neq \sigma_{i'}^2$, $R(w_1)$ must differ from $R(w_2)$. Hence the two strings must lead to different states. If on the other hand they differ only in positions $i' > j$ they will lead to the same state. There are 2^i strings that differ in positions 1 through i; thus for $i \le j$, there are 2^i distinct vertices in the ith layer of the graph. Since each vertex has out-degree 2, one arc for each symbol, the graph must have the desired form. □

Note that if we set all weights on the bottom rail to zero, and the weights $r_i^T(a) = 2^{i-1}$ and $r_i^T(b) = 0 \; \forall \; 1 \le i \le k$, we get a weighting that yields a complete binary tree of depth k when **DTA** is applied. It is easy to show that the *minimum* deterministic graph preserving shortest paths, however, consists of a trivial series-parallel graph in which all edges have weight zero, corresponding to the lower rail. We can remedy this by choosing weights more judiciously.

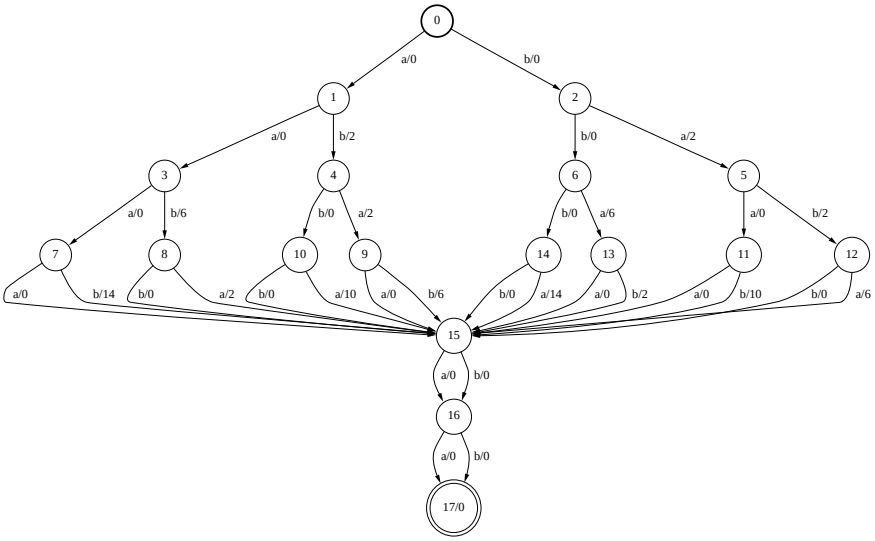


FIG. 8. *Result of minimizing dta*$(RG(6))$*, weighting* $RG(6)$ *as follows:* $r_i^T(a) = r_i^B(b) = 2^i$ *for* $1 \le i \le 4$*; all other weights were* 0*. States have been renamed, and remainders are not shown.*

THEOREM 5.3. *For any* $j \in [0,k]_Z$ *there is a reweighting* f *such that both* $dta(f(RG(k)))$ *and* $min(f(RG(k)))$ *have the following form: Layers* 0 *through* $j-1$ *form the complete binary tree on* $2^j - 1$ *vertices, and the remaining layers* j *through* k *form a trivial series-parallel graph with incoming arcs to the layer-*j *vertex from each vertex at layer* $j-1$.

See Figure 8. Theorem 5.3 is generalized by Theorem 5.10, and therefore we omit its proof here. Theorems 5.2 and 5.3 show that the bound obtained in Theorem 3.8 for the acyclic case is tight for binary alphabets.

We now address the sensitivity of the size expansion to the magnitude of the weights. Note that $RG(k)$ has $2k+1$ vertices and $4k$ arcs, but we use $\Theta(k)$ bits to encode the weight of each arc in the proofs of Theorems 5.2 and 5.3; the input size of the WFA is thus $n = \Theta(k^2)$ bits. The determinized WFA has $2^{k+1} - 1$ states and $2^{k+1} - 2$ transitions. Again we need $\Theta(k)$ bits to encode the weight of each transition, so the bit size of the determinized WFA is $\Theta(k2^k)$, or $2^{\Theta(\sqrt{n})}$ bits. So while the determinized WFA has exponentially more states than the original WFA, the size expansion in bits, while superpolynomial, is not exponential. We argue that exponential state-space expansion requires exponentially big weights for the rail graph.

THEOREM 5.4. *Let* f *be a reweighting. If* $\#dta(f(RG(k))) = \Omega(2^k)$*, then* $\Omega(k^2)$ *bits are required to represent* $f(RG(k))$.

Proof. Consider the $\Omega(2^k)$ vertices at depth k in the determinized graph. Each such state is labeled by a distinct $R(w)$ for some string $w = \sigma_1 \cdots \sigma_k$. Hence if there are $\Omega(2^k)$ states in $dta(f(RG(k)))$, there must be $\Omega(2^k)$ distinct values of $R(w)$. In addition, there must be $\Omega(2^k)$ distinct values of the absolute value of $R(w)$.

Recalling the formulation of $R(w)$ from (1), there can be at most $2^{k'}$ distinct values of $R(w)$, where k' is the number of distinct values of $(\delta_i(1) - \delta_i(0))$. Each value may be included in the sum or not, and at best a choice of inclusions and exclusions will lead to a unique sum. Therefore, the assumption of $\Omega(2^k)$ distinct remainders implies there must be $\Omega(k)$ distinct values of the $(\delta_i(1) - \delta_i(0))$.

Now ignore the $\lfloor \frac{k}{2} \rfloor$ low-order bits of the absolute values of the remainders, and consider only the remaining high-order bits. There must be $\Omega(2^{\lceil \frac{k}{2} \rceil})$ distinct values induced by the high-order bits, or else there cannot be $\Omega(2^k)$ distinct values overall. By the same argument as above, there must be $\Omega(\lceil \frac{k}{2} \rceil)$ distinct values of the $(\delta_i(1) - \delta_i(0))$ that affect the high-order bits of a *large* remainder, i.e., one with one of its high-order $\lceil \frac{k}{2} \rceil$ bits set to 1.

For a particular $(\delta_i(1) - \delta_i(0))$ to affect a high-order bit of a large remainder, $(\delta_i(1) - \delta_i(0))$ must have a nonzero bit at least as high as position $\lceil \frac{k}{2} \rceil - \log k$. This is only true if one of the four arc weights for layer i has a nonzero bit at least that high. Therefore, $\Omega(k)$ arc weights require some nonzero bit at least as high as position $\lceil \frac{k}{2} \rceil - \log k$. Hence $\Omega(k^2)$ bits are required to represent all the arc weights. □

COROLLARY 5.5. *Let f be a reweighting. If $\#min(f(RG(k))) = \Omega(2^k)$, then $\Omega(k^2)$ bits are required to represent $f(RG(k))$.*

Proof. Theorem 5.4 applies, because $\#min(f(RG(k))) = \Omega(2^k)$ implies that $\#dta(f(RG(k))) = \Omega(2^k)$. □

Finally, consider the following analogy between the hot graphs in section 4 and the rail graph. Observe that the hot graphs in section 4 contain some nondeterministic choices that cannot be resolved until the end of the input. This causes the respective deterministic expansions. In those graphs, these choices are part of the strings being accepted. The rail graph manifests this same phenomenon, but in terms of weights rather than strings. The weighted variants of the rail graph that expand when determinized do so because it is not clear until the end of the expansion which rail will provide the shorter path: at any point, the choice of top or bottom rail depends on the symbols that follow. Therefore, the determinization must maintain enough state information to provide for all possible outcomes. Furthermore, in the nonminimizable cases, whereas the language $L_{RG}(k)$ itself could be accepted by a $(k+1)$-state DFA, the weights on $RG(k)$ necessitate an exponential number of states and arcs in any deterministic WFA that induces all the appropriate path lengths.

5.4. Random weightings of $RG(k)$. An *i-bit reweighting function* (or simply *i-bit reweighting*) is a reweighting function f such that the weights on the arcs of $f(G)$ are constrained to be in $[0, 2^i - 1]_Z$. A function f^R is a *random reweighting function* (or simply *random reweighting*) if and only if it chooses the weights to assign to the transitions of G uniformly and independently at random from $\Re^+ \cup \{0, \infty\}$. Finally, let $x \in_R Y$ denote that x is selected uniformly and independently at random from set Y, and let $E[X]$ denote the expected value of some random variable X. We need the following technical claim.

CLAIM 5.6. *Let $X, Y, U, V \in_R [0, 2^k - 1]$. Then*

$$\max_{-2^{k+1}+1 < i < 2^{k+1}-1} \Pr(X - Y - (U - V) = i) \leq \frac{2}{3 \cdot 2^k} + O\left(1/4^k\right).$$

Proof. See Appendix A.1. □

THEOREM 5.7. *Let f^R be a random k-bit reweighting. $E[\#dta(f^R(RG(k)))] = \Theta(2^k)$.*

Proof. As before, let $R(w)$ be the remainder induced by string w of length k; i.e., the difference between the cost of the upper path and the cost of the lower path that respectively induce w. Let $\delta_i(\sigma)$ be the cost of the (top) arc labeled σ into vertex T_i minus the cost of the (bottom) arc labeled σ into vertex B_i. Recall from (1) that the rail graph with a specific weighting can be regarded as a function that hashes a k-bit string $w = \sigma_1 \cdots \sigma_k$ into a number $R(w)$.

Suppose $w_1 \neq w_2$. We can adapt a standard analysis from the theory of universal hash functions [7] to calculate the probability that $R(w_1) = R(w_2)$. Let $w_1 = \alpha_1 \cdots \alpha_k$ and $w_2 = \beta_1 \cdots \beta_k$. Without loss of generality, assume $\alpha_k \neq \beta_k$. (The strings must differ somewhere.) Suppose $R(w_1) = R(w_2)$. Then by definition (with some manipulation)

$$(\alpha_k - \beta_k)(\delta_k(1) - \delta_k(0)) = \sum_{i=1}^{k-1} (\beta_i - \alpha_i)(\delta_i(1) - \delta_i(0)).$$

Fix a set of weights on the arcs in the first $k-1$ layers, so the right-hand side of the equation is a constant. The value $(\alpha_k - \beta_k)$ is either -1 or 1, by assumption. Hence, for the given set of weights on the first $k-1$ layers, there is exactly one value of $(\delta_k(1) - \delta_k(0))$ that makes $R(w_1) = R(w_2)$. If we assign the weights randomly in the kth layer, the probability that $(\delta_k(1) - \delta_k(0))$ has the necessary value is upper-bounded by

$$\max_{-2^{k+1}+1 < C < 2^{k+1}-1} \Pr[X_1 - X_2 - (X_3 - X_4) = C],$$

where the X_is are uniform random variables in the range $[0, 2^k - 1]_Z$. (It is strictly upper-bounded, since the value of $(\delta_k(1) - \delta_k(0))$ lies in $[-2^{k+1} + 2, 2^{k+1} - 2]$, which is in general smaller than the range of values that the right-hand side of the equation can assume.)

By Claim 5.6, therefore, for any fixed assignment of the first $k-1$ values, the probability that $R(w_1) = R(w_2)$ is bounded above by $\frac{2}{3 \cdot 2^k} + O(1/4^k)$. Hence, summing over all possible assignments to the first $k-1$ values, the total probability that $R(w_1) = R(w_2)$ is also bounded above by $\frac{2}{3 \cdot 2^k} + O(1/4^k)$. We will use this result to compute the expected number of vertices at layer k in the determinized graph.

First, we compute the expected number of *collisions*: the number of strings of length k that have the same remainder. Since the number of strings is 2^k, this is simply

$$\sum_{w_1, w_2} \left(\frac{2}{3 \cdot 2^k} + O(1/4^k) \right) = \frac{2}{3} 2^k + O(1).$$

Suppose $x_1, \ldots, x_\ell$ form the set of layer-k vertices in the deterministic graph. Let S_i be the set of strings that map to vertex x_i, and let random variable s_i be $|S_i|$. The expected number of collisions can thus be written as $E[\sum_{i=1}^{\ell} \binom{s_i}{2}]$. Since the expected value is $\frac{2}{3} 2^k + O(1)$, it must be the case that in one-half of the weight assignments, the actual value of the sum is at most $\frac{4}{3} 2^k + O(1)$.

We now consider the minimum number of sets so that there is an assignment of strings to sets that satisfies

$$\sum_{i=1}^{\ell} \binom{s_i}{2} \leq \frac{4}{3} 2^k + O(1)$$

and

$$\sum_{i=1}^{\ell} s_i = 2^k.$$

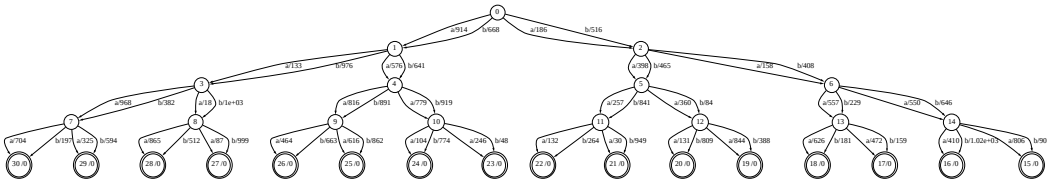


FIG. 9. *A rail tree of depth* 4*, with ten-bit random weights.*

Elementary calculus shows that the binomial sum is minimized when the s_i's are all equal, which implies that the minimal value of ℓ is $\Omega(2^k)$.

Therefore, in at least half the weight assignments, there are $\Omega(2^k)$ distinct remainders and so $\Omega(2^k)$ distinct states in the determinized graph. Hence the expected number of states in the determinized graph is also $\Omega(2^k)$. □

Since the arcs weights of ASR WFAs are (negated) log probabilities, we consider the behavior of determinization when the arcs are weighted with *log-random* numbers, i.e., logarithms of uniform random variables. On the rail graph, we find the same behavior with log-random weights as with random weights, including the increasing expansion as more bits are employed. As the following claim shows, retaining enough bits of the log-random weights drives the collision probability (as discussed in the proof of Theorem 5.7) low enough to ensure this behavior.

CLAIM 5.8. *Let* $X, Y, V, Z \in_R [1, 2^k - 1]$*. Let* $R = \lfloor 2^b \log X \rfloor$*,* $S = \lfloor 2^b \log Y \rfloor$*,* $T = \lfloor 2^b \log V \rfloor$*, and* $U = \lfloor 2^b \log Z \rfloor$ *for some* $b \geq k - O(1)$*. Then*

$$\max_{-k2^{b+1}+1<i<k2^{b+1}-1} \Pr(R - S - (T - U) = i) = \frac{1}{4 \cdot 2^k} + O\left(1/4^k\right).$$

Proof. See Appendix A.2. □

Thus we derive the analogue to Theorem 5.7 for log-random weights.

THEOREM 5.9. *Let* G *be* $RG(k)$ *weighted with logarithms of numbers chosen independently and uniformly at random from* $[1, 2^k - 1]_Z$*. Then* $E[\#dta(G)] = \Theta(2^k)$*.*

Proof. Apply Claim 5.8 instead of Claim 5.6 in the proof of Theorem 5.7, and alter the ensuing discussion appropriately to account for the respective constants. □

5.5. Variations of the rail graph. We remark that all of the above results extend even when we simplify the rail graph as follows.

1. Eliminate the b-arcs out of state 0.
2. Allow the symbols on the arcs at any layer i to differ from a and b. Enforce only that the symbols on the top rail at layer i be the same as those on the bottom rail at layer i.

Using the original definition of $RG(k)$ simplifies the presentation of the theorems and proofs.

Although the rail graph allows us to investigate the relationship between deterministic expansion and weighting in WFAs, the general problem of characterizing this relationship seems difficult. Consider, for example, a *rail tree*, as in Figure 9, in which the rail graph is extended into a tree in a straightforward manner. As shown in Figure 10, when determinized, the resulting tree has the same number of vertices and half the arcs of the original. This is because the language of the original tree contains only 2^4 distinct strings, and so the determinized tree can have only 2^4 leaves. So while many individual rail graphs are embedded in the rail tree, when determinized, it is as if only one expands at the expense of the others.

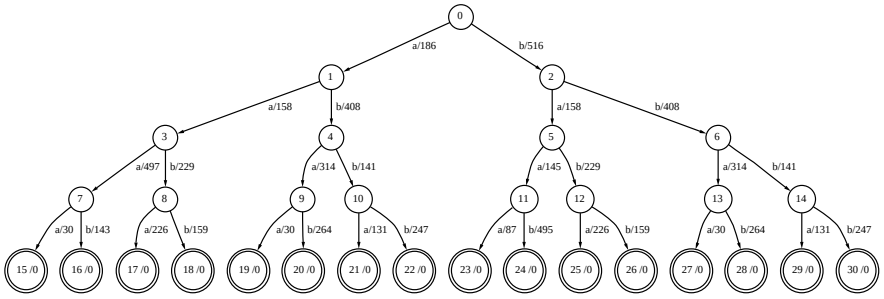


FIG. 10. *The result of determinizing the rail tree of Figure* 10.

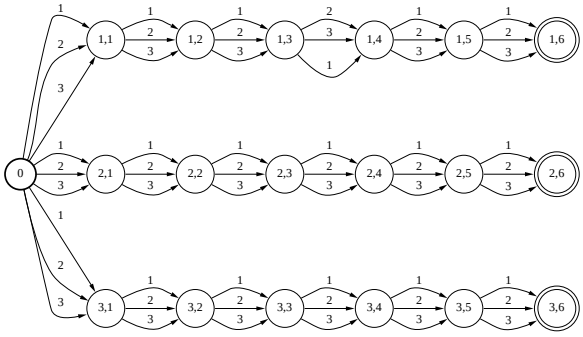


FIG. 11. *Topology of* $RG(3,6)$.

5.6. Extending $RG(k)$ to arbitrary alphabets. We can extend the rail graph to arbitrary alphabets, defining $RG(r,k)$, the *k-layer r-rail graph*, as follows. $RG(r,k)$ has $rk+1$ vertices: vertex 0 and, for $1 \le i \le r$ and $1 \le j \le k$, vertex v_j^i. Assume the alphabet is $\{1,\dots,r\}$. $RG(r,k)$ has arcs $(0, v_1^i, s)\ \forall\ 1 \le i,s \le r$ and also arcs $(v_j^i, v_{j+1}^i, s)\ \forall\ 1 \le i,s \le r$ and $1 \le j < k$. See Figure 11.

The subgraph induced by vertex 0 and vertices v_j^i for some i and $\forall\ 1 \le j \le k$ comprises *rail* i of $RG(r,k)$. The subgraph induced by vertices $v_j^i\ \forall\ 1 \le i \le r$ and some j comprises *layer* j of $RG(r,k)$. Vertex 0 comprises *layer* 0 of $RG(r,k)$. Thus, $RG(2,k)$ is the k-layer rail graph, $RG(k)$, defined in section 5.1.

Denote by $c(i,j,s)$ the weight of the arc labeled s into vertex v_j^i. Theorem 5.3 generalizes to $RG(r,k)$ as follows. (See Figure 12.)

THEOREM 5.10. *For any $j \in [0,k]_Z$ there is an assignment of weights such that the minimal deterministic realization of $RG(r,k)$ has the following form: Layers* 0 *through $j-1$ form the complete r-ary tree on $\frac{r^j-1}{r-1}$ vertices, and the remaining layers j through k form a trivial series-parallel graph with incoming arcs to the layer-j vertex from each vertex at layer $j-1$.*

Proof. Choose the following weighting. Set $c(i,\ell,s) = [(i+s) \mod r] \cdot r^\ell\ \forall\ 1 \le i,s \le r$ and $1 \le \ell \le j$. Set $c(i,\ell,s) = 0\ \forall\ 1 \le i,s \le r$ and $j < \ell \le k$.

We show that there are no two strings of length less than j that lead to the same vertex in *any* deterministic realization of the graph that preserves shortest paths. Hence the number of vertices at layer $\ell < j$ is equal to the number of strings of length

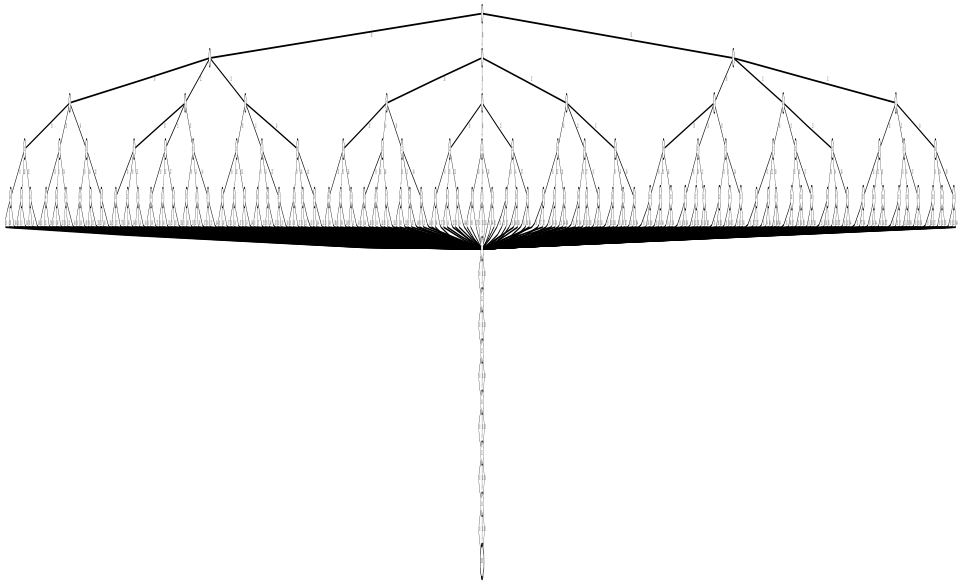


FIG. 12. *Result of minimizing dta($RG(3,10)$), weighting $RG(3,10)$ as follows.* $c(i,\ell,s) = [(i+s) \bmod r] \cdot r^\ell \ \forall\ 1 \le i,s \le 3$ *and* $1 \le \ell \le 4$. $c(i,\ell,s) = 0 \ \forall\ 1 \le i,s \le 3$ *and* $4 < \ell \le 10$. *Only topology is shown.*

ℓ, which is r^ℓ. The proof is by contradiction. Suppose that strings $w_1 \neq w_2$ lead to the same vertex x. Let $\ell < j$ be the length of strings w_1 and w_2. Now consider all strings that have w_1 or w_2 as a prefix. Let $\{w'_1, w'_2, \ldots, w'_m\}$ be the set of possible suffixes, of length $k - \ell$, that can follow any of these prefixes. (In our case, this is simply the set $\{1, \ldots, r\}^{k-\ell}$.)

Let $c(w_1 w'_i)$ and $c(w_2 w'_i)$ denote the costs of strings $w_1 w'_i$ and $w_2 w'_i$, respectively, in the nondeterministic graph, i.e., the sums of the costs of the arcs along the shortest paths inducing the respective strings in $RG(r,k)$. The deterministic graph must satisfy $c(w_1 w'_i) = c_d(w_1) + c_d(w'_i|x)$ and $c(w_2 w'_i) = c_d(w_2) + c_d(w'_i|x)\ \forall\ 1 \le i \le m$, where c_d denotes the cost function in the deterministic graph, and $c_d(w'_i|x)$ denotes the cost of string w'_i starting from vertex x. Hence we have that $\forall\ i$, $c(w_1 w'_i) - c(w_2 w'_i) = c_d(w_1) - c_d(w_2)$. The right-hand side of this equation is a fixed value, say, Δ.

Now consider the pair of suffixes $x\sigma_1 y$ and $x\sigma_2 y$, such that $\sigma_1 \neq \sigma_2$ and, for $i \in \{1,2\}$, $|\sigma_i| = 1$, $|x\sigma_i| = j - \ell$, and $|x\sigma_i y| = k - \ell$; i.e., $x\sigma_i y$ appended to w_1 or w_2 results in a string of length k with σ_i as the jth symbol. We claim that there exists some choice of $\sigma_1, \sigma_2 \in \{1, \ldots, r\}$ such that $c(w_1 x\sigma_1 y) - c(w_2 x\sigma_1 y) \neq c(w_1 x\sigma_2 y) - c(w_2 x\sigma_2 y)$. This contradicts the requirement that both differences should be equal to Δ and hence proves the theorem.

To see the claim, observe that the given weighting on $RG(r,k)$ forces the minimum cost path for any string with some symbol σ in position j to follow rail $(r - \sigma)$. Consider any position $i \le \ell$ in which w_1 and w_2 differ. Denote the ith symbol of w_1 (resp., w_2) by $w_1(i)$ (resp., $w_2(i)$). Then that position contributes $[(w_1(i) + r - \sigma_1) \bmod r - (w_2(i) + r - \sigma_1) \bmod r] \cdot r^i$ to the difference $\Delta_1 = c(w_1 x\sigma_1 y) - c(w_2 x\sigma_1 y)$ and $[(w_1(i) + r - \sigma_2) \bmod r - (w_2(i) + r - \sigma_2) \bmod r] \cdot r^i$ to the difference $\Delta_2 = c(w_1 x\sigma_2 y) - c(w_2 x\sigma_2 y)$. Picking $\sigma_1 = w_1(i)$ and $\sigma_2 = w_2(i)$ implies that position i contributes $\delta_1 = -[(w_2(i) - w_1(i)) \bmod r] \cdot r^i$ to Δ_1 and $\delta_2 = [(w_1(i) - w_2(i))$

mod $r] \cdot r^i$ to Δ_2. Δ_1 and Δ_2 are sums of the form $\sum_{i=1}^{j} c_i r^i$, where $-r < c_i < r$. For some $1 \leq c_1, c_2 < r$, $\delta_1 = -c_1 r^i$ and $\delta_2 = c_2 r^i$ are the only terms involving r^i in Δ_1 and Δ_2, respectively. It follows that $\Delta_1 \neq \Delta_2$, thereby proving the claim. □

We point out that Theorem 5.10 shows that the bounds in Theorem 3.8 for the acyclic case are tight for general alphabets. We also point out that Theorems 5.1 and 5.2 generalize easily to the k-layer r-rail graphs. As for the remaining results stated for binary alphabets in sections 5.3 and 5.4, it would be of some interest to extend them to arbitrary alphabets: while those extensions would not change the "nature" of the lower bound stated in Theorem 5.4, they might shed some more light on the relationship between hashing and **DTA**.

6. Experimental observations on ASR WFAs. In this section, we study whether the WFAs that are generated by ASR systems are weight-dependent. This possibility had not previously been considered by the ASR community, and if true, it suggests that the weights of ASR WFAs, which are generated from training sets with considerable manual intervention, must be carefully monitored to preserve their favorable characteristics.

6.1. Data. We experimented on 100 WFAs generated by the AT&T North American Business speech recognizer [27], using a grammar for the Air Travel Information System (ATIS), a standard 5000-word vocabulary DARPA test bed [4]. Each transition was labeled with a word from the ATIS vocabulary and weighted by the recognizer with the negated log of the probability of realizing that transition out of the source state; we refer to these weights as *speech weights*.

6.2. Method. For each input WFA, we varied the weights on the given topology. We determinized each with its speech weights, with zero weights, and with weights assigned independently and uniformly at random from the integral range $[0, 2^i - 1]$ (for each $0 \leq i \leq 8$). One WFA could not be determinized with speech weights due to computational limitations, and it is omitted from the data. The experiments were run on an SGI R4400 processor with 1 GB of main memory and an SGI R10000 processor with 1.5 GB of main memory.

Figure 13 shows how many WFAs expanded when determinized with different weightings. Figure 14 classifies the 63 WFAs that expanded with at least one weighting. For each WFA, we took the weighting that produced maximal expansion. This was usually the 8-bit random weighting, although due to computational limitations we were unable to determinize some WFAs with large random weightings. The x-axis indicates the open intervals within which the value $\log(|dta(G)|/|G|)$ falls.

Figure 15 provides additional classification of the eighteen WFAs that expanded when determinized with speech weights. The x-axis indicates the open intervals within which the ratio $|dta(G)|/|G|$ falls when G is determinized with its speech weights.

Since the utility of determinization in ASR includes the reduction in size achieved with actual speech weights, we provide some data on how much the WFAs shrink. In our sample, 82 WFAs shrank when determinized. For each of these WFAs, we computed the value $\log(|G|/|dta(G)|)$. In Figure 16, we plot the number of WFAs whose corresponding values fell in various ranges.

Figure 17 demonstrates the relationship between the (log of the) expansion ratio $|dta(G)|/|G|$ and the number of bits used in the random weights for the ten WFAs with highest final expansion value. For reference the functions i^2, $2^{\sqrt{i}}$, and 2^i are plotted, where i is the number of bits. Most of the WFAs exhibit subexponential growth as the number of bits increases, although some, like `q0t063`, have increased

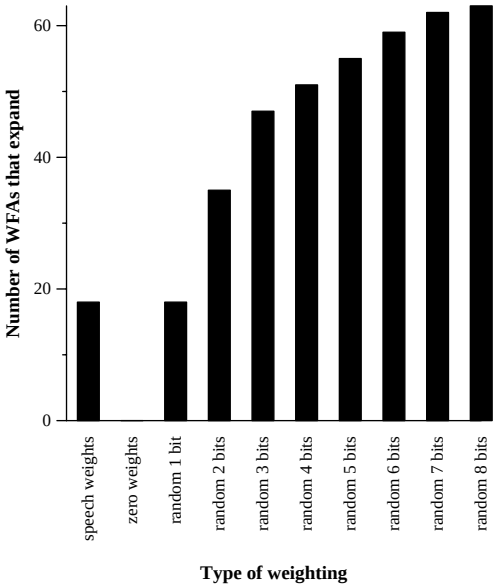


FIG. 13. *Number of WFAs that expanded when determinized with various weightings.*

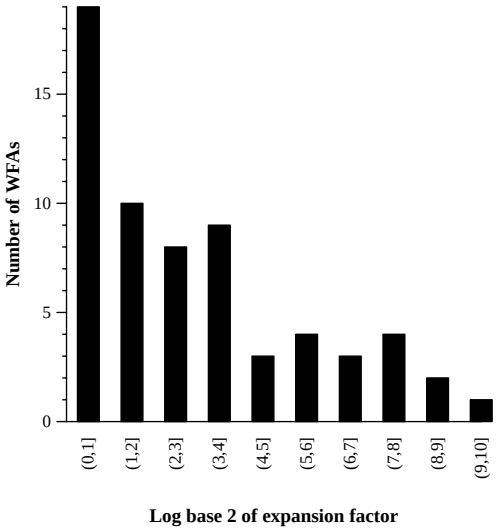


FIG. 14. *Histogram of expansion factors.*

by 128 times even with four random bits.

6.3. Discussion. The WFA that could not be determinized with speech weights was "slightly hot," in that the determinized zero-weighted variant had 2.7% more arcs than the original WFA. The remaining 99 WFAs shrank with zero weights: none was hot. If one expanded, it did so due to weights rather than topology.

Figure 13 indicates that many of the WFAs have some degree of weight-dependence, since 63 expanded when sufficiently large random weights were used. Finally, Figure 17 suggests that random weights are a good way to estimate the degree to which

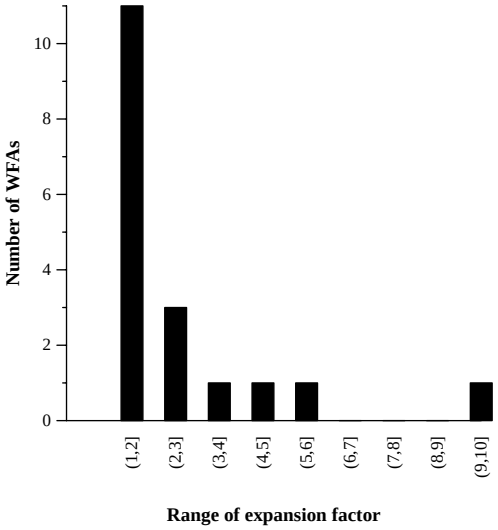


FIG. 15. *Expansion factors for WFAs that expanded when determinized with speech weights.*

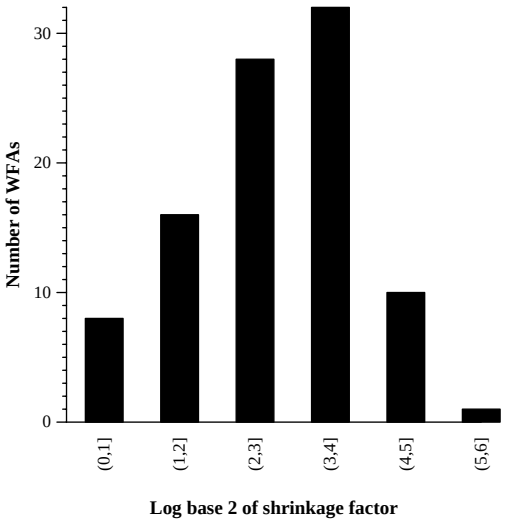


FIG. 16. *Shrinkage factors for WFAs that shrank when determinized with at least one weighting.*

a WFA is weight-dependent. Note that the expansion factor is some superlinear, possibly exponential, function of the number of random bits. This suggests that using large random weights, 32 bits, for example, should cause expansion if anything will. Analogous experiments on the minimized determinized WFAs yield results that are qualitatively the same, although fewer WFAs still expand after minimization. Hence weight-dependence seems to be a fundamental property of these WFAs rather than an artifact of this particular determinization algorithm.

We have used the analyses and experimental observations above to design an *approximate* WFA determinization algorithm: a variant of Mohri's algorithm that unifies state tuples whose remainders differ within a specified relative factor. Using this algorithm, we achieve size reductions in ASR language models that significantly

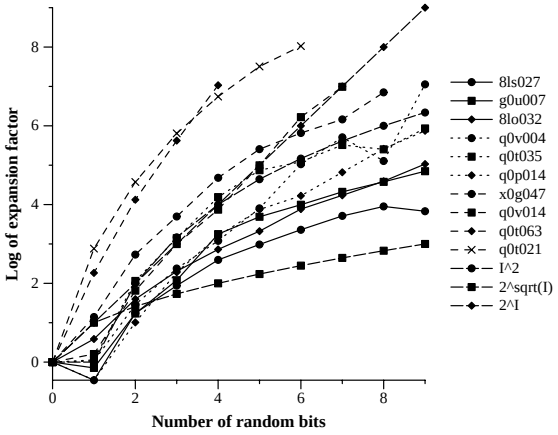


FIG. 17. *Relationship between expansion ratio and number of bits.*

exceed those of previous methods, with negligible effects on ASR performance (time and accuracy). We detail the algorithm and these results separately [6].

The experimental results show that ASR WFAs are generally not hot. This suggests the following open problem: characterize a class of WFAs that are not hot and to which many of the ASR examples belong. Such a characterization might be useful in generating ASR WFAs, so that hot WFAs are guaranteed not to occur.

Appendix. Proofs of claims.

LEMMA A.1. *Let Y_1 and Y_2 be random variables from the same probability distribution. Let X be a random variable such that*

$$\Pr(X = i) = \sum_{j=B+|i|}^{T} \Pr(Y_1 = j) \Pr(Y_2 = j - |i|)$$

for any i and some fixed B and T. Then $\Pr(X = i) \leq \Pr(X = 0)$ for any i.

Proof. Using the fact $ab \leq \frac{1}{2}\left(a^2 + b^2\right)$, we derive

$$\begin{aligned}
\Pr(X = i) &= \sum_{j=B+|i|}^{T} \Pr(Y_1 = j) \Pr(Y_2 = j - |i|) \\
&\leq \frac{1}{2} \sum_{j=B+|i|}^{T} \left(\Pr(Y_1 = j)^2 + \Pr(Y_2 = j - |i|)^2\right) \\
&= \frac{1}{2} \sum_{j=B+|i|}^{T} \Pr(Y_1 = j)^2 + \frac{1}{2} \sum_{j=B+|i|}^{T} \Pr(Y_2 = j - |i|)^2. \qquad (2)
\end{aligned}$$

By definition, $\Pr(X = 0) = \sum_{j=B}^{T} \Pr(Y_1 = j) \Pr(Y_2 = j)$. By symmetry, therefore,

$$\Pr(X = 0) = \sum_{j=B}^{T} \Pr(Y_1 = j)^2 = \sum_{j=B}^{T} \Pr(Y_2 = j)^2,$$

which, combined with (2), completes the proof. □

A.1. Proof of Claim 5.6. It should be clear that, for $-(2^k-1) \le i \le 2^k-1$,

$$\Pr(X-Y=i) = \Pr(U-V=i) = \left(\frac{1}{2^k}\right)^2 \left(2^k - |i|\right).$$

Consider $\Pr(X-Y-(U-V)=i)$ and the following cases.

$$\Pr(X-Y-(U-V)=i \mid i \ge 0) = \sum_{j=-2^k+1+i}^{2^k-1} \Pr(X-Y=j)\Pr(U-V=j-i).$$

$$\Pr(X-Y-(U-V)=i \mid i < 0) = \sum_{j=-2^k+1-i}^{2^k-1} \Pr(X-Y=j+i)\Pr(U-V=j).$$

Unifying the cases and exploiting symmetry yields

$$\Pr(X-Y-(U-V)=i) = \sum_{j=-2^k+1+|i|}^{2^k-1} \Pr\left(X-Y=j\right)\Pr\left(U-V=j-|i|\right).$$

By Lemma A.1, it suffices to solve for $i=0$.

$$\begin{aligned}
\Pr(X-Y-(U-V)=0) &= \sum_{j=-2^k+1}^{2^k-1} \Pr(X-Y=j)\Pr(U-V=j) \\
&= \sum_{j=-2^k+1}^{2^k-1} \left(\frac{1}{2^k}\right)^4 \left(2^k-|j|\right)^2 \\
&= \left(\frac{1}{2^k}\right)^4 \left[\left(2^k\right)^2 + 2\sum_{j=1}^{2^k-1}\left(2^k-j\right)^2\right] \\
&= \left(\frac{1}{2^k}\right)^4 \left[\left(2^k\right)^2 + 2\sum_{j=1}^{2^k-1} j^2\right] \\
&= \left(\frac{1}{2^k}\right)^4 \left[\left(2^k\right)^2 + 2\left(\frac{\left(2^k-1\right)\left(2^k\right)\left(2^{k+1}+1\right)}{6}\right)\right] \\
&\approx \left(\frac{1}{2^k}\right)^4 \left[\left(2^k\right)^2 + \frac{2}{3}\left(2^k\right)^3\right] \\
&= \frac{2}{3\cdot 2^k} + O\left(1/4^k\right). \qquad \square
\end{aligned}$$

A.2. Proof of Claim 5.8. For $0 \le i \le k2^b - 1$, $\Pr(R=i) = \Pr(S=i) = \Pr(T=i) = \Pr(U=i)$, and

$$\begin{aligned}
\Pr(R=i) &= \Pr\left(i \le 2^b \log Z < i+1\right) \\
&= \Pr\left(2^{i/2^b} \le Z < 2^{(i+1)/2^b}\right) \\
&= \frac{1}{2^k-1}\left[2^{i/2^b}\left(2^{1/2^b}-1\right)\right].
\end{aligned}$$

Now consider $\Pr(R - S = i)$ (which equals $\Pr(T - U = i)$). Exploiting symmetry as before, we derive that, for $-k2^b + 1 \le i \le k2^b - 1$,

$$\begin{aligned}
\Pr(R - S = i) &= \sum_{j=|i|}^{k2^b-1} \Pr(R = j) \Pr(S = j - |i|) \\
&= \sum_{j=|i|}^{k2^b-1} \left[\frac{1}{2^k - 1}\left(2^{1/2^b} - 1\right)\right]^2 2^{j/2^b} 2^{(j-|i|)/2^b} \\
&= \left[\frac{1}{2^k - 1}\left(2^{1/2^b} - 1\right)\right]^2 \frac{1}{2^{|i|/2^b}} \sum_{j=|i|}^{k2^b-1} 2^{2j/2^b} \\
&= \left[\frac{1}{2^k - 1}\left(2^{1/2^b} - 1\right)\right]^2 \frac{1}{2^{|i|/2^b}} \left[\frac{\left(2^{2/2^b}\right)^{k2^b} - 1}{2^{2/2^b} - 1} - \frac{\left(2^{2/2^b}\right)^{|i|} - 1}{2^{2/2^b} - 1}\right] \\
&= \left[\frac{1}{2^k - 1}\left(2^{1/2^b} - 1\right)\right]^2 \frac{1}{2^{|i|/2^b}} \left(\frac{2^{2k} - 2^{2|i|/2^b}}{2^{2/2^b} - 1}\right) \\
&= \frac{1}{(2^k - 1)^2} \cdot \frac{2^{2k} - 2^{2|i|/2^b}}{2^{|i|/2^b}} \cdot \frac{2^{1/2^b} - 1}{2^{1/2^b} + 1}.
\end{aligned}$$

By Lemma A.1, $\Pr(R - S = i)$ maximizes at $i = 0$.

$$\Pr(R - S = 0) = \frac{\left(2^k + 1\right)\left(2^{1/2^b} - 1\right)}{\left(2^k - 1\right)\left(2^{1/2^b} + 1\right)}.$$

As k gets large, $\frac{2^k+1}{2^k-1}$ tends to 1. We also have $2 < 2^{1/2^b} + 1 \le 3$. Finally, $2^{1/2^b} - 1 \le 1/2^k$ as long as $b \ge k - O(1)$. Thus,

$$\max_{-k2^b+1 \le i \le k2^b-1} \Pr(R - S = i) \approx \frac{1}{2 \cdot 2^k}.$$

Now consider $\Pr(R - S - (T - U) = i)$. As before, by symmetry we derive that

$$\Pr(R - S - (T - U) = i) = \sum_{j=-k2^b+1+|i|}^{k2^b-1} \Pr(R - S = j) \Pr(T - U = j - |i|)$$

for $-k2^{b+1} + 1 < i < k2^{b+1} - 1$. By Lemma A.1, it suffices to solve for $i = 0$.

$$\Pr(R-S-(T-U)=0)=\left[\frac{2^{1/2^b}-1}{(2^k-1)^2\left(2^{1/2^b}+1\right)}\right]^2\sum_{j=-k2^b+1}^{k2^b-1}\left(\frac{2^{2k}-2^{2|j|/2^b}}{2^{|j|/2^b}}\right)^2$$

$$<\left[\frac{2^{1/2^b}-1}{(2^k-1)^2\left(2^{1/2^b}+1\right)}\right]^2\left[\left(2^{2k}-1\right)^2+2\sum_{j=1}^{k2^b-1}\left(\frac{2^{4k}}{2^{2j/2^b}}+2^{2j/2^b}\right)\right]$$

$$=\frac{\left(2^{1/2^b}-1\right)^2\left(2^{2k}-1\right)^2}{(2^k-1)^4\left(2^{1/2^b}+1\right)^2}+\frac{2\left(2^{1/2^b}-1\right)^2}{(2^k-1)^4\left(2^{1/2^b}+1\right)^2}\left[2^{4k}\sum_{j=1}^{k2^b-1}\frac{1}{2^{2j/2^b}}+\sum_{j=1}^{k2^b-1}2^{2j/2^b}\right].$$

(3)

$$2^{4k}\sum_{j=1}^{k2^b-1}\frac{1}{2^{2j/2^b}}=2^{4k}\left[\frac{\left(\frac{1}{2^{2/2^b}}\right)^{k2^b}-1}{\frac{1}{2^{2/2^b}}-1}-1\right]$$

$$=2^{4k}\left[\frac{1/2^{2k}-1}{\frac{1}{2^{2/2^b}}-1}-1\right]<\frac{2^{4k}}{\left(2^{1/2^b}+1\right)\left(2^{1/2^b}-1\right)}. \tag{4}$$

$$\sum_{j=1}^{k2^b-1}2^{2j/2^b}<\frac{\left(2^{2/2^b}\right)^{k2^b}-1}{2^{2/2^b}-1}$$

$$=\frac{2^{2k}-1}{2^{2/2^b}-1}=\frac{\left(2^k+1\right)\left(2^k-1\right)}{\left(2^{1/2^b}+1\right)\left(2^{1/2^b}-1\right)}. \tag{5}$$

Combining (3)–(5) yields

$$\Pr(R-S-(T-U)=0)<\frac{\left(2^{1/2^b}-1\right)^2\left(2^{2k}-1\right)^2}{(2^k-1)^4\left(2^{1/2^b}+1\right)^2}$$

$$+\frac{2\left(2^{1/2^b}-1\right)^2}{(2^k-1)^4\left(2^{1/2^b}+1\right)^2}\left[\frac{2^{4k}}{\left(2^{1/2^b}+1\right)\left(2^{1/2^b}-1\right)}+\frac{\left(2^k+1\right)\left(2^k-1\right)}{\left(2^{1/2^b}+1\right)\left(2^{1/2^b}-1\right)}\right]$$

$$=\underbrace{\frac{\left(2^{1/2^b}-1\right)^2\left(2^{2k}-1\right)^2}{(2^k-1)^4\left(2^{1/2^b}+1\right)^2}}_{A}+\underbrace{\frac{2\cdot2^{4k}\left(2^{1/2^b}-1\right)}{(2^k-1)^4\left(2^{1/2^b}+1\right)^3}}_{B}+\underbrace{\frac{2\left(2^k+1\right)\left(2^{1/2^b}-1\right)}{(2^k-1)^3\left(2^{1/2^b}+1\right)^3}}_{C}.$$

Recall $2\le 2^{1/2^b}+1<3$ and $2^{1/2^b}-1\le 1/2^k$ as long as $b\ge k-O(1)$. Thus,

$$\max\{A\}\approx\frac{1}{4}\left(\frac{1}{2^k}\right)^2;$$

$$\max\{B\}\approx\frac{1}{4}\cdot\frac{1}{2^k};$$

$$\max\{C\}\approx\frac{1}{4}\left(\frac{1}{2^k}\right)^3. \qquad \square$$

Acknowledgments. We thank Mehryar Mohri, Fernando Pereira, and Antonio Restivo for fruitful discussions. Additionally, Fernando provided us with the WFAs we used in our experiments.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice–Hall, Englewood Cliffs, NJ, 1993.
[2] J. Berstel, *Transduction and Context-Free Languages*, Leitfäden Angew. Math. Mech., 38, Springer-Verlag, Berlin, 1979.
[3] J. Berstel and C. Reutenauer, *Rational Series and Their Languages*, EATCS Monographs on Theoretical Computer Science 12, Springer-Verlag, Berlin, 1988.
[4] E. Bocchieri, G. Riccardi, and J. Anantharaman, *The* 1994 *AT&T ATIS CHRONUS recognizer*, in Proceedings of the ARPA Spoken Language Technology Workshop, Austin, TX, Morgan-Kaufmann, 1995, pp. 265–268.
[5] A. L. Buchsbaum and R. Giancarlo, *Algorithmic aspects in speech recognition: An introduction*, ACM J. Exp. Algorithmics, 2 (1997), http://www.jea.acm.org.
[6] A. L. Buchsbaum, R. Giancarlo, and J. R. Westbrook, *Shrinking language models by robust approximation*, in Proceedings of IEEE Int'l. Conf. on Acoustics, Speech, and Signal Processing '98, Vol. 2, Seattle, WA, 1998, pp. 685–688. To appear in *Algorithmica* as, "An approximate determinization algorithm for weighted finite-state automata."
[7] J. L. Carter and M. N. Wegman, *Universal classes of hash functions*, J. Comput. System Sci., 18 (1979), pp. 143–154.
[8] C. Choffrut, *Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles*, Theoret. Comput. Sci., 5 (1977), pp. 325–337.
[9] C. Choffrut, *Contributions á l'étude de quelques familles remarquables de function rationnelles*, Ph.D. thesis, LITP-Université Paris 7, Paris, France, 1978.
[10] K. Culik II and J. Karhumäki, *Finite automata computing real functions*, SIAM J. Comput., 23 (1994), pp. 789–814.
[11] K. Culik II and P. Rajčáni, *Iterative weighted finite transductions*, Acta Inform., 32 (1995), pp. 681–703.
[12] D. Derencourt, J. Karhumäki, M. Latteux, and A. Terlutte, *On computational power of weighted finite automata*, Fund. Inform., 25 (1996), pp. 285–293.
[13] S. Eilenberg, *Automata, Languages, and Machines*, Vol. A, Academic Press, San Diego, 1974.
[14] J. Goldstine, C. M. R. Kintala, and D. Wotschke, *On measuring nondeterminism in regular languages*, Inform. and Comput., 86 (1990), pp. 179–194.
[15] J. Goldstine, H. Leung, and D. Wotschke, *On the relation between ambiguity and nondetermism in finite automata*, Inform. and Comput., 100 (1992), pp. 261–270.
[16] L. Grove, *Algebra*, Academic Press, New York, 1983.
[17] J. Kari and P. Fränti, *Arithmetic coding of weighted finite automata*, RAIRO Inform. Théor. Appl., 28 (1994), pp. 343–360.
[18] C. M. R. Kintala and D. Wotschke, *Amounts of nondeterminism in finite automata*, Acta Inform., 13 (1980), pp. 199–204.
[19] W. Kuich and A. Salomaa, *Semirings, Automata, Languages*, EATCS Monographs on Theoretical Computer Science 5, Springer-Verlag, Berlin, 1986.
[20] M. Mohri, *Finite-state transducers in language and speech processing*, Comput. Linguist., 23 (1997), pp. 269–311.
[21] M. Mohri, *On the use of sequential transducers in natural language processing*, in Finite-State Language Processing, MIT Press, Cambridge, MA, 1997, pp. 355–382.
[22] M. Mohri, *Minimization algorithms for sequential transducers*, Theoret. Comput. Sci., 234 (2000), pp. 177–201.
[23] F. Pereira and M. Riley, *Speech recognition by composition of weighted finite automata*, in Finite-State Language Processing, MIT Press, Cambridege, MA, 1997, pp. 431–453.
[24] F. Pereira, M. Riley, and R. Sproat, *Weighted rational transductions and their application to human language processing*, in Proceedings of ARPA Human Language Technology Conference, San Francisco, CA, Morgan-Kaufmann, 1994, pp. 249–254.
[25] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Texts and Monographs in Computer Science, Springer-Verlag, New York, 1988.
[26] M. O. Rabin, *Probabilistic automata*, Inform. and Control (Shenyang), 6 (1963), pp. 230–245.

[27] M. D. Riley, A. Ljolje, D. Hindle, and F. C. N. Pereira, *The AT&T* 60,000 *word speech-to-text system*, in Proceedings of the 4th Euro. Conf. on Speech Communication and Technology, Vol. 1, Madrid, Spain, European Speech Communication Association (ESCA), 1995, pp. 207–210.

[28] E. Roche, *Analyse Syntaxique Transformationelle du Francais par Transducteurs et Lexique-Grammaire*, Ph.D. thesis, LITP-Université Paris 7, Paris, France, 1993.

[29] A. Salomaa and M. Soittola, *Automata-Theoretic Aspects of Formal Power Series*, Springer-Verlag, Berlin, 1978.

[30] M. Silberztein, *Dictionnaires électroniques et analise automatique de textes: le systéme INTEX*, Ph.D. thesis, Masson, Paris, France, 1993.

[31] A. Weber and R. Klemm, *Economy of description for single-valued transducers*, Inform. and Comput., 118 (1995), pp. 327–340.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1532–1551

ALGORITHMS FOR THE RELAXED ONLINE BIN-PACKING MODEL*

GIORGIO GAMBOSI†, ALBERTO POSTIGLIONE‡, AND MAURIZIO TALAMO§

Abstract. The typical online bin-packing problem requires the fitting of a sequence of rationals in $(0, 1]$ into a minimum number of bins of unit capacity, by packing the ith input element without any knowledge of the sizes or the number of input elements that follow. Moreover, unlike typical online problems, this one issue does not admit any data reorganization, i.e., no element can be moved from one bin to another.

In this paper, first of all, the "Relaxed" online bin-packing model will be formalized; this model allows a constant number of elements to move from one bin to another, as a consequence of the arrival of a new input element.

Then, in the context of this new model, two online algorithms will be described. The first presents linear time and space complexities with a 1.5 approximation ratio and moves, at most once, only "small" elements; the second, instead, is an $O(n \log n)$ time and linear space algorithm with a $1.33\ldots$ approximation ratio and moves each element a constant number of times. In the worst case, as a result of the arrival of a new input element, the first algorithm moves no more than three elements, while the second moves as many as seven elements. Please note that the number of movements performed is explicitly considered in the complexity analysis.

Both algorithms are below the theoretical $1.536\ldots$ lower bound, effective for the online bin-packing algorithms without the movement of elements. Moreover, our algorithms are "more online" than any other linear space online bin-packing algorithm because, unlike the algorithms already known, they allow the return of a (possibly relevant) fraction of bins before the work is carried out.

Key words. complexity, approximation algorithms, online algorithms, bin packing

AMS subject classifications. 68Q25, 68R05, 90C27

PII. S0097539799180408

1. Introduction.

1.1. The bin-packing problem. The bin-packing problem (see survey in [7] and in [8]) is a major issue in theoretical computer science: it consists of "packing" a set of nonoverlapping objects into a minimum number of well-defined areas. More formally [7], [8], given a positive integer C, it provides for the packing of a set of integer size elements $L = \{a_1, a_2, \ldots, a_n\}$, with $size(a_i) \in (0, C] \cap N_0$, into a minimum number of bins of equal capacity C.

This problem models the variable partitioning storage management in multiprogrammed computer systems and the assignment of commercials to mass media station breaks and truck packing. Bin-packing also models a variant of the scheduling problem in multiprocessors where the objective is to minimize the number of processors in

*Received by the editors January 1, 1999; accepted for publication (in revised form) January 28, 1999; published electronically November 17, 2000. This work was partially performed in the framework of Esprit BRA project 3075, "Algorithms and Complexity," and of Italian MURST 40% project, "Algoritmi e Strutture di Calcolo," and was partially supported by National Research Council project "Sistemi Informatici e Calcolo Parallelo."

http://www.siam.org/journals/sicomp/30-5/18040.html

†Dipartimento di Matematica, University "Tor Vergata," via della Ricerca Scientifica, 00133 Rome, Italy (gambosi@mat.uniroma2.it).

‡Dipartimento di Scienze della Comunicazione, University of Salerno, via Ponte don Melillo, 84084 Fisciano (SA), Italy (ap@unisa.it).

§Dipartimento di Informatica e Sistemistica, University "La Sapienza," via Salaria 113, 00198, Rome, Italy (talamo@dis.uniroma1.it).

which all tasks are to be completed within a given deadline. (When the common deadline is the capacity, processors are represented by bins and elements are represented by tasks whose size is given by the execution time.)

An interesting case is when $C = 1$ and $size(a_i) \in (0, 1] \bigcap Q$, which results in a combinatorial optimization problem that is NP-hard in the strong sense[1] since it contains 3-Partition as a special instance [15].

We are interested in searching for approximated fast (polynomial) online bin-packing algorithms that require the packing of the ith element without information on the sizes or the number of the following input elements and whose solution is far from the optimal for a small, fixed, multiplicative constant.

All the known online algorithms share the approach that no element can be moved from the bin it was first inserted in. Moreover, all the $\Theta(n)$-*space* algorithms are offline on output, i.e., no algorithm releases any of the used bins until the end of the input list has been reached, while all the $\Theta(1)$-*space* algorithms release all the bins except for a constant number of them.[2]

As pointed out in [23] a bin-packing algorithm is an algorithm made up of two parts: the first part reorders the list according to a preprocessing rule; the second part generates the packing. An online algorithm has no preprocessing step.

1.2. The relaxed model for the online bin-packing problem. In this paper, we will focus our attention on online algorithms, according to the classical definition [1], which states the following.

DEFINITION 1.1. *The online execution of a sequence of instructions σ requires that the instructions in σ be executed from left to right and that the ith instruction in σ be executed without looking at any of the instructions that follow.*

The above definition corresponds to the definition of online algorithms considered in task systems and server problems (see, for example, [4] and [28]). Please note that the above definition admits internal data reorganization, which is a frequent practice in most online algorithms.

According to these definitions, we introduce a new online bin-packing model, named "Relaxed," which allows a constant number of elements to be moved from one bin to another consequent to the arrival of new input elements. This new model calls for a careful definition of a cost function on the set of the possible item movements, in order to account explicitly for them in the overall algorithm complexity analysis.

Please note that a very limited number of applications of the bin-packing problem cannot be represented by our model. A typical example of such an application is the cutting stock problem, where the more abstract operation "assign an element of size s to a bin" is interpreted in terms of "cut a piece of length s from a stock element."

On the other side there are many real-life situations where the rearrangement of an allocated element is possible and this affects (by lowering) the cost of the resolution process, i.e., packing trucks and multiprocessor memory management strategies. The "Relaxed" model works very well in all situations in which the operation modeled by the assignment of an element to a bin can be "undone" by paying something. Such

[1] Note that, when the number of possible element sizes is a priori bounded, or (it is the same) in the integer formulation C is fixed, the problem can be exactly solved in polynomial time by exhaustive search, although the degree of the polynomial can be very high [7], [8] or it can be exactly solved asymptotically using special linear programming techniques [16], [17]; moreover, the decision problem "Is there a partition of L into disjoints sets $L_1, L_2, \ldots, L_k$ such that $\sum_{a \in L_i} size(a) \leq C$, for each L_i" is NP-complete and solvable in pseudopolynomial time for each fixed $k \geq 2$.

[2] Online algorithms which are not offline on output were considered in [10].

a situation arises, for example, when the input list is not known in advance (it could be infinite, too) and each element (that could arrive with a considerable delay from the previous one) needs to be processed online, while the guaranteed performance needs to be maintained at any time for the set of elements currently involved. In this situation a classical online algorithm (BEST-FIT, $HARMONIC_M$, etc.) or an offline algorithm could be applied in correspondence to each element arrival. In the first case no known algorithm uses, in the worst case, more than 63% of the bin space, while no algorithm can use, in the worst case, more than 65% of the bin space (since 1.53 is the theoretical lower bound). In the second case whenever an element arrives all other elements can be moved out from the bin they are contained in and be assigned to some other bin, according to the new computed solution. In our model only a (known) limited number of element movements is admitted, in correspondence to the arrival of a new element.

In general, this new model is particularly suitable when the elapsed time between two consecutive input elements is $\geq \delta K$, where δ is the maximum cost for each element movement, and K is the maximum number of element movements occurring in correspondence to each input element (so this model permits us to take advantage of the "dead times" between two successive input elements).

In [11] and [13] we informally introduced the "Relaxed" model and gave an $O(n \log n)$-*time* $O(n)$-*space* class of algorithms that, for each prefix of the input sequence, returns a 1.5 asymptotical approximation ratio. This value is below the $1.53\ldots$ theoretical lower bound [5], [26] well grounded for the restricted case and indicates that the relaxation of the classical online bin-packing problem conditions is convenient and theoretically interesting. Some experimental simulations allow us to guess that this class of algorithms has (on the average) very good behavior.

Our paper shows how this result is improved in two different ways by giving two linear space algorithms: the first presents a 1.5 approximation ratio with an $O(n)$ time complexity; the second presents a $1.33\ldots$ approximation ratio with an $O(n \log n)$ time complexity (they fill each bin in the worst case at least for 66% and 75%). Moreover, at the arrival of each input element, in the worst case the first algorithm moves no more than three elements while the second may move up to seven elements.

Last, please note that these algorithms are "more online" than all the other linear space online bin-packing ones because, unlike the known algorithms, they allow the return of a (possibly relevant) fraction of the bins before the work is carried out.

Section 2 gives definitions and a brief summary of the previous results on the online bin-packing problem; section 3 shows the "Relaxed" model, with regard to element movement, small element grouping operations, and definition of movement evaluation function; section 4 introduces the linear algorithm A_1 while section 5 gives an analysis of its performance; the $O(n \log n)$ algorithm A_2 is introduced in section 6 and its performance is analyzed in section 7; section 8 examines some conclusions and open problems.

2. Definitions and previous results.

2.1. Problem definition.

The classical one-dimensional bin-packing problem can be stated [15] as follows.

DEFINITION 2.1. *Given a finite set $L = \{a_1, a_2, \ldots, a_n\}$ of "elements" and a rational "size," $size(a) \in (0, 1]$, for each element $a \in L$, find a partition of L into disjoint subsets $L_1, L_2, \ldots, L_k$ such that the sum of the sizes of the elements in each L_i is no greater than 1 and such that k is as small as possible.*

Since the bin-packing problem contains 3-partition as a special case, it is an NP-hard problem in the strong sense [15], [21]. It is therefore very unlikely that there are fast (polynomial) algorithms for finding the best solution, unless P = NP, even if the magnitude of the numbers involved is bounded by a polynomial in n.

Given an (approximate) algorithm A for the bin-packing problem and a set L of elements, let $A(L)$ be the number of bins used by A to pack L. Therefore

$$OPT(L) \geq \sum_{a_i \in L} size(a_i) \tag{2.1}$$

is a lower bound for the number of bins necessary to pack L.

Now we are able to give some algorithm performance definitions [7], [8].

DEFINITION 2.2. *The performance of A with respect to OPT on the list L is*

$$R_A(L) \equiv \frac{A(L)}{OPT(L)}. \tag{2.2}$$

DEFINITION 2.3. *The absolute performance ratio R_A of the algorithm A is*

$$R_A \equiv \inf\{r \geq 1 | R_A(L) \leq r, \ \forall \ list \ L\}. \tag{2.3}$$

DEFINITION 2.4. *The asymptotic performance ratio R_A^∞ of the algorithm A is*

$$R_A^\infty \equiv \inf\{r \geq 1 | \ for \ some \ N > 0, R_A(L) \leq r, \ \forall \ L \ with \ OPT(L) \geq N\} \tag{2.4}$$

Note that $R_A \geq R_A^\infty$.

2.2. Previous results on the online version. The classical problem presents a variety of cases [7], [8]. In the online version [10] the following definition exists.[3]

DEFINITION 2.5. *"Items are assigned to bins in order $(a_1, \ a_2, \ \ldots)$, with item a_i assigned only according to the size of the previous items and the bins to which they were assigned, without considering the size or number of items that follow."*

The simplest bin-packing algorithm is Next-Fit [7], [8] which is $O(n)$-*time* and $O(1)$-*space*, but whose asymptotical performance, both in the worst and in the average cases, is very poor, respectively, 2.0 and 1.33 [7], [8], [9], [12].

The bin-packing algorithms most extensively used are BEST-FIT and FIRST-FIT [23]. Both algorithms are $O(n \log n)$-*time* and $O(n)$-*space* and present an acceptable asymptotical worst-case performance (i.e., 1.7, [23]) but an optimal asymptotically average performance [2], [3], [12], [32].[4]

Until now, the best online algorithms for the bin-packing problem, without moving elements from the bins they have been assigned to, belonged to the HARMONIC class, first introduced in [25], where an approximated algorithm called $HARMONIC_M$ was introduced; this algorithm is the optimal among all the $O(1)$-*space* algorithms. Such an algorithm has an $O(n)$-*time* complexity and a ratio $R_H^\infty(M) \leq 1.692$ for all $M \geq 12$. Lee and Lee [25], moreover, proved that $R_A^\infty \geq 1.6910$ for all constant space algorithms and that $\lim_{M \to \infty} R_H^\infty(M) = 1.6910$.

[3] More appropriately in such a case, we deal with a sequence of elements to be packed and not with a set.

[4] Note that in [10] it is proved that BEST-FIT obtains its worst-case performance even if a constant ($k \geq 2$) number of bins is maintained online; this reduces the computation time to $O(n \log k)$, that is, $O(n)$ since k is a small constant.

The same authors gave a more complex $O(n)$ algorithm, REFINED HARMONIC, which uses $O(n)$ space and presents a ratio $R^{\infty}_{RH} = \frac{373}{228} = 1.636\ldots$.

Later, the MODIFIED HARMONIC algorithm was introduced in [31], which is $O(n)$ both in time and in space complexity with a ratio $R^{\infty}_{MH} = 1.61(561)^{*}$. The authors also showed how an online algorithm with $R^{\infty}_{A} < 1.59$ can be obtained.

3. The relaxed model.

3.1. Motivations and previous results. All the algorithms mentioned in section 2.2 introduce the additional limit that the solution for $\langle a_1\ a_2\ \ldots\ a_i\rangle$ must derive from the one for $\langle a_1\ a_2\ \ldots\ a_{i-1}\rangle$ without performing any reorganization of the elements in the bins; that is, none of the elements among $a_1, a_2, \ldots, a_{i-1}$ can be moved from the bin it belongs to. In other words, all of these algorithms only search for a suitable bin to which to assign element a_i in order to obtain a good asymptotic approximation of the optimal solution. In this context some interesting lower bounds have been proved, as already pointed out.

A question arising from the above considerations is the following: "What happens if we interpret the online property of bin-packing in a less restricted way, just like the large majority of online models? Is it possible to obtain more efficient performances if a bin-packing algorithm can move the elements a certain number of times from one bin to another?"

In [11] and [13] an affirmative answer to this question is given by presenting a class of online algorithms, $HARMONIC_{REL}(M)$,[5] with time complexity $O(n \log n)$, space complexity $O(n)$, and asymptotic ratio $R^{\infty}_{HREL}(M) \leq 1.5$. In the worst case, the approximation ratio is independent of M, for $M \geq 3$, and the number of movements is limited in an amortized way by a (small) constant (2, for $M = 3$).

3.2. Grouping elements. In this paper we introduce a new operation: the "grouping of elements," i.e., *we assume that a certain number of very small items in the same bin can be collected together and considered as a single unit.* More formally,

Given a constant $0 < c < 1$, we assume that any set of elements smaller than c in the same bin can be collected together in a single group of overall size $\leq c$. This group will be considered as a single unit from now on.

Obviously, the grouping of elements does not modify the approximation ratio of OPT, since OPT is measured as the sum of the elements in the input list. We also assume that there is no kind of movement inside any group.

In the bin-packing problem the grouping operation is possible and convenient. For example, in the truck-packing problem it is useful to fit a collection of very small elements in the same box and then move them as a whole by moving the box. In multiprocessor storage management strategies, the grouping simply consists in collecting a subset of pages.

3.3. Moving elements. In the relaxed model, the critical operation regards moving (part of) the contents from one bin to another.

In the following, i, j are two bins and σ is a subsequence of (not necessarily contiguous) input element(s), all contained in the same bin.

The fundamental operation could therefore be stated as

$$\text{MOVE}(i, j, \sigma), \qquad i \neq j, \tag{3.1}$$

which means that σ is moved from bin i to bin j.

[5] Where REL stands for "Relocation."

This approach is quite natural for all the applications in which we may assume that several small elements can be "carried" from one bin to another in a single step.

3.4. MOVE operation cost. An online algorithm processes the input data one at a time, possibly modifying its internal data structures. Thus the evaluation of the performance of an algorithm is more realistic if it takes into account the number of movements of the elements in its data structures.

In a bin-packing algorithm when the elements move, several kinds of cost functions for the MOVE(i, j, σ) operation could be defined.

DEFINITION 3.1. *The cost of the* MOVE *operation is equal to the total size of all elements moved* ($\sum_{x\in\sigma} size(x)$).

DEFINITION 3.2. *The cost of the* MOVE *operation is equal to the number of elements moved* ($|\sigma|$).

In this paper we will consider a third way to define such a function. In our approach *we assume that each group can be moved at unitary cost.* That is, while moving a "large" element always has a cost equal to 1, we assume that "small" elements can be *grouped* together and moved as a whole, at unit cost. Therefore we have the following.

DEFINITION 3.3. *The cost associated to the* MOVE(i, j, σ) *operation is equal to the number of elements and groups contained in* σ.

If the element moving cost would only be a function of the size of the elements, any reasonable algorithm would tend to move a lot of small elements because the performance is better and there is no cost difference in moving a lot of small elements instead of a few big elements. If the element moving cost would only be a function of the number of the elements moved, there will be no cost difference between an algorithm that moves light elements and another that moves the same number of heavy elements. Therefore, the third cost function is the most likely. It should be clear that for any c, σ this function has a value which is in between the values assumed by the first two cost functions above defined.

3.5. Formal definition of grouping. Let us consider the following nonuniform partition of $(0, 1]$ in $M + 1$ subintervals:

$$(0,1] = \bigcup_{k=0}^{M} I_k,$$

$$I_0 = \left(\frac{3}{4}, 1\right]; \; I_1 = \left(\frac{2}{3}, \frac{3}{4}\right]; \; I_2 = \left(\frac{1}{2}, \frac{2}{3}\right]; \; \ldots; \; I_{M-1} = \left(\frac{1}{M-1}, \frac{1}{M}\right]; I_M = \left(0, \frac{1}{M}\right].$$

Let $c = \frac{1}{M}$ be the *border item size.* The grouping operation consists of collecting a set of elements smaller than c in a single group g (that will be a sort of "superitem"), so that, in each bin B (let $size(g) = \sum_{a\in g} size(a)$),

- for all $g \in B$, $size(g) \le c$;
- there are no pairs of groups $g \in B, h \in B$, such that $size(g) + size(h) \le c$;
- each group $g \in B$, except at most one, has $size(g) \ge \frac{1}{2}c$.

3.6. Grouping primitives.

3.6.1. Create group. This primitive regards the arrival of a new element in $(0, \frac{1}{M}]$ that cannot be merged in any group of the target bin. The operation consists in creating an empty group and in inserting this new element in it. At all times, there will be no more than one I_M-bin open.

3.6.2. Append. This primitive regards the arrival of a new element in $(0, \frac{1}{M}]$ that has to be merged into an existing group.

3.6.3. Primitive performances. Since we are not interested in any kind of arrangement of the elements within the group, a suitable representation (i.e., linked lists) allows all of these operations to be executed in constant time and space. This leads to the "packing" of such elements together, so that they can/must be moved as a whole, in one single step.

3.7. Evaluation function. Below we will show that each bin will contain a constant number of groups. Since the algorithm performance is measured as a function of the space wasted with respect to the sum of the sizes of the elements in the input list, the grouping of the elements does not affect the performance in any manner.

We will not detail the operations involved in inserting and deleting elements to and from bins nor the ones involved in the maintenance of the support data structures, mainly in empty conditions, because they can be easily performed in constant time.

In the following, let

- m be the maximum number of MOVE operations performed upon the arrival of a new element[6];
- r be the asymptotic performance of the algorithm.

Thus, we can assign to an approximation algorithm a pair of numbers, such as

$$A(m, r).$$

For example, the well-known BEST-FIT algorithm is A(1,1.7) since its performance ratio is 1.7. In general, we can say that a classical online bin-packing algorithm is A(1,r) ($r \geq 1.53$) since it does not move the elements already fitted in the bins and 1.53 is the lower bound for this kind of algorithm. Please note that the exact algorithm is A(m,1), for some $m \geq 0$, while our first algorithm, A_1, is A(3,1.5), and A_2 is A(7,1.33).

4. The linear algorithm A_1. A_1 is based on a nonuniform partition of interval $(0, 1]$ into four subintervals (levels):

$$(0, 1] = \bigcup_{k=0}^{3} I_k,$$

$$I_0 = \left(\frac{2}{3}, 1\right]; \quad I_1 = \left(\frac{1}{2}, \frac{2}{3}\right]; \quad I_2 = \left(\frac{1}{3}, \frac{1}{2}\right]; \quad \text{and } I_3 = \left(0, \frac{1}{3}\right].$$

In order to describe it, let us introduce the following points:

- $S = \langle a_1\ a_2\ \ldots\ a_n \ldots \rangle$ is the "input list."
- Let $a_i \in I_k$ $(0 \leq k \leq 2)$ be an element of S. Then a_i is called "I_k-element."
- Let $a_i \in I_3$ be an element of S. Then a_i is called "I_3-group."
- Let B be a bin. Then B is an "I_k-bin (I_3-bin)" $(0 \leq k \leq 2)$ if the first element that was initially assigned to it were an I_k-element (I_3-group).
  By subinterval definition, each I_k-bin $(1 \leq k \leq 2)$ contains no more than k I_k-elements and each I_0-bin contains no more than one I_0-element.

[6] the first insertion of a new element corresponds to a MOVE from outside into a bin

If an I_k-bin ($1 \le k \le 2$) exactly contains k I_k-elements it is *"filled"*; otherwise it is *"unfilled"*. An I_0-bin with an I_0-element is *"filled"* and an I_3-bin is filled only when its gap is $< \frac{1}{3}$.[7]

- For each k ($0 \le k \le 3$) let A_k be the name of the only unfilled I_k-bin.
- "*gap(B)*" is the space available in an I_k-bin, B ($0 \le k \le 2$), to insert I_l-elements (I_3-groups) ($k < l \le 2$). If B is filled, then $gap(B) = 1 - \sum_{a \in B} size(a)$; otherwise we conventionally assume that $gap(B) = 0$.

Algorithm A_1 is reported below (where l denotes the level of the next input element, x). Please note that if $x \in I_1$ is a "small" I_1-element (i.e., $size(x) < \frac{2}{3}$), A_1 tries to insert some I_3-groups in its gap; if this is not possible A_1 will mark this bin for a future I_3-group insertion. Please note that if $x \in I_3$, then A_1 first tries to insert it in the gap of some marked I_1-bin with enough room.

The algorithm uses two stacks of bins, L_1 and L_3, respectively associated with levels 1 and 3. L_1 maintains all the bins whose gap is still "fat" (i.e., $\ge \frac{1}{3}$), while L_3 maintains all the I_3-bins. If there is an unfilled I_3-bin, then it is the first bin in L_3. Please note that in every moment no more than one between L_1 and L_3 can be "not empty."

We do not explicitly consider the management of unfilled bins. For example, we assume that an unfilled bin is automatically generated at the arrival of an element which can be assigned to no other bin available at that time.

Algorithm A_1

For each input element x**:**
if $x \in I_0$ **then** "Insert x in A_0".
if $x \in I_1$ **then**
- "Insert x in A_1";
- **while** ($gap(A_1) \ge \frac{1}{3}$) **AND** ("There still exists an I_3-group", g) **do** "Move g to A_1."
- **if** "There is no more I_3-groups" **AND** ($gap(A_1) \ge \frac{1}{3}$) **then** "Push A_1 in L_1."

if $x \in I_2$ **then**
- "Insert x in A_2";

if $x \in I_3$ **then**
- **if** "There exists an I_1-bin, B, in L_1"
  then "Insert x in B, removing B from L_1 if its gap becomes $< \frac{1}{3}$."
  else "Insert x in A_3"

5. Performance analysis of A_1. In order to analyze the performance of A_1 we must first consider the total number of element movements within the bins at the arrival of a new element. Next, we will consider its asymptotic performance ratio.

5.1. Time, space, and movements.

LEMMA 5.1. *Each filled I_1-bin contains no more than two groups in its gap.*

Proof. Let us assume there are more than two groups in a filled I_1-bin. Let x, y, z be three of them. Since an I_1-bin B has $gap(B) < \frac{1}{2}$, it follows that $size(x) + size(y) + size(z) < \frac{1}{2}$. By definition, we know that in every bin all the groups except for one (at most) are $\ge \frac{1}{6}$ in size. Without loss of generality (w.l.o.g.) let us assume that $size(x) \ge \frac{1}{6}$. Therefore

$$\frac{1}{2} > size(x) + size(y) + size(z) \ge \frac{1}{6} + size(y) + size(z) \Rightarrow size(y) + size(z) < \frac{1}{3},$$

which is a contradiction, since every pair of groups in each bin has a total size $> \frac{1}{3}$. □

[7] Note that we distinguish among "I_k-filled bins" (that is, bins no more able to receive all possible items of their class, but still active) and "I_k-full bins" (that is, bins whose gap is empty or that are never used afterwards).

Please note that this bound is tight. It is easy to show that two groups can be fitted together in the gap of this bin. An example is the following: $(\frac{1}{2}+\epsilon), (\frac{1}{6}-2\epsilon), (\frac{1}{6}+3\epsilon)$.

COROLLARY 5.2. *No more than three movements will be performed at each insertion.*

Proof. The above lemma proves how the movements only occur at the arrival of I_1-elements with $size < 2/3$. However, the algorithm performs no more than three movements since each I_1-bin has a $gap < \frac{1}{2}$ and each pair of groups has $size > \frac{1}{3}$. This implies that, in the worst case, it is sufficient to move one group from A_3 and two groups from another bin in L_3. □

THEOREM 5.3. *Algorithm A_1 has space complexity $O(n)$ and time complexity $\Theta(n)$.*

Proof. The space complexity easily derives from the observation that each element is represented no more than once in L_1 and L_3.

As far as time complexity, according to the above lemma we know that the maximum number of element insertions in a bin is bounded by $3n$. Each insertion can be performed in $O(1)$ time. Moreover, the movement of an existing element is performed in $O(1)$ time since this movement uses the first element in the first bin on the list, accessed in constant time. Therefore, the time complexity is easily derived. □

5.2. Performance ratio. In order to derive the approximation ratio for A_1, the following lemmas are needed.

LEMMA 5.4. *If, after all elements have been considered, L_3 is not empty, then $R^{\infty}_{A_1} < \frac{3}{2}$.*

Proof. The gaps of I_0-bin and I_k-bin ($k \geq 2$) are $< \frac{1}{3}$, by definition.

As far as I_1-bins please note that there is at least one element in L_3 whose size is $\leq \frac{1}{3}$, which has not been moved to the gap of any I_1-bin; this implies that all the gaps of I_1-bins have size $\leq \frac{1}{3}$.

In conclusion, the maximum gap in each bin is $< \frac{1}{3}$ and, consequently,

$$R^{\infty}_{A_1} < \frac{3}{2}. \qquad \square$$

LEMMA 5.5. *If, after all the elements have been considered, L_3 is empty and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there was no pair of a_i, a_j, so that $a_i \in I_1$ and $a_j \in I_2$, then $R^{\infty}_{A_1} = 1$.*

Proof. In this case A_1 uses $N_0 + N_1$ or $N_0 + \frac{N_2}{2}$ bins, where N_j is the number of I_j-elements in the input set, since no I_1-elements or I_2-elements could be inserted in any bin which already has an I_0-element and 2 I_2-elements are inserted in the same I_2-bin. Since OPT cannot use fewer bins, then

$$R^{\infty}_{A_1} = 1. \qquad \square$$

LEMMA 5.6. *If, after all the elements have been considered, L_3 is empty and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there was at least one pair a_i, a_j, so that $a_i \in I_1$ and $a_j \in I_2$, then $R^{\infty}_{A_1} \leq \frac{3}{2}$.*

Proof. Let B_i be the number of bins of level i used by OPT. We can derive the maximum number of bins used by A_1 as a function of the B_i's.

By definition,

$$OPT = B_0 + B_1 + B_2.$$

Since in each I_1-bin OPT could have inserted no more than one I_2-element, the extra bins for A_1 are no more than $\frac{B_1}{2}$. Thus,

$$A_1 \leq B_0 + B_1 + B_2 + \frac{B_1}{2} \leq \frac{3}{2}(B_0 + B_1 + B_2) = \frac{3}{2}\text{OPT}. \qquad \square$$

THEOREM 5.7. *Algorithm A_1 has a ratio $R^{\infty}_{A_1} \leq \frac{3}{2}$.*

Proof. The proof derives directly from the previous three lemmas. $\square$

THEOREM 5.8. *Algorithm A_1 is $A(3, 1.5)$.*

Proof. The proof derives directly from Corollary 5.2 and Theorem 5.7. $\square$

6. The $O(n \log n)$ Algorithm A_2.

6.1. Main features. A_2 is based on a nonuniform partition of the interval $(0, 1]$ into six subintervals (levels):

$$(0, 1] = \bigcup_{k=0}^{5} I_k, \tag{6.1}$$

$$I_0 = \left(\frac{3}{4}, 1\right]; I_1 = \left(\frac{2}{3}, \frac{3}{4}\right]; I_2 = \left(\frac{1}{2}, \frac{2}{3}\right]; I_3 = \left(\frac{1}{3}, \frac{1}{2}\right]; I_4 = \left(\frac{1}{4}, \frac{1}{3}\right]; I_5 = \left(0, \frac{1}{4}\right]. \tag{6.2}$$

The definitions of I_k-element, I_k-bin, A_k-bin, "filled" bin, and gap are similar to the ones given for Algorithm A_1, while the definition of I_5-group is similar to the definition of I_3-group given for Algorithm A_1. Please note that an I_5-group B is "filled" if $gap(B) < \frac{1}{4}$. Thus, A_2 considers I_5-elements as "little" elements which can be collected in groups g_i and moved together. As pointed out in section 3.6, all the grouping primitives are constant in time and space.

6.2. Packing strategy. The algorithm operates as reported in Algorithm A_2, where l denotes the level of the next input element, x. When the input element is a "big" one (i.e., $size(x) > \frac{1}{2}$), A_2 inserts it in a new bin and tries to "fill" its gap with smaller elements from some other bin(s). If the input element is a "small" one (i.e., $size(x) \leq \frac{1}{2}$), the algorithm first tries to insert it in the gap of a filled bin which already exists; only if there is no room for x in any other existing bin, the algorithm inserts it in a new bin. During its execution, A_2 refers to an unfilled bin for each level.[8] The guidelines of the algorithm are the following:

- A_2 encourages the pairing of elements x, y, where $(x \in I_1, y \in I_4)$ or $(x \in I_2, y \in I_3)$.
- A_2 tries to fill the gap of I_1-, I_2-, I_3-filled bins with smaller elements (I_4-elements or I_5-groups) since there are no more I_5-groups or no bin B has $gap(B) > \frac{1}{4}$.

Both of these guidelines may imply the move of a few elements from one bin to another. Bins may be emptied as an effect of element moving: in this case, the emptied bins are considered as automatically disregarded. Finally, please note that the algorithm may return some of the used bins as output before the end of the input list.

[8] As for Algorithm A_1, we do not explicitly consider the management of unfilled bins.

6.3. Data structures. The algorithm requires the use of

- one stack S, containing all the I_5-bins; the unfilled bin is the top one;
- three dictionaries, D_2, D_3, and D_4, maintaining all the I_k-elements contained exclusively in I_k-bins (not necessarily filled), for $k = 2, 3, 4$.
- three dictionaries (tournaments), G_1, G_2, and G_3, maintaining the size of the gap of all the I_1-, I_2-, I_3-filled bins.

We will not give details of the operations involved in inserting and deleting elements to and from the bins and the ones involved in the lists and in I_5-groups maintenance, since they can be easily performed in constant time. Moreover, we will not give details of the operations regarding the tree data structures since they are well known and they may be executed in $O(\log n)$ time. G_1, G_2, and G_3 can be implemented as binary trees of depth $\lceil \log_2 n \rceil$ with n leaves corresponding to the n bins in sequence from left to right. Each internal node is labeled with the largest label among the labels of its sons and each leaf is labeled with the current gap of the bin it represents. Please note that bins containing pairs $x \in I_1$ and $y \in I_4$ or $x \in I_2$ and $y \in I_3$ are immediately returned as output by the algorithm, hence they are not represented in these directories. The tree representation chosen is similar to the one Johnson used to implement the FIRST-FIT algorithm [23]. Last, we will not refer to the possible output of any bin (e.g., either all the I_0-bins or all the bins containing an I_2-element and an I_3-element could be sent to the output).

6.4. Algorithm primitives.

6.4.1. Insert(b,A). This primitive inserts object b, which could be either an item or a group, into bin A and updates, if necessary, one or two of the dictionaries.

Insert (b,A)

- "**Insert** b in A."
- **if** $b \in I_5$ **then** "**Append** b to an existing group or **Create a new group** with only b."
- "**Update**, if necessary, D_2, D_3 or D_4 and G_1, G_2 or G_3"

The updating operation is an $O(\log n)$-time operation and will be carried out only if

- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 2, 3, 4)$ ("enter b in D_k");
- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 1, 2, 3)$ AND A is filled as a consequence of this new element insertion ("enter A in G_k");
- $(A \in I_k)$ and $(b \notin I_k)$ $(k = 1, 2, 3)$ ("Update the size of the gap of A in G_k"). This case occurs only if A is filled and b has to be inserted in its gap.

In conclusion, the Insert operation is $O(\log n)$ worst case time.

6.4.2. Extract(b,A). This primitive extracts object b, which could be either an item or a group, from bin A and updates, if necessary, one or two of the dictionaries.

Extract (b,A)

- "Pop b from A"
- "Update, if necessary, D_2, D_3 or D_4 and G_1, G_2 or G_3"

The updating operation is an $O(\log n)$-time operation and will be carried out only if

- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 2, 3, 4)$ ("extract b from D_k");
- both $(b \in I_k)$ and $(A \in I_k)$ $(k = 1, 2, 3)$, AND A becomes "unfilled" as a consequence of this element extraction ("extract A from G_k");
- $(A \in I_k)$ and $(b \notin I_k)$ $(k = 1, 2, 3)$ ("Update the size of the gap of A in G_k). This case occurs only if A is filled and b has to be extracted from its gap.

In conclusion, the Extract operation is $O(\log n)$ worst case time.

6.4.3. Move(b,B,A). This primitive moves object b from bin B to bin A. It is a composition of Extract(b,B) and Insert (b,A), so it is $O(\log n)$ worst case time.

6.4.4. Fill(C). This primitive fills the gap of the "filled" bin $C \in I_k (1 \le k \le 3)$ with smaller elements since there is no room in C (i.e., $gap(C) < \frac{1}{4}$) or there are no more of these little elements. This operation easily is $O(\log n)$ worst case time.

Fill (C)
if "there exists an I_4-bin B containing an element $b \in D_4$ such that $size(b) \le gap(C)$" **then** Move(b,B,C)
if $B \neq A_4$ **then** Move(x,A_4,B), "for whatever element $x \in A_4$"
else while "there exists $g \in A_5$ such that $size(g) \le gap(C)$ **AND** $gap(C) \ge \frac{1}{4}$" **do** Move(g,A_5, C)

6.4.5. MoveTheGap(C). This primitive moves all the objects (I_5-groups and eventually the only I_4-element) from the gap of bin C and distributes them among all the other bins. This operation easily is $O(\log n)$ worst case time.

MoveTheGap(C)
if "C contains an I_4-element, b" **then** Move(b,C,A)(where A is, in the sequence of checks, an I_1-bin, an I_2-bin, an I_3-bin or the A_4-bin)
for "every group $g \in C$" **do if** "there exists some I_1, I_2, I_3-bin C' with $size(g) \le gap(C')$" **then** Move (g,C,C') **else** Move(g,C,A_5)

6.5. The algorithm.

Algorithm A_2
For each element x:
if $x \in I_0$ **then**
- Insert(x,A), where A is a new I_0-bin;

if $x \in I_1$ **then**
- Insert(x,A), where A is a new I_1-bin;
- Fill(A);

if $x \in I_2$ **then**
- Insert(x,A), where A is a new I_2-bin;
- **If** there is a $b \in I_3$ in some I_3-bin B, so that $size(b) + size(x) \le 1$,
  then * Move (b,B,A); MoveTheGap (B);
  * **if** $B \neq A_3$ **then** "Move(b,A_3,B) for whatever element $b \in A_3$; Fill(B)"
  else * **Fill**(A);

if $x \in I_3$ **then**
- **If** there is a $b \in I_2$ in some I_2-bin B so that $size(b) + size(x) \le 1$
  then * **If** $size(x) > gap(B)$ **then** MoveTheGap (B);
  * Insert(x,B);
  else * Insert(x,A_3); **if** A_3 becomes filled, **then** Fill(A_3);

if $x \in I_4$ **then**
- **If** there is an I_1-bin B so that $size(x) \le gap(B)$, **then** Insert(x,B);
- **else** Insert (b,A) (where A is, in the sequence of checks, an I_2-bin, an I_3-bin or, at last, the A_4-bin).

if $x \in I_5$ **then**
- Create a group g containing only x;
- **If** there is an I_1, I_2, I_3-bin B such that $size(g) \le gap(B)$ **then** Insert(g,B) **else** Insert(g,A_5)

7. Performance analysis of A_2. In order to analyze the performance of the algorithm we must first consider the number of element movements caused by the arrival of a new input element and then its asymptotic performance ratio.

7.1. Time, space, and movements.

LEMMA 7.1. *Each filled I_1-bin may contain, in its gap, no more than two I_5-groups.*

Proof. If we assume there are more than two I_5-groups in a filled I_1-bin and let x, y, z be three of them, by definition it follows that

$$x \le \frac{1}{4}, \quad y \le \frac{1}{4}, \quad z \le \frac{1}{4}; \qquad x+y > \frac{1}{4}, \quad x+z > \frac{1}{4}, \quad y+z > \frac{1}{4}.$$

Moreover, an I_1-bin B, has $gap(B) < \frac{1}{3}$, thus $x+y+z < \frac{1}{3}$. Hence

$$\frac{1}{3} > x+y+z > \frac{1}{4} + z \Rightarrow z < \frac{1}{12}.$$

Therefore

$$x > \frac{1}{4} - z > \frac{1}{6}; \qquad y > \frac{1}{4} - z > \frac{1}{6}; \qquad x+y+z > \frac{1}{3},$$

which is a contradiction. □

LEMMA 7.2. *Each filled I_2-bin may contain no more than three I_5-groups or one I_4-element plus one I_5-group in its gap.*

Proof. Let us assume there are more than three I_5-groups in a filled I_2-bin. Let x, y, z, w be four of them. By definition of *group* we have that

$$x \le \frac{1}{4}, \quad y \le \frac{1}{4}, \quad z \le \frac{1}{4}, \quad w \le \frac{1}{4}, \quad x+y > \frac{1}{4}, \quad x+z > \frac{1}{4},$$
$$x+w > \frac{1}{4}, \quad y+z > \frac{1}{4}, \quad y+w > \frac{1}{4}, \quad z+w > \frac{1}{4}.$$

By construction, an I_2-bin, B, has $gap(B) < \frac{1}{2}$, so $x+y+z+w < \frac{1}{2}$.

Therefore

$$\frac{1}{2} > x+y+z+w > \frac{1}{4} + z + w \Rightarrow z+w < \frac{1}{4},$$

which is a contradiction.

Similarly, let us assume that the group contains one I_4-element together with two I_5-groups. Let x be the I_4-element and let y, z be the I_5-groups. By definition, we have that $x \ge \frac{1}{4}$, $y+z \ge \frac{1}{4}$.

By construction, an I_2-bin, B, has $gap(B) < \frac{1}{2}$, so $x+y+z < \frac{1}{2}$.

Therefore

$$\frac{1}{2} > x+y+z > \frac{1}{4} + y + z \Rightarrow y+z < \frac{1}{4},$$

which is a contradiction. □

LEMMA 7.3. *Each filled I_3-bin may contain no more than two I_5-groups or one I_4-element plus one I_5-group in its gap.*

Proof. The first bound is proved as in Lemma 7.1.

To prove the second bound let us assume that the group contains one I_4-element and one I_5-group. Let x be the I_4-element and let y, z be the I_5-groups. By definition, we have that $x \ge \frac{1}{4}$, $y+z \ge \frac{1}{4}$.

By construction, an I_3-bin, B, has $gap(B) < \frac{1}{3}$, so $x+y+z < \frac{1}{3}$.

Therefore

$$\frac{1}{3} > x+y+z > \frac{1}{4} + y + z \Rightarrow y+z < \frac{1}{12},$$

which is a contradiction. □

LEMMA 7.4. *Each I_5-bin may contain no more than seven I_5-groups.*

Proof. It follows from the definition of I_5-group. □

The bound of the previous lemma is tight. The sequence matching this bound is $\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8} + \epsilon$, when no one of the these elements has to be fitted in any of the I_k-bins ($0 \leq k \leq 3$) and there is no (unfilled) I_5-bin at the arrival of the first element of the subsequence.

THEOREM 7.5. *In every moment, Algorithm A_2 maintains at most a constant number of I_5-groups and I_4-elements, and no other different level element, in the gap of each bin.*

Proof. It derives from Lemmas 7.1, 7.2, 7.3, and 7.4 and by the fact that I_0-bins contain only I_0-elements and I_4-bins contain only I_4-elements.

The remaining part of the theorem is proved by observing that no I_3-element can be fitted in the gap of any I_0-, I_1-, I_4-, I_5-bin while those fitted in the gap of I_2-bins are immediately returned as output. Note that, for the same reason, no I_4-element is maintained in the gap of any I_1-bin. □

LEMMA 7.6. *When procedure* **Fill** *is called to "fill" an I_k-bin ($1 \leq k \leq 3$) it performs no more than two I_4-element movements or three I_5-group movements.*

Proof. The first bound is simply inferred from the algorithm structure.

The second one is easily obtained by considering that, in every moment, there is only one "unfilled" I_5-bin, named A_5. If $SIZE(A_5) > \frac{1}{4}$ procedure **Fill** moves no more than two groups from it since, whatever is the I_5-bin, any pair of groups has a total size $\geq \frac{1}{4}$, each filled bin has a gap $< \frac{1}{2}$, and the loop ends when $gap(A) < \frac{1}{4}$. If $SIZE(A_5) \leq \frac{1}{4}$ then it contains only one group. Let ϵ be its size. If $SIZE(A) + \epsilon < \frac{3}{4}$ and there is another I_5-bin (that is necessarily filled) the situation is the same as the above. Thus, the total number of group movements is three. □

The bound of the previous lemma is tight. In fact let us suppose that the input sequence starts with the following elements: $\frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8}, \frac{1}{8} + \epsilon, \epsilon, \frac{1}{2} + \epsilon$; in this case the first seven elements will be fitted in one I_5-bin, the eighth will be fitted in another I_5-bin and the last will be fitted in an I_2-bin. The **Fill** procedure will move ϵ, then $\frac{1}{8} + \epsilon$, and at last $\frac{1}{8}$ to the gap of the I_2-bin.

THEOREM 7.7. *In correspondence to each insertion, a constant number of element or group movements is performed.*

Proof. Let x be the current input element to be fitted in any bin. Then the algorithm makes

- $x \in I_0$
  - 0 movements
- $x \in I_1$
  - no more than three movements (Lemma 7.6) as a consequence of
    - (two I_4-element movements) OR (three I_5-group movements)
- $x \in I_2$
  - no more than seven movements[9] as a consequence of
    - one I_3-element movement
    - two movements (Lemma 7.3) as a consequence of
      - (one I_4-element and one I_5-group movement) OR (two I_5-group movements)
    - one I_3-element movement
    - no more than three movements (Lemma 7.6) as a consequence of
      - (two I_4-element movements) OR (three I_5-group movements)

    OR, alternatively,
  - no more than three movements (Lemma 7.6) as a consequence of

[9] If there exists a $b \in I_3$ such that $size(x) + size(b) \leq 1$.

(two I_4-element movements) OR (3 I_5-group movements)

$x \in I_3$ • no more than three movements[10] (Lemma 7.2) as a consequence of
(one I_4-element and one I_5-group movements) OR (three I_5-group movements)
OR, alternatively,
• no more than three movements (Lemma 7.6) as a consequence of
(two I_4-element movements) OR (three I_5-group movements)

$x \in I_4$ • 0 movements.

$x \in I_5$ • 0 movements.

Therefore, in each case, the total number of element (or group) movements is constant. □

COROLLARY 7.8. *No more than seven element movements occur at the arrival of a new input element when A_2 is applied to a list of n elements.*

Proof. The proof is easily derived from Theorem 7.7. □

THEOREM 7.9. *Algorithm A_2 has space complexity $O(n)$.*

Proof. The theorem is easily proved when observing that each element is represented no more than once in the list S or in the data structures involved and that the maximum number of gaps is n. □

THEOREM 7.10. *Algorithm A_2 has time complexity $O(n \log n)$.*

Proof. According to Theorem 7.7, the time complexity is bounded by n times the cost of an element insertion or movement. Each insertion and each movement of an element already in the structure is performed at most in $O(\log n)$ time, when the tree data structures are involved. In fact the directory representation for the gaps of I_1-, I_2-, I_3-bins allows the finding of the right element in no more than $\lceil \log_2 n \rceil$ binary comparisons and the update operations may be performed by using no more than $\lceil \log_2 n \rceil$ comparisons [23]. This same principle is valid for the heaps maintaining the I_3-and I_4-elements [33]. □

7.2. Performance ratio. In the following items we assume that

- N_j is the total number of I_j-elements in the input list L.
- H is the size of the best matching between I_2-and I_3-elements, that is, the maximum number of pairs $x_2 \in I_2$, $x_3 \in I_3$ which can be coupled. Note that this corresponds to the maximum matching in a bipartite graph $G = (N_2 \cup N_3, E)$ so that
  - $N_2 = \{x \in I_2\}$,
  - $N_3 = \{x \in I_3\}$,
  - $E = \{(x, y) | x \in N_2, y \in N_3, size(x) + size(y) \leq 1\}$.
- K is the size of the best matching between I_1-and I_4-elements, that is, the maximum number of pairs $x_1 \in I_1$, $x_4 \in I_4$ which can be coupled. Please note that this corresponds to the maximum matching in a bipartite graph $G = (N_1 \cup N_4, E)$ so that
  - $N_1 = \{x \in I_1\}$,
  - $N_4 = \{x \in I_4\}$,
  - $E = \{(x, y) | x \in N_1, y \in N_4, size(x) + size(y) \leq 1\}$.

LEMMA 7.11. *If S is not empty after all the elements have been considered, then $R_{A_2} < \frac{4}{3}$.*

Proof. By hypothesis, since there is at least one group with size $\leq \frac{1}{4}$ which cannot be moved to a different gap, all of the I_0-, I_1-, I_2-, I_3-bins have a gap $< \frac{1}{4}$.

[10] If there exists a $b \in I_2$ such that $size(x) + size(b) \leq 1$.

Moreover, each I_k-bin ($k \geq 4$) has a gap $< \frac{1}{4}$.
Consequently, the maximum gap in each bin is $< \frac{1}{4}$, and

$$R_{A_2} < \frac{4}{3}. \qquad \square$$

LEMMA 7.12. *If $H = 0$ and S is empty after all the elements have been considered, then $R_{A_2} \leq \frac{4}{3}$.*

Proof. We can observe two different situations:

- In the input list there is no I_4-element.
  In this case A_2 uses no more than $N_0 + N_1 + N_2 + \frac{N_3}{2}$ bins, since no I_2-element or I_3-element could be inserted in any bin already containing an I_1-element or an I_0-element and no more than two I_3-elements may be inserted in any other bin. OPT cannot use fewer bins, since $H = 0$ implies that no bin can contain both an I_2-element and an I_3-element. Therefore

$$R_{A_2} = 1.$$

- In the input list there were some I_4-elements.
  Let $\alpha = N_1 + N_2 + \frac{N_3}{2}$. In this case we can still obtain two situations:
  $N_4 \leq \alpha$ then
  - OPT uses at least $\alpha + N_0$ bins.
  - A_2 uses no more than $N_0 + \alpha + \frac{N_4}{3}$ bins, in case it does not insert any I_4-element into some other bin. Then $A_2 \leq N_0 + \alpha + \frac{\alpha}{3} = N_0 + \frac{4}{3}\alpha$ and so

$$R_{A_2} \leq \frac{\frac{4}{3}\alpha + N_0}{\alpha + N_0} < \frac{4}{3}.$$

  $N_4 > \alpha$ then let $\beta = N_4 - \alpha$.
  - OPT cannot use fewer than $N_0 + \alpha + \frac{\beta}{3}$ bins, since no more than one I_4-element can fit into an I_1-, I_2-, or I_3-bin.
  - A_2 uses no more than $N_0 + \alpha + \frac{N_4}{3}$ bins. Then $A_2 \leq N_0 + \alpha + \frac{\alpha+\beta}{3} = N_0 + \frac{4}{3}\alpha + \frac{\beta}{3}$. Therefore we have

$$\frac{A_2}{OPT} \leq \frac{\frac{4}{3}\alpha + \frac{\beta}{3} + N_0}{\alpha + \frac{\beta}{3} + N_0} < \frac{4}{3}. \qquad \square$$

LEMMA 7.13. *If $H \neq 0$ and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there are no I_k-elements ($k \geq 4$), then $R_{A_2} < \frac{5}{4}$.*

Proof. OPT uses at least
N_0 bins to pack all the I_0-elements;
N_1 bins to pack all the I_1-elements;
N_2 bins to pack all the I_2-elements;
$\frac{N_3 - H}{2}$ bins to pack all the I_3-elements, since H of them are inserted in the gap of the H I_2-elements.
Hence,

$$OPT \geq N_0 + N_1 + N_2 + \frac{N_3}{2} - \frac{H}{2}.$$

A_2 uses no more than

N_0 bins to pack all the I_0-elements;
N_1 bins to pack all the I_1-elements;
N_2 bins to pack all the I_2-elements;
$\frac{N_3-\frac{H}{2}}{2}$ bins to pack all the I_3-elements, since $\frac{H}{2}$ of them are necessarily packed with a corresponding I_2-element [22].

Hence

$$A_2 \leq N_0 + N_1 + N_2 + \frac{N_3}{2} - \frac{H}{4}.$$

In conclusion, since $H \leq N_3$ and $H \leq N_2$, it follows that

$$\frac{A_2}{OPT} \leq \frac{4N_0 + 4N_1 + 4N_2 + 2N_3 - H}{4N_0 + 4N_1 + 4N_2 + 2N_3 - 2H} \leq \frac{4N_0 + 4N_1 + 5N_2 + 2N_3 - 2H}{4N_0 + 4N_1 + 4N_2 + 2N_3 - 2H} < \frac{5}{4}. \qquad \square$$

LEMMA 7.14. *If $H \neq 0$ and in the input sequence $L = \{a_1, a_2, \ldots, a_n\}$ there are some I_k-elements ($k \geq 4$), and S is empty after all elements have been considered, then $R_{A_2} \leq \frac{4}{3}$.*

Proof. Let B_i be the number of bins of level i used by OPT and let B'_i be the number of bins of level i used by A_2. We calculate the maximum number of bins used by A_2 as a function of the B_i's.

We can obtain two different situations:

- In the case that no I_4-bins are returned as output, since in such a case there are no I_5- and I_4-bins, we can say that $OPT = B_0 + B_1 + B_2 + B_3$:
  $B'_0 = B_0$;
  $B'_1 = B_1$;
  $B'_2 = B_2$;
  $B'_3 \leq B_3 + \frac{\frac{H}{2}}{2}$;
  Therefore, since $H \leq B_2$,

$$A_2 \leq B_0 + B_1 + \frac{5}{4}B_2 + \frac{5}{4}B_3 \leq \frac{5}{4}\text{OPT}.$$

- In the case that some I_4-bins are returned as output, since in such a case there are no I_5-bins, we can say that $OPT = B_0 + B_1 + B_2 + B_3 + B_4$:
  $B'_0 = B_0$;
  $B'_1 = B_1$;
  $B'_2 = B_2$;
  $B'_3 \leq B_3 + \frac{\frac{H}{2}}{2}$;
  $B'_4 = B_4 + [K + (B_2 - H) + B_3]\frac{1}{3} = \frac{B_2}{3} + \frac{B_3}{3} + B_4 + \frac{K}{3} - \frac{H}{3}$.
  Therefore, since $K \leq B_1$ and $H \geq 0$,

$$\begin{aligned} A_2 &\leq B_0 + B_1 + \tfrac{4}{3}B_2 + \tfrac{4}{3}B_3 + B_4 + \tfrac{k}{3} - \tfrac{H}{12} \\ &\leq B_0 + \tfrac{4}{3}B_1 + \tfrac{4}{3}B_2 + \tfrac{4}{3}B_3 + B_4 \leq \tfrac{4}{3}\ \text{OPT}. \end{aligned} \qquad \square$$

THEOREM 7.15. *Algorithm A_2 has a ratio of $R_{A_2} \leq \frac{4}{3}$.*

Proof. The proof is easily given by the previous lemmas. $\square$

THEOREM 7.16. *Algorithm A_2 is $A\ (7, 1.33)$.*

Proof. The proof is given by Corollary 7.8 and Theorem 7.15. $\square$

8. Conclusions and open problems. This paper focuses its attention on the possibility of maintaining a guaranteed approximation of the optimal solution for the online bin-packing problem in terms of time computation and element movements and with a limited reorganization of the previous solutions.

The problem is equivalent to the one considered in the more general bin-packing model where the elements can move (for a limited number of times) from the bin they are currently assigned to. Please note that this model still fits the general definition of online algorithms.

It would be interesting to see if these algorithms frequently touch any particular element and move it many times: it is also possible to demonstrate that the first algorithm (A_1) moves an element no more than once.

Contrarily to the offline model, the requests arriving to this model reach it online and the bins are always ready to be closed with no additional effort. This model is suitable in many different fields (for example in multiprocessor storage management and in packing trucks).

In the environment of this less restricted model, we have presented two new algorithms with the best approximation ratio available at this time, respectively, for linear and $O(n \log n)$ algorithms.

These algorithms are also "more" online than all the other linear space online bin-packing algorithms, because they allow the return of a fraction of the bins before the end of the execution.

There are still a lot of problems which remain to be solved in this area. First, it would be interesting to check if algorithms more efficient than A_2 can be found (as far as approximation ratio and/or time complexity are concerned). Generally speaking, it would also be interesting to define the lower bounds of the approximation ratio of the $O(n \log n)$ algorithms that allow the element to freely move between the bins.

It would also be interesting to verify if there is some kind of relation between the (amortized) number of movements allowed at the arrival of each input element and the asymptotic performance ratio. In other words: is there an algorithm that for each ϵ constant is $A(\frac{1}{\epsilon}, f(\epsilon))$?

It would also be useful to gain more knowledge about whether the approximation ratio can be maintained and if it is possible to send output the bins containing "enough" of the elements contained therein (e.g., what happens when all the bins with gap less than $\frac{k}{3}$ ($k \leq 1$) are send output?).

Other interesting questions concern the capacity of deleting elements from the bins maintaining the guaranteed algorithm performance (the target is to minimize the space wasted considering the actual element involved) and the analytical evaluation of the average performance of this algorithm compared to the ones characterized in [27], [30], [29], [32]. It would be very interesting to study the performances of the algorithm as a function of the number of movements admitted.

Acknowledgments. We are grateful to two anonymous referees for their useful remarks and to Giuseppe Alesio and Gennaro Gravina for useful discussions.

We also wish to thank Lucio Bianco of CNR IASI, Rome, for suggesting the transportation applications.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[2] J. L. Bentley, D. S. Johnson, F. T. Leighton, and C. C. McGeoch, *An experimental study of bin packing*, in Proceedings of the 21st Annual Allerton Conference on Communication, Control, and Computing, University of Illinois, Urbana, IL, 1983, pp. 51–60.

[3] J. L. Bentley, D. S. Johnson, F. T. Leighton, C. C. McGeoch, and L. A. McGeoch, *Some unexpected expected behavior results for bin-packing*, in Proceedings of the 16th ACM Symposium on the Theory of Computing, Washington, DC, 1984, pp. 279–288.

[4] A. Borodin, N. Linial, and M. Saks, *An optimal online algorithm for metrical task systems*, in Proceedings of the 19th ACM Symposium on the Theory of Computing, New York, 1987, pp. 373–382.

[5] D. J. Brown, *An improved BL lower bound*, Inform. Process Lett., 11 (1980), pp. 37–39.

[6] D. J. Brown and P. Ramanan, *Online bin packing in linear time*, in Proceedings of the 1984 Conference on Information Sciences and Systems, Princeton University, Princeton, NJ, 1984, pp. 328–332.

[7] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin-packing—An updated survey*, in Algorithm Design for Computer System Design, G. Ausiello and M. Lucertini, eds., Springer-Verlag, New York, 1984, pp. 49–106.

[8] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin-packing: A survey*, in Approximation Algorithms of NP-Hard Problems, D. S. Hochbaum ed., PWS Publishing Company, Boston, 1996, pp. 46–93.

[9] E. G. Coffman, Jr., M. Hofri, K. So, and A. C. Yao, *A stochastic model of bin packing*, Informat. Control, 44 (1980), pp. 105–115.

[10] J. Csirik and D. S. Johnson, *Bounded space online bin-packing: Best is better than First*, in Proceedings of the Second ACM–SIAM Symposium on Discrete Algorithms, San Francisco,1991, pp. 309–319.

[11] G. Gambosi, A. Postiglione, and M. Talamo, *On the Online Bin-Packing Problem*, IASI Technical Report R.263, Roma, 1989.

[12] G. Gambosi, A. Postiglione, and M. Talamo, *New algorithms for online bin-packing*, in Algorithms and Complexity, G. Ausiello, D. Bovet, and R. Petreschi, eds., World Scientific Publishing, Roma, 1990, pp. 44–59.

[13] G. Gambosi, A. Postiglione, and M. Talamo, *Online maintenance of an approximate bin-packing solution*, Nordic J. Comput., 4 (1997), pp. 151–166.

[14] M. R. Garey, R. L. Graham, D. S. Johnson, and A. C. Yao, *Resource constrained scheduling as generalized bin-packing*, J. Combin. Theory Ser. A, 21 (1976), pp. 257–298.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, CA, 1979.

[16] P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting stock problem*, Oper. Res., 9 (1961), pp. 849–859.

[17] P. C. Gilmore and R. E. Gomory, *A linear programming approach to the cutting stock problem—Part* II, Oper. Res., 11 (1963), pp. 863–888.

[18] R. L. Graham, *Bounds for certain multiprocessing anomalies*, Bell System Tech. J., 45 (1966), pp. 1563–1581.

[19] R. L. Graham, *Bounds on multiprocessing timing anomalies*, SIAM J. Appl. Math., 17 (1969), pp. 416–429.

[20] N. Karmarkar and R. M. Karp, *An efficient approximation scheme for the one-dimensional bin-packing problem*, in Proceedings of the 23th IEEE Symposium on Foundation of Computer Science, Chicago, 1982, pp. 312–320.

[21] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. M. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–105.

[22] B. Korte and D. Hausmann, *Matroids and independence systems*, in Modern Applied Mathematics: Optimization and Operations Research, B. Korte, ed., North-Holland, Amsterdam, 1982, pp. 517–553.

[23] D. S. Johnson, *Fast algorithms for bin-packing*, J. Comput. System Sci., 8 (1974), pp. 272–314.

[24] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, *Worst-case performance bounds for simple one-dimensional packing algorithms*, SIAM J. Comput., 3 (1974), pp. 299–325.

[25] C. C. Lee and D. T. Lee, *A simple online bin-packing algorithm*, J. ACM, 3 (1985), pp. 562–572.

[26] F. M. Liang, *A lower bound for on-line bin packing*, Inform. Process. Lett., 10 (1980), pp. 76–79.

[27] R. Loulou, *Probabilistic behavior of optimal bin-packing solutions*, Oper. Res. Lett., 3 (1984), pp. 129–135.

[28] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, *Competitive algorithms for server problems*, J. Algorithms, 11 (1990), pp. 208–230.
[29] P. Ramanan and K. Tsuga, K., *Average-case of the modified harmonic algorithm*, Algorithmica, 4 (1989), pp. 519–533.
[30] P. Ramanan, *Average-case analysis of the SMART-NEXT-FIT algorithm*, Inform. Process. Lett., 31 (1989), pp. 221–225.
[31] P. Ramanan, D. J. Brown, C. C. Lee, and D. T. Lee, *On-line bin-packing in linear time*, J. Algorithms, 10 (1989), pp. 305–326.
[32] P. W. Shor, *The average-case of some on-line algorithms for bin-packing*, Combinatorica, 6 (1986), pp. 179–200.
[33] R. E. Tarjan, *Data Structures and Network Algorithms*, CMBS-NSF Regional Conf. Ser. in Appl. Math. 44, SIAM, Philadelphia, 1983.
[34] A. C. Yao, *New algorithms for bin-packing*, J. ACM, 27 (1980), pp. 207–227.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1552–1578

TIGHT BOUNDS FOR SEARCHING A SORTED ARRAY OF STRINGS*

ARNE ANDERSSON†, TORBEN HAGERUP‡, JOHAN HÅSTAD§, AND OLA PETERSSON¶

Abstract. Given a k-character query string and an array of n strings arranged in lexicographical order, computing the rank of the query string among the n strings or deciding whether it occurs in the array requires the inspection of

$$\Theta\left(\frac{k\log\log n}{\log\log(4+\frac{k\log\log n}{\log n})}+k+\log n\right)$$

characters in the worst case.

Key words. string searching, character comparisons, dictionaries, potential functions, lower bounds

AMS subject classifications. 68Q17, 68Q25, 68W05, 68W40

PII. S0097539797329889

1. Introduction.

Given n strings, each of k characters, arranged in lexicographical order (i.e., a string precedes a string from which it differs if it has the smaller character in the first position in which the two strings differ), how many characters must we inspect to determine whether a k-character query string is present? We assume that the n strings are given in a $k \times n$ array and that no extra information is available. If k is a constant, we can solve the problem with $\Theta(\log n)$ character inspections by means of binary search, and this is optimal; but what happens for larger values of k? In the presence of preprocessing and extra storage, there are efficient methods, such as using a tree, each node of which can be implemented as a weighted tree as suggested by Mehlhorn [7, Sect. III.6.3], or the suffix array of Manber and Myers [6]; but what if we are given just the sorted strings?

The question is a basic one; we are simply asking for the complexity of searching a dictionary for a string, where the common assumption that entire strings can be compared in constant time is replaced by the assumption that only single characters can be compared in constant time. For sufficiently long strings, the latter assumption seems more realistic. At first glance the problem may appear easy—some kind of

*Received by the editors November 12, 1997; accepted for publication (in revised form) September 5, 2000; published electronically November 28, 2000.
http://www.siam.org/journals/sicomp/30-5/32988.html

†Computing Science Department, Information Technology, Uppsala University, Box 311, SE–751 05 Uppsala, Sweden (arnea@csd.uu.se). Part of this work was done while this author was at Lund University.

‡Fachbereich Informatik, Johann Wolfgang Goethe-Universität Frankfurt, D–60054 Frankfurt am Main, Germany (hagerup@informatik.uni-frankfurt.de). Part of this work was done while this author was with the Max-Planck-Institut für Informatik in Saarbrücken, Germany and was supported by the ESPRIT Basic Research Actions Program of the EU under contract No. 7141 (project ALCOM II).

§Department of Numerical Analysis and Computing Science, Royal Institute of Technology, SE–100 44 Stockholm, Sweden (johanh@nada.kth.se). Current address: School of Mathematics, Institute for Advanced Study, 1 Einstein Drive, Princeton, NJ 08540 (johan@ias.edu). Part of this work was done while this author was visiting MIT.

¶Department of Mathematics, Statistics, and Computer Science, Växjö University, SE–351 95 Växjö, Sweden; and Department of Computer Science, Lund University, Box 118, SE–221 00 Lund, Sweden (sithtry@yahoo.com). Part of this work was done while this author was visiting Columbia University.

generalized binary search should do the trick. However, closer acquaintance with the problem reveals an unexpected intricacy.

Given the relevance of this problem, surprisingly few results have been reported. Hirschberg [4] indicated a lower bound of $\Omega(k+\log n)$ and upper bounds of $O(k \log n)$ and $O(k+n)$ and combined the upper bounds to derive a bound of $O(k \log(2+n/k))$, all of which is straightforward. A later publication by the same author [3] mentions a first nontrivial upper bound of $O(k \log n/\log k)$. Kosaraju [5] gave an algorithm with a running time of $O(k\sqrt{\log n} + \log n)$. The only nontrivial lower bound deals with constant factors: Kosaraju [5] showed that at least roughly $\log n + \frac{1}{2}\sqrt{k \log n} = O(k + \log n)$ characters need to be inspected. We determine the exact complexity of the problem, up to a constant factor.

Before formulating our result, we describe two relevant computational problems more formally. For all integers $k, n \geq 1$ and all ordered sets Σ, an instance of the *string-ranking problem* of size $k \times n$ over the alphabet Σ is given by a list $s, s_1, \ldots, s_n$ of $n+1$ strings, each consisting of k characters drawn from Σ, such that $s_1 \preceq \cdots \preceq s_n$, where $\preceq$ denotes the lexicographical order derived from the order on Σ. The task is to compute $|\{j : 1 \leq j \leq n \text{ and } s_j \preceq s\}|$, i.e., the rank of s in the multiset $\{s_1, \ldots, s_n\}$. An instance of the *string-membership problem* of size $k \times n$ over Σ is given by a list of the same form, and the task is to output "yes" if $s = s_j$ for some $j \in \{1, \ldots, n\}$, and "no" otherwise.

The string-membership problem clearly is no harder than the string-ranking problem in the sense that after solving the latter, we can solve the former after inspecting at most k additional characters. An algorithm for the string-ranking problem also allows us to determine the indices of the first and last occurrences, if any, of the query string. As implied by our result, stated below, these problems in fact all have the same deterministic complexity, up to a constant factor. The logarithm function to base 2 is denoted by "log."

THEOREM 1.1. *For all integers $k \geq 1$ and $n \geq 4$ and all ordered sets Σ of at least two elements, the solution of instances of size $k \times n$ of the string-ranking problem and of the string-membership problem over the alphabet Σ requires the inspection of*

$$\Theta\left(\frac{k \log\log n}{\log\log\left(4 + \frac{k \log\log n}{\log n}\right)} + k + \log n\right)$$

characters in the worst case.

As a curiosity we note that for the special case $k = \Theta(\log n)$, natural in view of the lower bound of $\Omega(k + \log n)$, we get a tight bound of

$$\Theta\left(\frac{\log n \log\log n}{\log\log\log\log n}\right),$$

which would have been hard to guess in advance.

This paper is based on the two conference presentations [1] and [2]. The proofs given here are significantly shorter and simpler due to the use of potential functions. All four authors contributed equally to both upper-bound and lower-bound parts of the paper.

After introducing notation and terminology in section 2, we provide intuition in section 3, and we prove the upper bound in section 4 and the lower bound in section 5. Sections 3, 4, and 5 can be read independently of each other.

2. Preliminaries.

2.1. The leftmost-all-1 problem. To simplify the presentation, we introduce a simplified searching problem called the *leftmost-all*-1 problem and demonstrate the upper bound first for this problem. For all integers $k, n \geq 1$, the leftmost-all-1 problem of size $k \times n$ is the special case of the string-ranking problem of size $k \times n$ obtained by fixing the alphabet to be $\{0, 1\}$ and the query string to be $1^{k-1}0$ (i.e., $k-1$ 1's followed by one 0). We assume an instance of the leftmost-all-1 problem of size $k \times n$ to be given in a $k \times n$ matrix I in the following way: for $i = 1, \ldots, k$ and $j = 1, \ldots, n$, $I[i, j]$ is the ith character of the jth string; i.e., each string is written vertically from top to bottom, and the strings are ordered from left to right. The rows and columns of I are numbered from top to bottom and from left to right, respectively, the number of a row or column also being called its *index*. The task is to determine the number of columns in I that contain at least one 0. An alternative formulation, which gives the problem its name, is that the task is to compute one less than the index of the leftmost column in I containing the string 1^k, or n if there is no such column. An algorithm for the leftmost-all-1 problem is said to perform a *probe* when it examines an entry in I, and we charge the algorithm according to how many probes it performs.

2.2. Surface and fence algorithms. In this subsection we introduce terminology convenient for discussing solutions to the leftmost-all-1 problem. The terminology is illustrated in Figures 2.1 and 2.2.

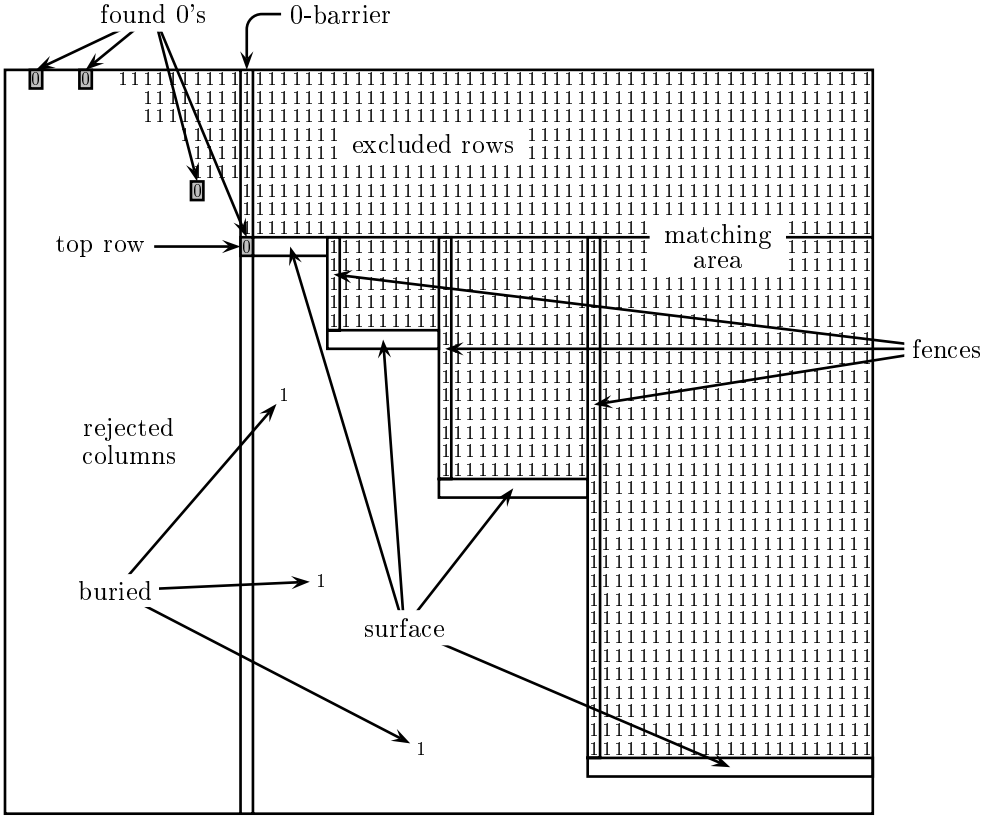


FIG. 2.1. *Terminology related to surface algorithms.*

Consider an algorithm for the leftmost-all-1 problem that inspects entries in a $k \times n$ input matrix I one by one. Once the algorithm has established that certain positions in I contain 1's, it may be able to deduce from the sortedness of I and without actual probes that certain other positions in I must also contain 1's. Specifically, let $1 \leq r \leq k$ and $1 \leq c \leq n$, and suppose that the algorithm has already established that $I[i, c] = 1$ for $i = 1, \ldots, r$. Then, clearly, we also have $I[i, j] = 1$ for $i = 1, \ldots, r$ and $j = c + 1, \ldots, n$. We say that these additional occurrences of 1 in I are deduced by 1-*extension*.

If a column in I is known to contain at least one 0 because a 0 was found in the column itself or in a column to its right, the column and all positions in the column are said to be *rejected*; such a column cannot be the leftmost column containing the string 1^k. The rightmost rejected column is called the 0-*barrier*; initially, before any columns have been rejected, we take the 0-barrier to be an imaginary column of index 0. The remaining positions are classified as follows. If a nonrejected position is known to contain a 1, either because it was explicitly probed or by way of 1-extension, it belongs to the *matching area*—the entries in this area are known to match those of the string 1^k. A nonrejected position outside of the matching area is a *surface position* if all positions above it belong to the matching area, and a *buried position* otherwise. Of course, each column contains at most one surface position.

For $r = 1, \ldots, k$, row r is said to be *excluded* if none of the rows $1, \ldots, r$ contains a surface position. The part of an excluded row outside of the rejected columns is known to contain only 1's. If and when an algorithm for the leftmost-all-1 problem manages to exclude the last row, it can therefore output the number of rejected columns as its result. As long as at least one row is not excluded, we define the *top row* to be the topmost nonexcluded row. Initially, no columns are rejected and no rows are excluded, row 1 is the top row, all positions in the top row are surface positions, and all other positions are buried positions.

A *surface probe* is a probe that inspects the entry in a surface position, and a *surface algorithm* is an algorithm all of whose probes are surface probes. We call a surface probe *successful* if it returns a 1 (the string probed still matches the string 1^k) and *unsuccessful* if it returns a 0. The following observations are helpful.

Whenever a surface algorithm finds a 1 in some position, that position and all positions above it and to its right subsequently are part of the matching area. If the position containing the 1 is in the column next to the 0-barrier, the row containing the 1 is excluded and the top row moves down by one position.

Whenever a surface algorithm finds a 0 in a particular column, the 0-barrier moves to that column, and that column and all nonrejected columns to its left are rejected. Since the newly rejected columns may have contained all surface positions of some rows, this may also cause the top row to move downwards. The new top row will be either the row in which the 0 was found or a row below it; the latter happens when the position immediately to the right of the 0 belongs to the matching area.

It can be seen that at all times during the execution of a surface algorithm, a position above or to the right of a position in the matching area also belongs to the matching area—we express this by saying that the matching area is *monotonic*—so that the boundary between the matching area and the remaining positions forms a "staircase" going down and to the right. The part of a column contained in the matching area but outside of the excluded rows is called a *fence* if at least one position immediately to its left is not part of the matching area (i.e., a fence resides in a column where the matching area becomes "deeper"). A *fence algorithm* is a surface algorithm

each of whose probes is a *top-row probe*, i.e., a probe in the (current) top row, or an *extending probe*, i.e., a probe of an entry immediately below an existing fence.

The height of a fence F, denoted $|F|$, is defined as the number of positions contained in F. It is obvious that every fence is strictly higher than every fence to its left. When a fence algorithm executes a probe below a fence F, we will say that it attempts to *extend* F. If the extension is successful, the height of F usually increases by 1, i.e., the position probed and the positions belonging to F before the probe form a fence that we identify with F; two exceptions are noted below. If the extension is unsuccessful, F and all fences to the left of F (equivalently, shorter than F) are *excluded*, since they are now completely contained in the excluded rows (and in the rejected columns), while the height of every other fence decreases by the number of rows excluded.

An exceptional case of a successful extension of a fence F or of a successful top-row probe that creates F occurs if, before the probe, some fence F' to the right of F had the same height as F after the probe; i.e., a "stair" of the "staircase" vanishes. In this case we will say that F and F' *merge* to create a new fence. We shall frequently need to distinguish between new fences that result from merges and new fences that result from successful top-row probes (without a merge or before a merge triggered by the probe); we shall say that the latter fences are created *from scratch*.

Another exceptional situation happens when a fence algorithm finds a 1 in the column next to the 0-barrier or a 0 in the column immediately to the left of some fence. In either case, no new fence is created, no merge takes place, one or more rows are excluded, and one or more fences may be excluded. We call a probe *excluding* if it causes one or more rows to be excluded.

The *gap* of a fence F is defined as its distance from the 0-barrier, i.e., as $c_F - c_Z$, where c_F and c_Z are the indices of the column containing F and of the 0-barrier, respectively. Observe that the gap of every fence is at least 2. The *index* of a fence is one more than the number of fences strictly to its right; in other words, the fences are numbered from right to left. The proofs of both the upper bound and the lower bound associate with each fence F an integer *weight*, denoted $\|F\|$. We define the *cumulative weight* of a fence as the sum of its own weight and the weights of all fences of strictly smaller index; i.e., if the weights are summed from right to left, the partial sum obtained for each fence is its cumulative weight.

We will denote by $\mathcal{F}$ the (current) list of fences, ordered from left to right. For example, if $\mathcal{F} = (F_N, \ldots, F_1)$, then F_N and F_1 are the leftmost and rightmost fences, respectively, and $1 \le |F_N| < \cdots < |F_1|$. For $i = 2, \ldots, N$, we call F_i the *left neighbor* of F_{i-1} and F_{i-1} the *right neighbor* of F_i.

Figure 2.2 illustrates the various possibilities for a surface probe. Rejected columns and excluded rows are separated from the remaining positions by double vertical and horizontal lines, respectively, the matching area is filled with 1's, and unknown entries that are still of interest are represented by dark squares. Consider the situation in (a), in which we have two fences: F_1 of height 3 in column 10 and F_2 of height 2 in column 7. The six situations in (b)–(g) show possible results of performing one probe, starting from (a). The contents of the probed position are circled. (b) shows a successful top-row probe, creating a new fence of height 1 from scratch in column 5. (c) shows an unsuccessful top-row probe in column 5. (d) shows a successful extension of F_2, leading to a merge of F_1 and F_2. (e) shows an unsuccessful attempt to extend F_2. (f) shows a successful top-row probe in the column next to the 0-barrier; no new fence is created, but one row is excluded. (g) shows an unsuccessful top-row probe

in the column next to the leftmost fence F_2; F_2 and the rows that it spans are excluded. Finally, starting from the situation in (b), (h) shows a successful top-row probe performed when the leftmost fence is of height 1; the new fence immediately merges with its right neighbor to form a new leftmost fence.

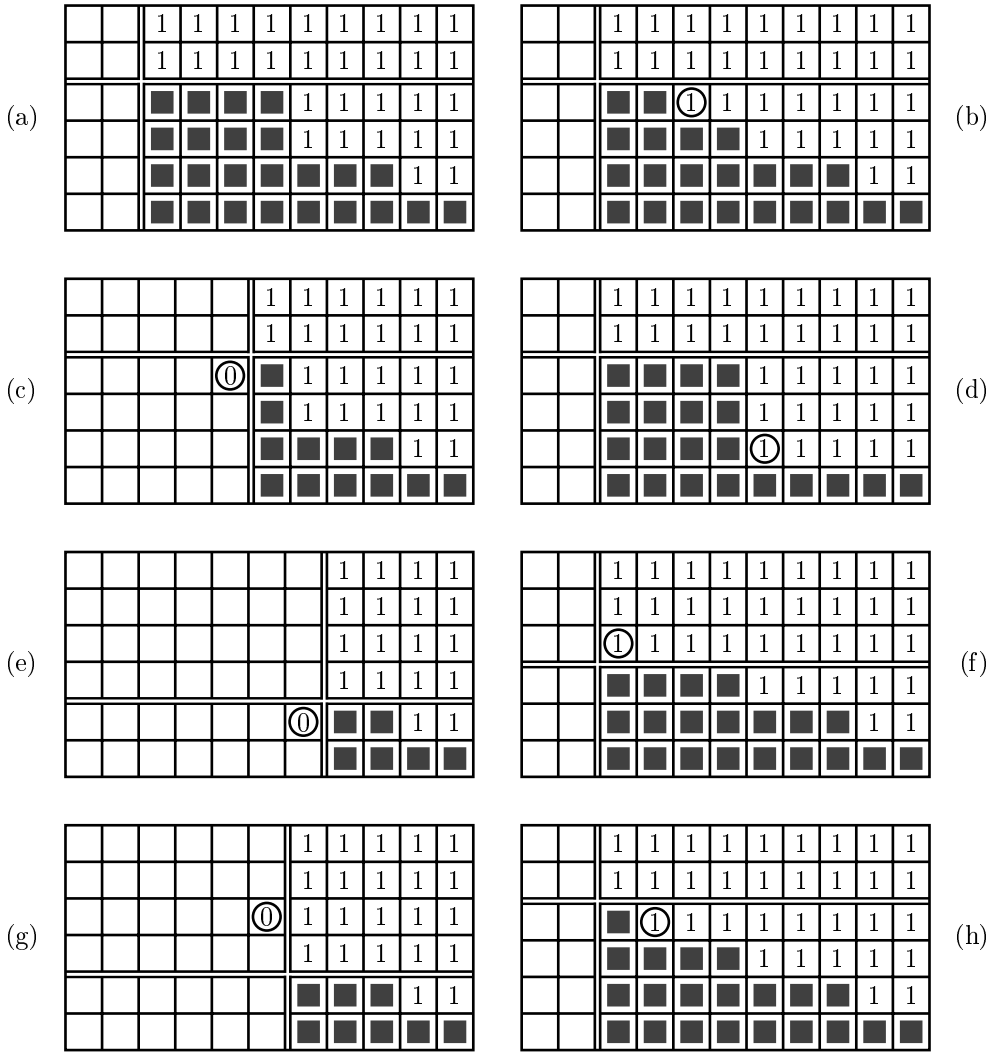


FIG. 2.2. *Possible outcomes of a surface probe.*

3. Intuitive sketch. Before delving into the formal arguments, let us try to provide intuition behind some important parameters used in the proofs of the upper and lower bounds. The reader should keep in mind that we do not claim to *prove* anything in this section, but merely to show how one may arrive at the parameter values of the proofs through plausible reasoning. Exact values of the parameters and rigorous proofs are given in the remainder of the paper. This section can be skipped without loss of continuity.

First, fence algorithms do come naturally—and one conclusion of our work is that they are optimal, up to a constant factor—so let us concentrate on fence algorithms.

Since the goal of a fence algorithm can be viewed as that of excluding all k rows, a natural measure of its progress is the number of rows that it has managed to exclude so far. Excluding a row comes at a certain price, however, since, in some sense, all probes of positions in a row become worthless when the row is excluded. Our main concern, therefore, will be not to perform too many probes per row, at least in an amortized sense. A symmetric argument could be made concerning rejected columns instead of excluded rows, but this appears to offer less useful insights.

Important properties of a fence algorithm are the number of fences, their spatial distribution, and their heights. Suppose that we aim for a bound of the form kt for some value of t. That is, we wish to spend an average of t probes per row. We must analyze what happens when we fail to extend a fence. One the one hand, we gain one or more rows, i.e., they are excluded, but on the other hand we lose probes. We identify two kinds of losses.

1. All fences contained in the excluded rows disappear, and thus all probes spent to create these short fences are now useless.
2. If N fences remain after the row exclusions, N probes per row used in building these fences are now useless. This suggests that we should keep the number of fences bounded by t.

Let us define some quantities needed to analyze losses of the first kind. When creating a new fence from scratch, it is natural to probe the middle entry of the unknown part of the top row, that is, to make one step of a binary search for locating the rightmost 0 in that row. Suppose that we perform more such probes, all of which find 1's. In effect, the new fence moves leftward and away from its neighboring fence, while still being of height 1. It is intuitively clear that as the new fence F moves to the left, it becomes more and more valuable to the fence algorithm. In order to quantify this, we count the number of binary-search steps used to move F to its current position and call this number the *weight* of F, denoted $\|F\|$. When two fences merge, we define the weight of the resulting fence to be the sum of the weights of the two fences from which it is formed. It can be seen that with this rule, the total weight of all fences can grow to around $\log n$, but not beyond that.

A second important quantity associated with a fence F is the total number of probes spent to construct F. We call this the *investment* in F and denote it by $\mathit{Invest}(F)$. Investments also add up when fences merge. Investments differ from weights in that vertical extensions of fences are counted in the former, but not in the latter. As a consequence, $\mathit{Invest}(F) \geq \|F\|$ for every fence F.

As observed above, we want at most t fences, and a natural way to spread the fences is to make sure that, going from left to right, they are of exponentially increasing weights. If weights increase by a factor of a from each fence to the next, then we should have $a^t = \log n$ or, equivalently,

$$\log a = \frac{\log \log n}{t}, \tag{3.1}$$

since the maximum weight of any fence is around $\log n$.

Consider a loss of the first kind. As stated above, we choose the weights of the existing fences to be exponentially increasing from left to right, and it turns out that the same will be true for the investments in the fences. Hence a constant fraction of the loss is due to the disappearance of the tallest excluded fence; for simplicity, we assume that this is the only loss.

We want the height of a fence to be a function simply of its weight, and we denote the relevant function by T. Now consider the situation when we fail to extend a fence

F. We gain $T(\|F\|)$ rows and lose the investment $\mathit{Invest}(F)$. To keep the cost of t probes per row, this means that we need

$$T(\|F\|) \geq \mathit{Invest}(F)/t \geq \|F\|/t. \tag{3.2}$$

It is natural to assume that it is optimal for the first inequality to hold with equality. Under this assumption, let us analyze the key operation of merging two fences.

Consider two fences F and F' that merge, with F' to the left of F. The merge is caused by a number of probes that extend F'. Assume that before these probes are made, F and F' are of heights $T(\|F\|)$ and $T(\|F'\|)$, respectively, and that $tT(\|F\|)$ and $tT(\|F'\|)$ probes, respectively, have been invested in them. Furthermore, since the weight of the new fence is the sum of the weights of the old fences, assume that after its creation it is extended to height $T(\|F\| + \|F'\|)$. The extension needs $T(\|F\| + \|F'\|) - T(\|F\|)$ probes, so that afterwards the investment in the new fence will be

$$t(T(\|F\|) + T(\|F'\|)) + T(\|F\| + \|F'\|) - T(\|F'\|).$$

If we disregard the last term (which turns out to be insignificant) and observe that T grows at least linearly, by (3.2) we see that this expression is at least $(t+1)(T(\|F\|)+ T(\|F'\|))$. As the investment in the new fence should be at most $tT(\|F\| + \|F'\|)$, we obtain the relation

$$(t+1)(T(\|F\|) + T(\|F'\|)) \leq tT(\|F\| + \|F'\|).$$

Combining this with the relation

$$\|F\| + \|F'\| = (1 + 1/a)\|F\|,$$

which expresses the intended meaning of the parameter a, yields

$$(t+1)T(\|F\|) \leq tT((1 + 1/a)\|F\|)$$

or

$$(1 + 1/t)T(\|F\|) \leq T((1 + 1/a)\|F\|).$$

Setting $T(x) = dx^p$ for arbitrary constants $d > 0$ and $p \geq 1 + a/t$ is sufficient to satisfy this requirement, and choosing

$$T(x) = \frac{x^{1+a/t}}{t}$$

also takes care of the condition of (3.2).

Finally, the biggest possible weight of a fence (namely, $\log n$) should correspond to the biggest possible height of a fence (namely, k), i.e.,

$$T(\log n) = \frac{(\log n)^{1+a/t}}{t} = k.$$

Together with (3.1), this implies $a \log a = \log(kt/\log n)$. After some simplification, this and (3.1) yield suitable values for a and t.

Essentially, the algorithm can be derived from this discussion by defining everything precisely and adjusting a few constants. This is done in the next section.

The lower bound is proved by means of an adversary that keeps track of the investments made by an algorithm. Whenever the algorithm has not protected its investment by erecting tall enough fences, the adversary reveals information that makes the algorithm lose part of its investment at too high a cost. The adversary's actions basically force the algorithm to behave as the algorithm in the proof of the upper bound, or it will do worse.

4. The upper bound. In this section we first describe a fence algorithm that solves the leftmost-all-1 problem using a number of probes that is within the upper bound of Theorem 1.1. Later, in section 4.4, we extend the methods to solve the original string-ranking problem using at most twice as many probes, where a probe, in the case of the string-ranking and string-membership problems, is the inspection of a character in the sorted sequence of input strings.

Consider an input I of size $k \times n$, where $k \geq 1$ and $n \geq 4$ (the condition $n \geq 4$ simply ensures that $\log\log n$ is well defined and at least 1). We begin by defining a number of parameters in terms of k and n. First,

$$a = \sqrt{\max\left\{\log\left(\frac{k\log\log n}{\log n}\right), 4\right\}} \qquad \text{and} \qquad t = \left\lceil \frac{\log\log n}{\log a} \right\rceil + 2.$$

Second, for all real $x \geq 0$, take

$$T(x) = \frac{x^{1+ea/t}}{t},$$

where $e = 2.718\ldots$ is the base of the natural logarithm function. T maps the set of nonnegative real numbers to itself, and its derivative T' satisfies $T'(x) \geq 1/t$ for all $x \geq 1$. The only other properties of T of relevance to us are expressed in the two lemmas below.

LEMMA 4.1. $T(2\log n) = O(k + \log n/\log\log n)$.

Proof.

$$\begin{aligned} T(2\log n) = \frac{(2\log n)^{1+ea/t}}{t} &\leq \frac{(2\log n)^{1+ea\log a/\log\log n}\cdot \log a}{\log\log n} \\ &\leq 2^{1+2ea\log a}\cdot \log a \cdot \frac{\log n}{\log\log n} = O\left(2^{a^2}\cdot\frac{\log n}{\log\log n}\right). \end{aligned}$$

Since $2^{a^2}\cdot \log n/\log\log n = k$ if $a > 2$, the lemma follows. □

LEMMA 4.2. *Let x, y, and ϵ be nonnegative real numbers with $0 \leq \epsilon \leq 1$ and $y \geq x \geq \epsilon y$. Then*

$$T(x+y) \geq (1+\epsilon a/t)(T(x)+T(y)).$$

Proof. By the mean-value theorem, $e^{\epsilon/e} \leq 1+\epsilon$, and therefore

$$x + y \geq (1+\epsilon)y \geq e^{\epsilon/e}\cdot y$$

and

$$\begin{aligned} tT(x+y) &= (x+y)^{1+ea/t} \\ &\geq (e^{\epsilon/e} \cdot y)^{ea/t}(x+y) \\ &= e^{\epsilon a/t} \cdot y^{ea/t}(x+y) \\ &\geq (1+\epsilon a/t)y^{ea/t}(x+y) \\ &\geq (1+\epsilon a/t)(x^{1+ea/t} + y^{1+ea/t}) \\ &= (1+\epsilon a/t)t(T(x)+T(y)). \qquad \square \end{aligned}$$

Using the lemma with $x = y$ and $\epsilon = 0$, we obtain the following.

COROLLARY 4.3. *For all* $x \geq 0$, $T(2x) \geq 2T(x)$.

4.1. The algorithm. Whenever the fence algorithm to be described performs a top-row probe, it probes in the middle of the surface part of the top row. More precisely, if the leftmost and rightmost surface positions in the top row are in columns c_{L} and c_{R}, respectively, the top-row probe is performed in column $\lfloor (c_{\mathrm{L}} + c_{\mathrm{R}})/2 \rfloor$ or in column $\lceil (c_{\mathrm{L}} + c_{\mathrm{R}})/2 \rceil$. (If $c_{\mathrm{L}} + c_{\mathrm{R}}$ is odd, either choice is acceptable.) It will be convenient to allow probes below the actual input, i.e., in "row r" for $r > k$. By convention, such probes always return 1.

As mentioned earlier, the algorithm associates with each fence F a positive integer, called the *weight* of F and denoted by $\|F\|$. The *target height* of F is defined as $H(F) = \lceil T(\|F\|) \rceil$. We say that F is *of target height* if $|F| = H(F)$ and *below target height* if $|F| < H(F)$.

The algorithm repeatedly performs one probe using the procedure Probe, specified below, until all k rows have been excluded and then outputs the number of rejected columns, which is easily seen to be the correct answer even if fictitious probes below row k were performed. Upon entry to the procedure, the notation is assumed to be fixed so that $\mathcal{F} = (F_N, \ldots, F_1)$. (Recall that $\mathcal{F}$ is the list of all fences in the order from left to right.)

Probe:
  if some fence is below target height
  then perform an extending probe below the rightmost such fence
  else
    if $N \geq 2$ **and** $\|F_N\| > \|F_{N-1}\|/a$
    then perform an extending probe below F_N
    else perform a top-row probe.

In the interest of succinctness, the description given above does not specify the manipulation of fence weights. When a new fence is created from scratch, it is given weight 1. When two fences F and F' merge, the resulting fence acquires $\|F\| + \|F'\|$ as its weight. Finally, when a nonexcluding top-row probe encounters a 0 and $N \geq 1$, the weight of the leftmost fence is increased by 1. No other weight changes take place. In particular, only the leftmost fence ever changes its weight.

It may be helpful to visualize how a second fence is created by the algorithm. As long as there is only one fence and this fence is of height 1, the new fence created by a top-row probe that encounters a 1 immediately merges with the old fence. Informally, the net effect can be viewed as the old fence moving to the left and increasing its weight by 1. When the weight of the single fence has increased sufficiently for its target height to reach 2, the fence may be extended beyond height 1, after which a second fence can be created.

4.2. Properties of the algorithm.

LEMMA 4.4. *Suppose that $\mathcal{F} = (F_N, \ldots, F_1)$ before a probe that extends a fence F_i such that $2 \le i \le N$ and $\|F_i\| \le \|F_{i-1}\|/a$ before the probe. Then the probe will not cause F_i and F_{i-1} to merge.*

Proof. Consider the situation before the probe. F_i is below target height, whereas F_{i-1} is not, since otherwise F_i would not be extended. In particular, $T(\|F_i\|) \ge 1$. Since $a \ge 2$ and $T(x) \ge 2T(x/2)$ for all $x \ge 0$ (Corollary 4.3), $T(\|F_{i-1}\|) \ge 2T(\|F_i\|) \ge T(\|F_i\|) + 1$ and therefore $H(F_{i-1}) > H(F_i)$. The lemma follows. ∎

The following two lemmas are proved together by induction on the number of probes executed.

LEMMA 4.5. *At all times, with $\mathcal{F} = (F_N, \ldots, F_1)$, $\|F_i\| \le \|F_{i-1}\|/a \le \|F_{i-1}\|/2$ for $2 \le i < N$.*

LEMMA 4.6. *Every merge combines the two leftmost fences.*

Proof. Lemma 4.4 states that if $2 \le i < N$ and $\|F_i\| \le \|F_{i-1}\|/a$ before an extension of F_i, then the extension will not cause a merge. Conversely, if the condition of Lemma 4.5 holds before a merge that involves the leftmost fence, it will clearly hold afterwards. Since the condition of Lemma 4.5 is vacuously satisfied initially, while no fence is ever created from scratch unless $N \le 1$ or the claim holds even for $i = N$, it can be seen that the condition must hold at all times. ∎

LEMMA 4.7. *At all times, if $\mathcal{F} = (F_N, \ldots, F_1)$ and $1 \le i \le j < N$, then $\|F_j\| \le a^{i-j}\|F_i\|$ and $T(\|F_j\|) \le 2^{i-j}T(\|F_i\|)$.*

Proof. The first part of the lemma is obtained by $j - i$ applications of Lemma 4.5. Since $x \le y/2$ implies $T(x) \le T(y)/2$ for all $x, y \ge 0$ (Corollary 4.3), the second part can be proved in a similar way. ∎

LEMMA 4.8. *At all times, if $\mathcal{F} = (F_N, \ldots, F_1)$ and $N \ge 2$, then $\|F_N\| \le \|F_{N-1}\|$.*

Proof. Initially, the claim is vacuously satisfied. We show that if it holds immediately before a probe, then it holds immediately after the probe. Let $\mathcal{F} = (F_N, \ldots, F_1)$ before the probe.

When a fence is created from scratch, it is given weight 1, and the claim is clearly satisfied. An unsuccessful top-row probe may increase $\|F_N\|$ by 1 but is not performed unless $N \le 1$ or $\|F_{N-1}\| \ge a\|F_N\| \ge \|F_N\| + 1$, so that the claim also holds after the probe. Lemma 4.6 states that every merge combines the two leftmost fences. By induction and Lemma 4.5, their combined weight is bounded by the weight of the right neighbor, if any, of the resulting fence. In symbols: $\|F_N\| + \|F_{N-1}\| \le 2\|F_{N-1}\| \le \|F_{N-2}\|$. Finally, and again using Lemma 4.5, the claim is easily seen to hold after the exclusion of one or more fences. ∎

LEMMA 4.9. *At all times, with $\mathcal{F} = (F_N, \ldots, F_1)$, $\sum_{j=i}^{N} T(\|F_j\|) \le 2T(\|F_i\|)$ for $i = 1, \ldots, N$.*

Proof. Assume that $i < N$, since otherwise the claim is trivial. By Lemmas 4.7 and 4.8, $T(\|F_j\|) \le 2^{i-j}T(\|F_i\|)$ for $j = i, \ldots, N-1$ and $T(\|F_N\|) \le T(\|F_{N-1}\|)$. Thus

$$\sum_{j=i}^{N} T(\|F_j\|) \le \left(\sum_{j=i}^{N-1} 2^{i-j} + 2^{i-N+1} \right) \cdot T(\|F_i\|) = 2T(\|F_i\|). \qquad \blacksquare$$

LEMMA 4.10. *At all times, with $\mathcal{F} = (F_N, \ldots, F_1)$, $H(F_i)/2 \le |F_i| \le H(F_i)$ for $i = 1, \ldots, N-1$.*

Proof. Initially, the claim is vacuously satisfied. We show that if it holds immediately before a probe, then it holds immediately after the probe. Let $\mathcal{F} = (F_N, \ldots, F_1)$ before the probe.

Consider first the upper bound of the lemma, i.e., the claim that only the leftmost fence can have a height exceeding its target height. By induction and since the weight of a fence never decreases, when a fence becomes the leftmost fence, because of row exclusions or a merge or because the fence was just created from scratch, its height will not exceed its target height. Moreover, once the algorithm starts extending a fence beyond its target height, the fence must be the leftmost fence, and the algorithm will continue to extend it until an extension is unsuccessful, a merge takes place, or the algorithm terminates. In all cases, the offending fence disappears before it can acquire a left neighbor.

The other inequality of the lemma states that only the leftmost fence can have a height below half its target height. Since no other fence is below target height when a new fence is created from scratch, while only the leftmost fence can increase its weight, this could be invalidated only by row exclusions, which decrease fence heights. Before a top-row probe is performed, no fence is below target height, so that even if the probe excludes one row, the height of every surviving fence after the probe will still be at least half its target height. Consider, therefore, a probe that excludes fences $F_N, \ldots, F_i$, and let j be arbitrary with $1 \le j < i-1$. Before the probe, F_{i-1} and F_j are both of target height, by the upper bound established above, so that at this time, by Corollary 4.3 and Lemma 4.5,

$$\begin{aligned} |F_j| = H(F_j) &\ge T(\|F_j\|) \ge T(2\|F_{i-1}\|) \\ &\ge 2T(\|F_{i-1}\|) \ge 2(H(F_{i-1}) - 1) = 2(|F_{i-1}| - 1). \end{aligned}$$

Hence, before the probe, $|F_i| \le |F_{i-1}| - 1 \le |F_j|/2$, so that after the probe we still have $|F_j| \ge H(F_j)/2$. This holds for all j with $1 \le j < i-1$; i.e., the lower bound of the lemma is satisfied. ∎

The following consequence of the previous lemma and its proof will be useful later.

COROLLARY 4.11. *Immediately before the algorithm performs a probe, every fence strictly to the right of the position probed is of target height.*

LEMMA 4.12. *At all times, if $\mathcal{F} = (F_N, \ldots, F_1)$, then for $i = 1, \ldots, N$, the gap of F_i is at most $2^{\lceil \log n \rceil + 1 - w_i}$, where $w_i = \sum_{j=1}^{i} \|F_j\|$ is the cumulative weight of F_i.*

Proof. The proof is by induction on the number of probes performed. Since the gap of a fence never increases, the proof amounts to a simple inspection of the cases in which a fence is created or its cumulative weight increases.

Consider first the case in which a fence F is created from scratch. If F is created without a right neighbor (i.e., it is the only fence), it has a cumulative weight of 1, and its gap is bounded by n, so the claim is satisfied. Immediately after the creation from scratch of a fence F with a right neighbor F', the cumulative weight w of F is one more than the cumulative weight w' of F', and by the policy of placing new fences created from scratch in the middle of the surface part of the top row, the gap of F is at most $1/2$ plus half the gap of F'. By induction, therefore, the gap of F is at most $\lfloor 2^{\lceil \log n \rceil + 1 - w'}/2 + 1/2 \rfloor = 2^{\lceil \log n \rceil + 1 - w}$ (recall that the gap of F' is at least 2), and the claim continues to hold.

When two fences merge, the cumulative weight of every fence after the merge is the same as the cumulative weight of the fence residing in the same column before the merge. Finally, an unsuccessful top-row probe that increases the weight of the

leftmost fence by 1 also reduces its gap to at most 1/2 plus half the old gap, and the claim continues to hold as above. □

COROLLARY 4.13. *No weight of a fence ever exceeds* $\lceil \log n \rceil + 1 < 2 \log n$.

LEMMA 4.14. *The number of fences never exceeds* t.

Proof. If at some point more than t fences exist, we can apply Lemma 4.7 with $i = 1$ and $j = t$ and derive the following contradiction from the previous corollary:

$$1 \leq \|F_t\| \leq a^{1-t} \cdot \|F_1\| < a^{-\log\log n/\log a - 1} \cdot 2 \log n = 2/a. \qquad \square$$

4.3. Analysis of the number of probes. In this subsection we complete the analysis of the algorithm by showing the number of probes performed to be $O(kt + \log n)$. The approach is simple: we define a potential function Φ of the state of the algorithm, initially zero, show that every probe increases Φ by at least one, and bound the maximum value of Φ.

Consider a point in time during the execution of the algorithm at which $\mathcal{F} = (F_N, \ldots, F_1)$, and let X_{C} and X_{R} be the numbers of rejected columns and of excluded rows, respectively. The potential function Φ has the form $\Phi = \Phi_1 + \Phi_2 + \Phi_3 + \Phi_4$, where $\Phi_1 = \log(n/(n - X_{\mathrm{C}}))$ measures the ratio of original columns to remaining columns, $\Phi_2 = 13tX_{\mathrm{R}}$ is proportional to the number of excluded rows, $\Phi_3 = \sum_{i=1}^{N} |F_i|$ is the total height of all fences, and

$$\Phi_4 = 2t \cdot \left(\min\{T(\|F_N\|), 3|F_N|\} + \sum_{i=1}^{N-1} T(\|F_i\|) \right),$$

taken to be zero if no fences exist. Φ_3 and Φ_4 decompose naturally into contributions by the individual fences. The contribution of a fence F_i to Φ_3 is its height $|F_i|$, and the contribution of F_i to Φ_4 is essentially $2t$ times $T(\|F_i\|)$. An exception concerns the leftmost fence F_N. If the height of F_N is less than one third of $T(\|F_N\|)$, the contribution of F_N to Φ_4 instead is $6t$ times its height. If and when column n is rejected, Φ_1 becomes undefined; since this also causes the algorithm to terminate without additional probes, it is of no concern to the proof.

LEMMA 4.15. *Every nonexcluding successful top-row probe increases* Φ *by at least* 1.

Proof. The probe leaves Φ_1 and Φ_2 unchanged. If it does not cause a merge, it increases Φ_3 by 1 and does not decrease Φ_4. If the probe causes a merge, it leaves Φ_3 unchanged and increases Φ_4 by replacing the leftmost fence by a fence of the same height whose weight is greater by 1. Since the leftmost fence was of target height before the probe (Corollary 4.11) and $T'(x) \geq 1/t$ for all $x \geq 1$, the increase in Φ_4 is at least 2. In either case, the net increase in Φ is at least 1. □

LEMMA 4.16. *Every successful extending probe increases* Φ *by at least* 1.

Proof. Even if the probe causes a merge, consider an imagined intermediate situation in which the fence in question has been extended, but no merge has yet taken place. Until this point, the probe leaves Φ_1 and Φ_2 unchanged, increases Φ_3 by 1, and does not decrease Φ_4. Overall, Φ increases by at least 1, and we are done if no merge happens.

In the rest of the proof we assume that a merge happens and prove that it does not decrease Φ. Φ_1 and Φ_2 are not affected by the merge. Let $\mathcal{F} = (F_N, \ldots, F_1)$ immediately before the merge. By Lemma 4.6, the fences that merge are the leftmost ones, i.e., F_N and F_{N-1}. By Corollary 4.11 and Lemma 4.8, the fences that merge as well as the resulting fence are all of height $|F_N| = H(F_{N-1}) \geq T(\|F_{N-1}\|) \geq T(\|F_N\|)$.

The contribution to $\Phi_3 + \Phi_4$ of F_N and F_{N-1} before the merge is at most

$$2|F_N| + 2t(T(\|F_N\|) + T(\|F_{N-1}\|)) \le 6t|F_N|,$$

the contribution to $\Phi_3 + \Phi_4$ of the fence resulting from the merge is

$$|F_N| + 2t \cdot \min\{T(\|F_N\| + \|F_{N-1}\|), 3|F_N|\},$$

and no other fence changes its contribution.

If $3|F_N| < T(\|F_N\| + \|F_{N-1}\|)$, the merge clearly increases $\Phi_3 + \Phi_4$. Otherwise the increase in Φ caused by the merge is at least

$$2t[T(\|F_N\| + \|F_{N-1}\|) - (T(\|F_N\|) + T(\|F_{N-1}\|))] - |F_N|.$$

By Lemmas 4.4 and 4.8, we must have $\|F_{N-1}\| \ge \|F_N\| > \|F_{N-1}\|/a$. We can therefore apply Lemma 4.2 with $\epsilon = 1/a$, which shows that Φ increases by at least

$$\frac{2t}{t}[T(\|F_N\|) + T(\|F_{N-1}\|)] - |F_N|.$$

To see that this is nonnegative, observe that $T(\|F_{N-1}\|) > 1$ (since $|F_{N-1}| \ge 2$) and therefore $|F_N| = \lceil T(\|F_{N-1}\|) \rceil \le 2T(\|F_{N-1}\|)$. □

LEMMA 4.17. *Every nonexcluding unsuccessful probe increases Φ by at least 1.*

Proof. The probe is a top-row probe and leaves $\Phi_2 + \Phi_3$ unchanged. If no fences are present when the probe is performed, at least half of the remaining columns are rejected, causing Φ_1 to increase by at least 1 while Φ_4 remains zero. If one or more fences are present at the time of the probe, the weight of the leftmost fence increases by 1. Since the fence was of target height before the probe, by Corollary 4.11, an argument as in the proof of Lemma 4.15 shows that this increases Φ_4 by at least 2, while Φ_1 does not decrease. □

LEMMA 4.18. *Every excluding probe causes all nonrejected columns to be rejected or increases Φ by at least 1.*

Proof. Let $\mathcal{F} = (F_N, \ldots, F_1)$ immediately before the probe under consideration. Suppose that the probe leads to the exclusion of m rows but not to the rejection of all nonrejected columns. Then the probe does not decrease Φ_1, increases Φ_2 by $13tm$, and, since there are at most t fences (Lemma 4.14), decreases Φ_3 by at most tm.

If no fences are excluded and $N \ge 1$, $|F_N|$ decreases by m and $\|F_i\|$ is unchanged for $i = 1, \ldots, N$, so that Φ_4 decreases by at most $6tm$.

If at least one fence is excluded, let the excluded fences be $F_N, \ldots, F_i$. By Lemmas 4.9 and 4.10, the contribution to Φ_4 of the excluded fences $F_N, \ldots, F_i$ before the probe was at most $4tT(\|F_i\|) \le 8t|F_i|$ if $i < N$ and at most $6t|F_i|$ if $i = N$, so that their exclusion decreases Φ_4 by at most $8t|F_i| \le 8tm$. If $i > 1$, the contribution of F_{i-1} to Φ_4 may also decrease because F_{i-1} becomes the new leftmost fence. Since F_{i-1} was of target height before the probe under consideration, however (Corollary 4.11), this happens only if F_{i-1} loses more than two thirds of its height, in which case its contribution to Φ_4 before the probe was $2tT(\|F_{i-1}\|) \le 2t|F_{i-1}| \le 3tm$. Altogether, therefore, Φ_4 decreases by at most $8tm + 3tm = 11tm$.

In either case Φ_4 decreases by at most $11tm$, yielding a net increase in Φ of at least $13tm - tm - 11tm = tm \ge 1$. □

LEMMA 4.19. *As long as at least one column has not been rejected, $\Phi = O(kt + \log n)$.*

Proof. As long as at least one column has not been rejected, $\Phi_1 \leq \log n$, and $\Phi_2 \leq 13kt$. By Corollary 4.13, the weight of every fence remains bounded by $2 \log n$. Coupled with the facts that $\sum_{i=1}^{N} T(\|F_i\|) \leq 2T(\|F_1\|)$ if $N \geq 1$ (Lemma 4.9), that $T(2 \log n) = O(k + \log n / \log \log n)$ (Lemma 4.1), and that $t = O(\log \log n)$, this shows that $\Phi_4 = O(kt + \log n)$. In order to complete the proof by demonstrating that $\Phi_3 = O(kt + \log n)$, it suffices, since there are at most t fences (Lemma 4.14), to show that no fence is ever of height more than $\lceil T(2 \log n) \rceil$. Since only the leftmost fence can have a height exceeding its target height (Lemma 4.10), this follows immediately from Corollary 4.13 for all other fences. As for the leftmost fence F, its height is bounded by 1 or by the height of a nonleftmost fence (that may disappear at the creation of F) when F is created and whenever it is not the only fence. We finally observe that the algorithm never extends a fence whose height is no smaller than its target height if it is the only fence, and we conclude that even the leftmost fence can never acquire a height of more than $\lceil T(2 \log n) \rceil$. □

LEMMA 4.20. *The total number of probes performed by the algorithm is* $O(kt + \log n)$.

Proof. Since $\Phi = 0$ initially, the claim follows immediately from Lemmas 4.15–4.19. □

We can conclude the following theorem.

THEOREM 4.21. *For all integers* $k \geq 1$ *and* $n \geq 4$, *leftmost-all-*1 *problems of size* $k \times n$ *can be solved by a fence algorithm using*

$$O\left(\frac{k \log \log n}{\log \log (4 + \frac{k \log \log n}{\log n})} + k + \log n\right)$$

probes in the worst case.

4.4. String ranking. In this subsection we extend the algorithm for the leftmost-all-1 problem to solve the original string-ranking problem using at most twice as many probes. The upper bound of our main result, Theorem 1.1, follows from Theorem 4.21 and Lemma 4.22 below.

LEMMA 4.22. *For all integers* $k, n, m \geq 1$, *if there is a surface algorithm that solves instances of size* $k \times n$ *of the leftmost-all-*1 *problem using at most* m *probes, then there is an algorithm that solves instances of size* $k \times n$ *of the string-ranking problem using at most* $2m$ *probes.*

Proof. Let I and s be the input matrix and the query string, respectively, and denote by s_i the ith character of s for $i = 1, \ldots, k$. We derive a $k \times n$ matrix I' from I as follows. First, for each column of I that coincides with s, the corresponding column of I' contains 1's in every position. Now assume that $1 \leq c \leq n$ and that column c of I differs from s in at least one position, and let r be the smallest row index with $I[r, c] \neq s_r$. Then we set $I'[r, c] = 0$ if $I[r, c] < s_r$ and $I'[r, c] = 2$ if $I[r, c] > s_r$, and all other entries in column c of I' are set to 1. It can be seen that I' is sorted and that the task is to compute the number of columns in I' that do not contain a 2.

With the understanding that each occurrence of 2 is to be considered equivalent to a 1, a convention that again preserves sortedness, we can run a process $\mathcal{A}_{\mathrm{L}}$ that executes the given surface algorithm for the leftmost-all-1 problem on the input I'. This computes the number of columns containing a 0, which is not what we want. On the other hand, we can instead run a "mirrored" process $\mathcal{A}_{\mathrm{R}}$ that also executes the given surface algorithm but interchanges the roles of left and right, $<$ and $>$, and 0 and 2. (In particular, $\mathcal{A}_{\mathrm{R}}$ considers 0 to be equivalent to 1.) This process will indeed

compute the number of columns that do not contain a 2 or, rather, n minus that number.

There is a catch, however, namely, that it is not clear how to produce the input I' before starting $\mathcal{A}_{\mathrm{R}}$ without using too many probes. Instead, $\mathcal{A}_{\mathrm{R}}$ must be able to convert the outcome of each of its probes in I to the corresponding entry in I' without performing additional probes. This takes a little care and requires us to execute both $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ in an interleaved fashion.

Let us consider the situation from the perspective of $\mathcal{A}_{\mathrm{L}}$. Since $\mathcal{A}_{\mathrm{L}}$ is a surface algorithm, initially it has no difficulty converting the entries read in I to the corresponding entries in I', the reason being that in each column, it probes from the top towards the bottom. When about to exclude one or more rows, however, $\mathcal{A}_{\mathrm{L}}$ runs into a problem. Outside of the rejected columns, the rows of I' to be excluded are known to contain only characters that $\mathcal{A}_{\mathrm{L}}$ considers to be equivalent to 1, namely, 1's and 2's, and $\mathcal{A}_{\mathrm{L}}$ will assume that all such characters are in fact 1's. However, any occurrence of a 2 changes the interpretation of a 0 that may later be discovered further down in the same column, and therefore $\mathcal{A}_{\mathrm{L}}$ may later convert entries of I incorrectly to ones of I' if it excludes a row containing 2's. Whenever $\mathcal{A}_{\mathrm{L}}$ is about to exclude a row, it therefore needs help. In complete symmetry, $\mathcal{A}_{\mathrm{R}}$ can run until it is about to exclude one or more rows, at which point, since the rows of I' to be excluded might contain 0's, $\mathcal{A}_{\mathrm{R}}$ needs help.

Now consider a situation in which both $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are blocked and waiting for help, and let r_{L} and r_{R} be the indices of the topmost rows about to be excluded by $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$, respectively. Also let z_{L} and z_{R} be the indices of the 0-barriers of $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$, respectively. Assume inductively that up to the present point, neither of the two processes has ever excluded a row of I' with a position containing an entry different from 1 and rejected by neither $\mathcal{A}_{\mathrm{L}}$ nor $\mathcal{A}_{\mathrm{R}}$. Also assume that $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are modified so that whenever one of the processes wants to query a position that has been rejected by the other process, it receives a 1 as the answer to its query without consulting I. By the inductive hypothesis, the column of I' of index z_{L} is known to contain a string smaller than 1^k and therefore must be to the left of the column of I' of index z_{R}, which is known to contain a string larger than 1^k. In other words, $z_{\mathrm{L}} < z_{\mathrm{R}}$. Moreover, for all c with $z_{\mathrm{L}} < c < z_{\mathrm{R}}$, every position of the form $I[\min\{r_{\mathrm{L}}, r_{\mathrm{R}}\}, c]$ is known to contain an entry that is both ≥ 1 and ≤ 1 and thus equals 1. It follows that if $r_{\mathrm{L}} \leq r_{\mathrm{R}}$, then $\mathcal{A}_{\mathrm{L}}$ can proceed and exclude row r_{L} without falsifying the inductive property, while if $r_{\mathrm{R}} \leq r_{\mathrm{L}}$, $\mathcal{A}_{\mathrm{R}}$ can resume operation.

Thus $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are never simultaneously blocked. Moreover, once one of the processes terminates, the other process can finish without being suspended again. Since the two processes $\mathcal{A}_{\mathrm{L}}$ and $\mathcal{A}_{\mathrm{R}}$ are copies of the given surface algorithm, except that sometimes they wait and that some answers are given to them for free, the total number of probes performed is bounded by $2m$. □

Although the upper bounds of Theorems 1.1 and 4.21 specify only the number of probes performed, we note that the algorithms realizing the upper bounds can be executed in a total time that is within the bound on the number of probes, each probe being followed by exactly one (three-way) comparison between two characters. The only nontrivial observation needed is that during an execution of the algorithm described in section 4.1, whenever a fence that is not the leftmost fence is of target height, it remains of target height until the next row exclusion, at which point we can afford to step through a list of all fences.

5. The lower bound. The aim of this section is to prove the following theorem, which implies the lower-bound part of Theorem 1.1.

THEOREM 5.1. *For all integers $k \geq 1$ and $n \geq 4$, every deterministic algorithm for the string-membership problem or the leftmost-all-1 problem performs*

$$\Omega\left(\frac{k \log\log n}{\log\log(4 + \frac{k \log\log n}{\log n})} + k + \log n\right)$$

probes on some input of size $k \times n$.

We prove Theorem 5.1 by exhibiting an adversary that forces every deterministic algorithm $\mathcal{A}$ for either problem to spend as many probes as stated in the theorem before announcing its answer. In the case of the string-membership problem, the lower bound is proved for a special case: the alphabet Σ and the string s whose presence is to be tested are fixed to be $\{0, 1\}$ and $1^{k-1}0$, respectively.

The adversary fixes entries of a legal input I online in response to the queries made by $\mathcal{A}$. Whenever $\mathcal{A}$ poses a query (r, c), i.e., asks for the value of $I[r, c]$, the adversary executes a call $\mathsf{Process}(r, c)$, where $\mathsf{Process}$ is described in the next subsection, that fixes zero or more entries of I. Subsequently, $I[r, c]$ will have been fixed to either 0 or 1, and the value to which it was fixed is returned to $\mathcal{A}$ as the answer to its query.

We formally define a *position* to be an element of $\{1, \ldots, k\} \times \{1, \ldots, n\}$. The adversary maintains information about the part of I already fixed in a $k \times n$ array J with entries drawn from $\{0, 1,$ "tentative-1," "?"$\}$ and a set P of *pending positions*. For $r = 1, \ldots, k$ and $c = 1, \ldots, n$, by definition, if $(r, c) \in P$, then $I[r, c]$ has been fixed to 1; if $(r, c) \notin P$, then $I[r, c]$ has been fixed to the value $b \in \{0, 1\}$ exactly if $J[r, c] = b$.

Although the adversary is not a fence algorithm probing the input I, it will be very convenient for the proof to reuse the terminology introduced for fence algorithms in section 2.2. In order to make this possible, it suffices to define the rejected positions and the matching area, since all other relevant concepts (fences, surface positions, the 0-barrier, etc.) were derived from these basic notions. But this is easy: a position (r, c) is rejected exactly if $J[i, j] = 0$ for some position (i, j) with $j \geq c$, and a nonrejected position (r, c) belongs to the matching area exactly if $J[r, c] \in \{1,$ "tentative-1"$\}$. The adversary will carry out explicit 1-extension (or, rather, "tentative-1-extension") to ensure that the matching area remains monotonic. Thus entries of 0 in J correspond to probes answered 0, and entries of 1 or "tentative-1" correspond to probes answered 1. The difference between 1 and "tentative-1" is that a 1 is permanent, as is a 0, while a "tentative-1" may later be changed to 0 or 1. If $J[r, c] =$ "?" for some position (r, c), the adversary has not yet decided upon the value of $I[r, c]$ (unless $(r, c) \in P$, in which case $I[r, c] = 1$).

To a first approximation, the adversary fixes only those entries of I that were queried by $\mathcal{A}$ or whose values can be deduced from such entries by 1-extension. In order to simplify the book-keeping, however, we let the adversary sometimes fix additional entries of I. Informally, this allows us to assume that $\mathcal{A}$ operates largely as the algorithm analyzed in the previous section. The lower bound holds even if the additional information about I volunteered by the adversary in this manner is made known to $\mathcal{A}$. It is therefore not necessary to distinguish between the information available to $\mathcal{A}$ and that available to the adversary—this is obvious anyway, since the adversary operates according to a fixed, deterministic strategy.

5.1. The adversary's strategy. The adversary's strategy is formulated in terms of a number of parameters that we introduce next. It will be convenient to

use the natural logarithm function "ln" to base e instead of the logarithm function to base 2 employed in the previous section. First, let

$$a = \ln\left(\frac{k \ln\ln n}{\ln n}\right) \qquad \text{and} \qquad v = 3a + 1.$$

For $k = O(\log n/\log\log n)$, the bound of Theorem 5.1 reduces to a trivial bound of $\Omega(k + \log n)$. This allows us to assume a to be larger than any convenient constant. In particular, we will assume that $a \geq 4$ and hence

$$v \leq a^2. \tag{5.1}$$

Next, we take

$$t = \left\lfloor \frac{\ln\ln n}{8 \ln a} \right\rfloor.$$

Similarly as before, the bound of Theorem 5.1 reduces to the trivial bound of $\Omega(k + \log n)$ for $t = O(1)$ and hence for $a = (\log n)^{\Omega(1)}$, for which reason we will assume that $t \geq 2$ and that the following relations hold:

$$e^{1/(3t)} \leq 1 + 1/(2t), \tag{5.2}$$

$$4at \leq \sqrt{\ln n}, \tag{5.3}$$

$$kt + 1 \leq n^{1/4}. \tag{5.4}$$

The parameters a and t have essentially the same meaning as in the proof of the upper bound. In particular, the goal of the adversary is to force $\mathcal{A}$ to spend $\Omega(t)$ probes per row. We associate with each fence F (as implied by J) a weight, $\|F\|$, which is maintained, with one exception, as in the proof of the upper bound. Every new fence created from scratch has weight 1, and when two fences F and F' merge to form a new fence, the new fence is given weight $\|F\| + \|F'\|$. The difference to the proof of the upper bound is that these two rules are the only ones that govern the weights of fences. In particular, the rejection of a number of columns does not change the weight of any surviving fence.

For each integer i and all $x \geq 0$, take

$$\phi_i(x) = \frac{x}{t}\left(\frac{x}{v^{t-i}}\right)^{a/t} \ln\left(\frac{x}{v^{t-i}} + e\right) \geq 0,$$

and note for later use that the derivative

$$\phi_i'(x) = \frac{1}{t}\left(\frac{x}{v^{t-i}}\right)^{a/t}\left[\left(1 + \frac{a}{t}\right)\ln\left(\frac{x}{v^{t-i}} + e\right) + \frac{1}{1 + e \cdot v^{t-i}/x}\right]$$

of ϕ_i is a strictly increasing function. Recall that the *index* of a fence F is one more than the number of fences to the right of F. We define the *value* (to $\mathcal{A}$) of a fence F of index i as

$$\Phi(F) = |F| + \|F\| + \phi_i(\|F\|).$$

The fence will be called *dense* if $\Phi(F) \geq t|F|$. A fence is *critical* if it is dense or if its index is t. The following technical lemma is needed later.

LEMMA 5.2. *If a fence F of index i is not dense, then*

$$|F| \geq \frac{\|F\|}{at} \cdot z^{a/t},$$

where $z = \|F\|/v^{t-i}$.

Proof. Since F is not dense,

$$|F| \geq \frac{\Phi(F)}{t} \geq \frac{\|F\|}{t} + \frac{\|F\|}{t^2} \cdot z^{a/t} \ln(z+e).$$

If $z \leq a^{t/a}$, then

$$|F| \geq \frac{\|F\|}{t} \geq \frac{\|F\|}{t} \cdot \frac{z^{a/t}}{a},$$

as desired. If $z > a^{t/a} \geq e^{t/a}$, on the other hand, then

$$|F| \geq \frac{\|F\|}{t^2} \cdot z^{a/t}(t/a) = \frac{\|F\|}{at} \cdot z^{a/t}. \qquad \square$$

The adversary exercises tight control over the horizontal placement of fences. As an aid in describing this mechanism, we introduce a set L of "legal fence-column indices" and a corresponding set of "legal fence columns." Suppose that $\mathcal{F} = (F_N, \ldots, F_1)$, that F_i is in column c_i for $i = 1, \ldots, N$, and that the 0-barrier is in column c_Z. Then $L = \{c^*, c_N, c_{N-1}, \ldots, c_1\}$, where $c^* = c_Z + \max\{\lfloor e^{-a} \cdot (c_N - c_Z) \rfloor, 1\}$, with c_N taken to be $n+1$ if $N = 0$. A legal fence column is a column whose index belongs to L. Thus every column that already contains a fence is a legal fence column, and there is one more legal fence column to the left of the leftmost fence, about e^{-a} of the way from the 0-barrier to that fence. The adversary rejects the column of every probe to the left of the leftmost legal fence column and translates every other probe to the nearest legal fence column to the left of or in the column of the position probed and in this way allows fences to grow only in legal fence columns. Note that since $e^{-a} \leq 1/2$, this implies that no two fences will ever reside in adjacent columns.

We now describe the strategy of the adversary precisely by giving the procedure Process and two subroutines PutZero and OneProbe that it employs. Before the first call of Process, every entry of J is initialized to "?," and the set P of pending positions is set to $\emptyset$. For the sake of a succinct description of PutZero, we take $\min \emptyset$ to be ∞, i.e., distinct from every integer.

PutZero(c):
  for $(i,j) \in \{1, \ldots, k\} \times \{1, \ldots, c\}$ **do**
    $J[i,j] := \begin{cases} 0, & \text{if } i = \min\{\ell \mid 1 \leq \ell \leq k \text{ and } J[\ell, j] \neq 1\}, \\ 1, & \text{otherwise.} \end{cases}$

OneProbe(r, c):
  if $c < \min L$ **then** PutZero(c) **else**
    $c' := \max\{j \in L \mid j \leq c\}$;
    for $j \in \{c', \ldots, n\}$ **do if** $J[r,j] =$ "?" **then** $J[r,j] :=$ "tentative-1";
    if column c' contains a critical fence **then** PutZero(c').

Process(r, c):
  if (r,c) is buried **then** insert (r,c) in P and return 1 **else**

if $J[r, c] =$ "?" **then**
  OneProbe(r, c);
  while P contains a surface position **do**
    Let (r', c') be a surface position in P;
    OneProbe(r', c');
    Remove from P all positions $(i, j) \in P$ with $J[i, j] \neq$ "?";
  for $(i, j) \in \{1, \ldots, k\} \times \{1, \ldots, n\}$ **do**
    if $J[i, j] =$ "tentative-1" **then** $J[i, j] := 1$;
Return $J[r, c]$.

We illustrate the adversary's strategy through an extensive example worked in Figure 5.1. The symbols "0," "1," and "T" denote positions in J containing the values 0, 1 and "tentative-1," respectively. Each "P" denotes a position $(i, j) \in P$ with $J[i, j] =$ "?," while other occurrences of "?" in J are not shown explicitly. The matching area is separated from the remaining positions by a staircase line, and surface positions are shown shaded.

Assume that Process$(6, 7)$ is called when the situation is as shown in (a). Since $(6, 7)$ is a surface position, a call OneProbe$(6, 7)$ is executed. The argument $(6, 7)$ of OneProbe is indicated by a circle in (a), and the effect of the call OneProbe$(6, 7)$ is shown in (b). The position $(6, 7)$ is "moved" left until it hits a legal fence column, the fence F in that column is extended by one position containing the value "tentative-1," and "tentative-1-extension" is carried out from the new fence position. The latter causes several formerly buried positions in P to become part of the surface, and the call of Process proceeds to execute OneProbe(r', c') for one such position (r', c'). This second extension of F, the result of which is shown in (c), causes F to merge with its right neighbor. The transition from (c) to (d) gives rise to yet another merge. We assume that the fence in column 6 resulting from the merge is dense, so that PutZero(6) is executed. The outcome is shown in (e): four columns were rejected, and six rows were excluded. Each excluded column contains a 0 in the position that belonged to the surface in situation (a) and 1's in all other positions. In situation (e), the surface still contains elements of P, so another call of OneProbe is executed. Let us assume that the argument of this call is the leftmost eligible surface position, $(9, 8)$, and that the condition $c < \min L$ is satisfied. Then the call PutZero(8) leads to the situation in (f). The effect of three more calls of OneProbe, none of which is assumed to call PutZero, is shown in (g). Since the surface in (g) contains no elements of P, no further calls of OneProbe are initiated. The call Process$(6, 7)$ finally converts all occurrences of "tentative-1" to 1—we call this step, shown in the transition from (g) to (h), the *consolidation*—and returns the value of $J[6, 7]$, which is 0.

5.2. Properties of the strategy. We first show in a series of lemmas that the answers provided by the adversary are consistent with a fixed input I, by which we mean that each query (r, c) is answered by $I[r, c]$. We also argue (Lemma 5.8) that I can be chosen to be sorted, i.e., as a legal input to the string-membership and leftmost-all-1 problems.

LEMMA 5.3. *For all* $(r, c) \in \{1, \ldots, k\} \times \{1, \ldots, n\}$*, the value of* $J[r, c]$ *can change only according to the transitions indicated in Figure* 5.2*. Moreover, no occurrences of "tentative-*1*" are present in* J *at the start or end of a call of* Process*, and no call of* Process *returns the value "tentative-*1*."*

Proof. The claim concerning "tentative-1" is a consequence of the consolidation. It is clear from a simple inspection of the code that, following the initialization, no "?" is stored in J and that no occurrence of 1 is replaced by a different value. Moreover, an

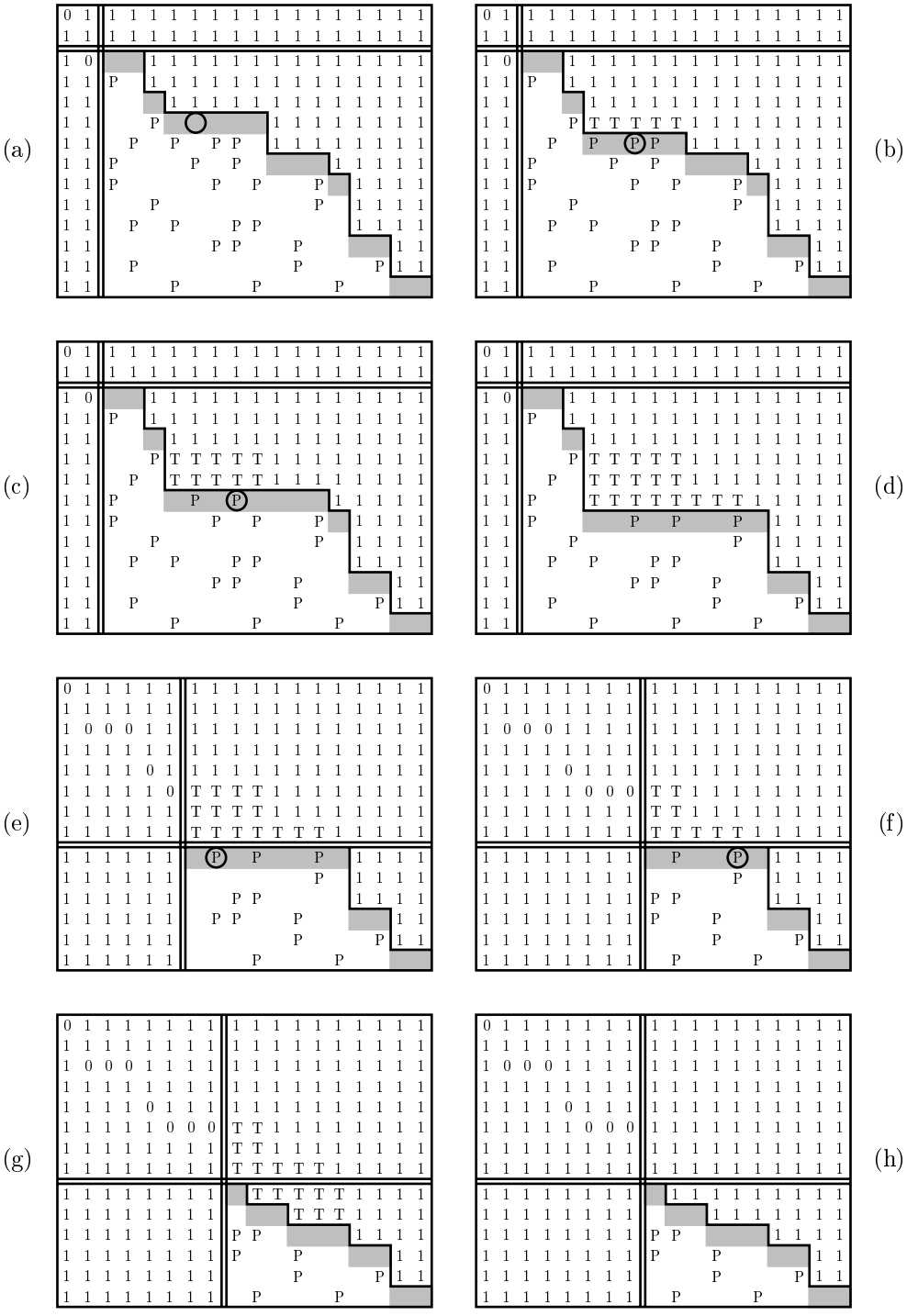


FIG. 5.1. *An example execution of* Process.

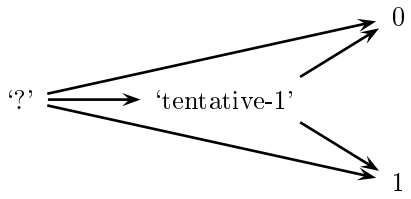


FIG. 5.2. *The possible transitions of an entry in J.*

occurrence of 0 could be changed only in a call of PutZero. What remains, therefore, is to show that no 0 in a column rejected in a call of PutZero is changed by a subsequent call of PutZero. But this is obvious. □

LEMMA 5.4. *Every call* Process(r, c) *returns a value in* $\{0, 1\}$.

Proof. By Lemma 5.3, the claim is obvious if (r, c) is buried or $J[r, c] \neq$ "?" at the start of the call. But otherwise, a call OneProbe(r, c) is executed, which can easily be seen to store a value different from "?" in $J[r, c]$. The lemma now follows by a second appeal to Lemma 5.3. □

LEMMA 5.5. *All positions whose entries are set to* 0 *during a call of* Process *were surface positions at the start of the call of* Process.

Proof. The only occasions on which a 1 is stored in a position in J are when the column containing the position is rejected and during the final consolidation. By this observation and Lemma 5.3, immediately before a call of Process causes a column to be rejected, the topmost position in the column that contains a value different from 1 is the same as it was at the start of the call of Process, at which time it was a surface position. □

LEMMA 5.6. *The matching area at all times is monotonic.*

Proof. The matching area initially is empty and hence monotonic. It is changed through only two types of operations: the removal of a number of leftmost (rejected) columns and the inclusion of a surface position and all positions to its right that do not already belong to the matching area. Both operations preserve monotonicity. □

LEMMA 5.7. *No column of J is ever rejected while it contains the string* 1^k.

Proof. Suppose that $1 \leq c \leq n$ and that column c of J is rejected while containing 1^k. Then column c must contain 1^k already at the start of the call of Process that rejects it. At that time, by Lemmas 5.3 and 5.6, all columns to the right of column c also contain 1^k, in which case column c cannot be rejected. □

LEMMA 5.8. *The answers provided by the adversary are consistent with a sorted input.*

Proof. Let I be the input obtained from the final value of J by changing to 1 all entries that are not 0. We prove that each query (r, c) is answered by $I[r, c]$ and that I is sorted.

A query (r, c) is answered by 0 only if $J[r, c] = 0$ at the time of the answer and thus, by Lemma 5.3, only if $I[r, c] = 0$. The same argument applies to answers of 1, except that, because of the probes of buried positions, we must additionally show that no entry in J of a position that belongs to P at some time is ever set to 0. Assume, to the contrary, that such an entry is set to 0 in some call of Process. By Lemma 5.3, the value of the entry must have been "?" at the start of the call of Process; i.e., the corresponding position still belonged to P at that time. But given that P contained

no surface positions at that time, due to the termination condition of the **while** loop in Process, this contradicts Lemma 5.5.

We now show that I is sorted and begin by observing that no column of I contains more than one 0. Let $1 \le c < c' \le n$ and $1 \le r' \le k$ and suppose that $I[r', c'] = 0$. By Lemmas 5.3 and 5.7, we must have $I[r, c] = 0$ for some $r \in \{1, \ldots, k\}$. In order to complete the demonstration that I is sorted, it suffices to show that $r \le r'$.

By Lemma 5.5, (r, c) belongs to the surface at some time τ, and (r', c') belongs to the surface at the same or some later time (since column c' is rejected simultaneously with or later than column c). If $(\bar{r}, c')$ is the surface position in column c' at time τ, we have $r \le \bar{r}$ by the monotonicity of the matching area at time τ (Lemma 5.6) and $\bar{r} \le r'$ because, as long as a column contains a surface position, the row index of its surface position never decreases. (This is a consequence of Lemma 5.3.) Thus indeed $r \le r'$. □

LEMMA 5.9. *When $\mathcal{A}$ terminates, either all rows except at most one have been excluded, or all entries in the last row outside of the rejected columns have been fixed to* 1.

Proof. Assume that $\mathcal{A}$ terminates while the entry of some position (k, c) in the last row is still unfixed. We complete the tableau J to two inputs I_0 and I_1: I_1 is obtained simply by fixing all remaining unfixed entries of I to 1. I_0 is obtained as follows. If $c > 1$, first PutZero$(c-1)$ is executed. Then the entry in position (k, c) is fixed to 0, and all remaining unfixed entries of I are fixed to 1.

The inputs I_0 and I_1 are both consistent with all answers obtained by $\mathcal{A}$. It was already argued in the preceding proof that I_1 is sorted, and, using essentially the same argument, it can be seen that I_0 is sorted as well, so that I_0 and I_1 are both legal inputs. Moreover, column c of I_0 contains the string $1^{k-1}0$, column c of I_1 contains 1^k, and if two or more rows had not been excluded when $\mathcal{A}$ terminated, no column of I_1 contains $1^{k-1}0$. But in that case, whether $\mathcal{A}$ is an algorithm for the string-membership problem with query string $1^{k-1}0$ or for the leftmost-all-1 problem, I_0 and I_1 are not associated with a common correct output. Thus $\mathcal{A}$ cannot produce its answer, which is a contradiction. □

Recall that the *gap* of a fence is its distance from the 0-barrier and that its *cumulative weight* is the sum of its own weight and the weights of all fences to its right. We define the *bias* of a fence F of gap g and cumulative weight w as the quantity

$$Bias(F) = \frac{\ln(n+1) - \ln g - 2aw}{-\ln(1 - e^{-a})}.$$

We apply this definition even to an imaginary fence F_0 in column $n+1$ and of cumulative weight 0 and define the *maximum bias* B as $\max_{i=0}^{N} Bias(F_i)$, where $\mathcal{F} = (F_N, \ldots, F_1)$.

LEMMA 5.10. *$B = 0$ initially, $B \ge 0$ always, and a call of* PutZero *or* OneProbe *increases B by at most* 1.

Proof. The first two claims are obvious, the second one because $Bias(F_0) \ge 0$. It is not difficult to see from the definition of the set of legal fence-column indices that after the rejection of one or more columns, the gap of every surviving fence is at least $1 - e^{-a}$ times what it was before the rejection—every potential new 0-barrier is at most a fraction of e^{-a} of the way from the current 0-barrier to the fence. A call of PutZero or OneProbe that causes columns to be rejected therefore increases B by at most 1.

If a new fence F is created from scratch at a time when the previous leftmost fence F' (F_0 if there are no fences) has gap g, we have $e^{-a}g \geq 2$, so that the gap of F is at least $e^{-a}g - 1 \geq e^{-2a}g$. Since the cumulative weight of F is one more than that of F', we will have $Bias(F) \leq Bias(F')$. Thus a call of OneProbe that does not cause columns to be rejected also does not increase B. □

Note that since a critical fence is excluded immediately after its creation, the number of fences never exceeds t.

5.3. Analysis of the number of probes. The number of probes needed is bounded from below using the potential function

$$\Phi = \Phi_1 + \Phi_2 + \Phi_3 + \Phi_4,$$

where $\Phi_1 = \sum_{F \in \mathcal{F}} \Phi(F)$ is the total value of all fences, $\Phi_2 = tX_{\mathrm{R}}$ is t times the number of excluded rows, $\Phi_3 = 4|P|$ is four times the number of pending positions, and $\Phi_4 = B$ is the maximum bias.

LEMMA 5.11. *If $\Phi \leq kt$, then the gap of every fence, including the imaginary fence F_0, is greater than $n^{1/4}$.*

Proof. Assume that $\Phi \leq kt$ and that the gap g of some fence F (possibly F_0) is at most $n^{1/4}$, and let w be the cumulative weight of F. By the mean-value theorem, $\ln(1-x) \geq -2x$ for $0 \leq x \leq 1/2$. Using this with $x = e^{-a} = \ln n/(k \ln\ln n)$ and assuming that $Bias(F) \geq 0$, we obtain

$$\begin{aligned} Bias(F) &= \frac{\ln(n+1) - \ln g - 2aw}{-\ln(1-e^{-a})} \\ &\geq (\ln(n+1) - \ln g - 2aw) \cdot \frac{k \ln\ln n}{2 \ln n} \geq \left(\frac{3}{4} - \frac{2aw}{\ln n}\right) \cdot 4kt. \end{aligned}$$

Since $Bias(F) \leq kt$, we must have $2aw/\ln n \geq 1/2$. This relation holds also if $Bias(F) < 0$. In either case, (5.3) therefore implies that $w \geq t\sqrt{\ln n}$. In particular, $F \neq F_0$. Let $\mathcal{F} = (F_N, \ldots, F_1)$. Then $1 \leq N \leq t$, and, therefore, by (5.1),

$$\frac{w/N}{v^{t-1}} \geq \frac{w/N}{a^{2t}} \geq e^{\ln(w/t) - 2t\ln a} \geq e^{\ln(\sqrt{\ln n}) - (\ln\ln n)/4} = e^{(\ln\ln n)/4}.$$

Now, by the convexity of ϕ_1,

$$\begin{aligned} \Phi &\geq \sum_{i=1}^{N} \phi_i(\|F_i\|) \geq \sum_{i=1}^{N} \phi_1(\|F_i\|) \geq N\phi_1\left(\frac{1}{N}\sum_{i=1}^{N}\|F_i\|\right) \\ &\geq N\phi_1(w/N) \geq \frac{w}{t}\left(\frac{w/N}{v^{t-1}}\right)^{a/t} \ln\left(\frac{w/N}{v^{t-1}}\right) \\ &\geq \frac{\ln n}{4at} \cdot e^{(\ln\ln n)/4 \cdot a/t} \cdot \frac{\ln\ln n}{4} \geq \frac{\ln n}{2a} \cdot e^{2a\ln a} \cdot \ln a \geq e^a \ln n = k \ln\ln n > kt, \end{aligned}$$

which is a contradiction. □

LEMMA 5.12. *During the execution of $\mathcal{A}$, Φ increases by at least $(k-1)t$.*

Proof. $\Phi = 0$ initially, so let us consider the situation when $\mathcal{A}$ terminates and show that $\Phi \geq (k-1)t$. This is obvious if all except at most one row have been excluded, so assume that this is not the case and that $\Phi \leq kt$. Let g be the gap of an arbitrary fence, or of F_0 if no fence exists. By Lemma 5.9, we have $|P| \geq g - 1$ and hence, by (5.4), $g \leq kt + 1 \leq n^{1/4}$. Lemma 5.11 now shows that $\Phi > kt$, which is a contradiction. □

LEMMA 5.13. *Every call of* PutZero *increases* Φ *by at most* 1.

Proof. The call increases neither Φ_1 nor Φ_3. By Lemma 5.10, Φ_4 increases by at most 1. If the call causes the exclusion of $m > 0$ rows, some fence of height m was critical—here we use the fact that no two fences are ever in adjacent columns—so the corresponding increase in Φ_2 of tm is compensated by a decrease in Φ_1 of at least tm caused by the exclusion of a dense fence of height m or by a reduction by m in the height of each of t fences. □

LEMMA 5.14. *If* $\Phi \le kt$ *before a call of* OneProbe*, the call increases* Φ *by at most* 4.

Proof. Unless a call of OneProbe simply executes a call of PutZero, which increases Φ by at most 1 according to the previous lemma, it begins by extending a fence F' by one position or creating a new fence F' from scratch. Extending an existing fence by one position increases its value by 1, and the value of a fence of height and weight 1 is bounded by 3; we here use the fact that there are never more than t fences, so that we never employ ϕ_i for $i > t$. Until this point, therefore, Φ has increased by at most 3.

Subsequently to the operation on the fence F', it may disappear through exclusion because it is in the column next to the 0-barrier, as part of the execution of a call of PutZero, or through merging with its right neighbor. The first case is ruled out by Lemma 5.11 in conjunction with (5.3), which shows that $e^{-a} \cdot n^{1/4} \ge n^{1/4-1/(4t)} \ge n^{1/8} \ge 2$, where the last inequality follows from the assumption $t \ge 2$. The second case increases Φ by another at most 1, according to Lemma 5.13, for a total increase in Φ of at most 4. What remains, therefore, is to assume that F' merges with its right neighbor F and to show that this does not increase Φ.

Let the indices of F and F' be i and $i+1$, respectively. The new fence resulting from the merge has index i, and the remaining fences are affected by the merge only insofar as some of them decrease their index by 1; since this decreases Φ, we need not account for it here. What is to be shown, hence, is that the value of the new fence resulting from the merge is no larger than the combined value of the two fences from which it is formed.

All three fences of interest have the same height $|F|$. Let us write $\|F'\| = u\|F\|$, where $u > 0$ is a suitable real number. Then the weight of the new fence is $(1+u)\|F\|$, so that the relation to be shown is

$$\begin{aligned}(|F| + \|F\| + \phi_i(\|F\|)) + (|F| + u\|F\| + \phi_{i+1}(u\|F\|)) \\ - (|F| + (1+u)\|F\| + \phi_i((1+u)\|F\|)) \ge 0\end{aligned}$$

or, equivalently,

$$|F| + \phi_i(\|F\|) + \frac{\phi_i(uv\|F\|)}{v} - \phi_i((1+u)\|F\|) \ge 0.$$

The derivative of the left-hand side above with respect to u is

$$\|F\|\phi_i'(uv\|F\|) - \|F\|\phi_i'((1+u)\|F\|).$$

Recall that ϕ_i' is a strictly increasing function. This implies that the original left-hand side has a unique minimum that occurs for the value of u satisfying $uv = 1 + u$, i.e., $u = 1/(v-1)$. It therefore suffices to prove the original claim for this value of u, i.e., to show that

$$|F| + \phi_i(\|F\|) + \frac{1}{v}\phi_i\left(\frac{v}{v-1}\|F\|\right) - \phi_i\left(\frac{v}{v-1}\|F\|\right) \ge 0$$

or, equivalently, that

$$\frac{1}{q}\phi_i(q\|F\|) - \phi_i(\|F\|) \le |F|,$$

where we introduced the abbreviation $q = v/(v-1)$. Note that $q = 1 + 1/(3a)$ and hence

$$q^{a/t} = \left(1 + \frac{1}{3a}\right)^{a/t} \le e^{1/(3t)}.$$

Take $z = \|F_i\|/v^{t-i}$. Then

$$\begin{aligned}
\frac{1}{q}\phi_i(q\|F\|) - \phi_i(\|F\|) &= \frac{\|F\|}{t}(qz)^{a/t}\ln(qz+e) - \frac{\|F\|}{t}z^{a/t}\ln(z+e) \\
&\le \frac{\|F\|}{t}z^{a/t}[q^{a/t}(\ln q + \ln(z+e)) - \ln(z+e)] \\
&= \frac{\|F\|}{t}z^{a/t}[(q^{a/t}-1)\ln(z+e) + q^{a/t}\ln q] \\
&\le \frac{\|F\|}{t}z^{a/t}\left[(e^{1/(3t)}-1)\ln(z+e) + e^{1/(3t)}\frac{1}{3a}\right].
\end{aligned}$$

By (5.2), we can bound the right-hand side above by

$$\frac{\|F\|}{2t^2}\cdot z^{a/t}\ln(z+e) + \frac{\|F\|}{2at}\cdot z^{a/t} = \frac{\phi_i(\|F\|)}{2t} + \frac{\|F\|}{2at}\cdot z^{a/t}.$$

Since critical fences are excluded as soon as they arise, F is not dense at the time of the merge. Thus the first term of the right-hand side above is bounded by $|F|/2$, and Lemma 5.2 shows that the same is true of the second term. This completes the proof of the lemma. □

LEMMA 5.15. *If $\Phi \le kt - 4$ before a call of* Process*, the call increases Φ by at most* 4.

Proof. Consider a call Process(r, c). If (r, c) is buried, the call increases $|P|$ by at most 1 and Φ by at most 4. Otherwise, the call executes a call of OneProbe and, subsequently, zero or more times, executes a call of OneProbe and decreases $|P|$ by at least 1 and hence Φ by at least 4. By a simple induction based on Lemma 5.14, $\Phi \le kt$ at the start of each call of OneProbe, and Φ altogether increases by at most 4. The final consolidation does not affect Φ. □

Theorem 5.1 follows from Lemmas 5.8, 5.12, and 5.15.

6. Conclusions. We have given tight bounds for a fundamental searching problem. The problem is natural and easy to formulate, yet the solution—the bound achieved as well as its proof—is surprisingly complicated.

As mentioned in the introduction, the problem becomes much easier if preprocessing and extra space are allowed. It should be noted that our lower bound imposes no restrictions on the model of computation other than the absence of preprocessing; a search algorithm is allowed to use extra memory and arbitrary data structures during its execution.

Acknowledgment. We thank Jyrki Katajainen for bringing the papers by Hirschberg [3, 4] to our attention.

REFERENCES

[1] A. Andersson, T. Hagerup, J. Håstad, and O. Petersson, *The complexity of searching a sorted array of strings*, in Proceedings of the 26th Annual ACM Symposium on Theory of Computing, Montreal, Quebec, Canada, 1994, pp. 317–325.

[2] A. Andersson, J. Håstad, and O. Petersson, *A tight lower bound for searching a sorted array*, in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 417–426.

[3] D. S. Hirschberg, *A lower worst-case complexity for searching a dictionary*, in Proceedings of the 16th Annual Allerton Conference on Communication, Control, and Computing, University of Illinois, Urbana, IL, 1978, pp. 50–53.

[4] D. S. Hirschberg, *On the complexity of searching a set of vectors*, SIAM J. Comput., 9 (1980), pp. 126–129.

[5] S. R. Kosaraju, *On a multidimensional search problem*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing, Atlanta, GA, 1979, pp. 67–73.

[6] U. Manber and G. Myers, *Suffix arrays: A new method for on-line string searches*, SIAM J. Comput., 22 (1993), pp. 935–948.

[7] K. Mehlhorn, *Data Structures and Algorithms, Vol.* 1: *Sorting and Searching*, Springer-Verlag, Berlin, 1984.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1579–1593

MINIMUM NETWORKS IN UNIFORM ORIENTATION METRICS*

M. BRAZIL†, D. A. THOMAS†, AND J. F. WENG†

Abstract. In this paper we use the variational method to systematically study properties of minimum networks connecting any given set of points (called terminals) in a λ-plane, in which all lines are in λ uniform orientations $i\pi/\lambda$ $(0 \le i < \lambda)$. We prove a number of angle conditions for Steiner minimum λ-trees, which are similar to the ones in the Euclidean case. In particular, we show that there exists a Steiner minimum λ-tree whose minimum angles at Steiner points are $\lfloor 2\lambda/3 \rfloor \pi/\lambda$ and whose maximum angles are $\lceil 2\lambda/3 \rceil \pi/\lambda$. We also investigate the assignment of nonstraight edges and unequal angles in Steiner minimum λ-trees, and we prove that there exists a Steiner minimum λ-tree in which every full component has at most one nonstraight edge. From these properties we are able to devise a number of finite methods for constructing Steiner minimum λ-trees. One of these methods is based on using algorithms for finding graphical Steiner minimum trees, and the other uses a generalization of the method of Melzak for Euclidean Steiner trees.

Key words. VLSI design, Steiner tree, fixed orientations, variational method

AMS subject classifications. 05C05, 90B99, 94C15

PII. S0097539798347190

1. Introduction.

Given λ orientations $i\omega\,(i = 1, 2, \ldots, \lambda)$ in the Euclidean plane, where $\lambda(\ge 2)$ is an integer and $\omega = \pi/\lambda$ is a unit angle, we represent the orientations by the angles with the x-axis of corresponding straight lines. For a given λ, a line or line segment with one of these orientations is said to be in a *legal direction*. Define the distance between two points p and q to be the length of the shortest path joining p and q with all edges in legal directions. This definition of distance induces a metric, called the λ-*metric*, in the Euclidean plane. The plane in which a λ-metric is defined is called λ-*oriented*, or a λ-*plane*. Note that a λ-plane is a Minkowski plane in which the unit disc is a regular 2λ-gon with the x-axis being a diagonal direction. A network (or tree) in the plane, composed of line segments in legal directions, is called a λ-*network* (respectively, λ-*tree*). Such objects are said to belong to a λ-*geometry*.

Let N be a set of n points in a λ-plane. The Steiner problem in a λ-plane asks for a shortest λ-network interconnecting the given points possibly incorporating some additional nodes to shorten the network. The network T, which must be a λ-tree, is called the *Steiner minimum* λ-*tree* for N. The given points are referred to as *terminals*, and the additional points are referred to as *Steiner points*.

The study of Steiner minimum λ-trees is of both theoretical and practical interest. Note that the λ-metric is the rectilinear metric when $\lambda = 2$, the hexagonal metric when $\lambda = 3$, and the Euclidean metric when $\lambda = \infty$ [11]. Hence it represents an important generalization of some of the key areas of research on this problem. The study of the hexagonal metric formed part of the basis of the solution of the Steiner ratio conjecture for the Euclidean Steiner problem [2]. Rectilinear Steiner minimum trees (Steiner minimum λ-trees for $\lambda = 2$) have been employed in the design of printed circuit boards and VLSI chips for a long time. Recent developments in VLSI fabrication

*Received by the editors November 17, 1998; accepted for publication (in revised form) August 24, 2000; published electronically November 28, 2000. This work was supported by a grant from the Australian Research Council.

http://www.siam.org/journals/sicomp/30-5/34719.html

†Department of Electrical and Electronic Engineering, The University of Melbourne, Victoria 3010, Australia (m.brazil@ee.unimelb.edu.au, d.thomas@ee.unimelb.edu.au, j.weng@ms.unimelb.edu.au).

technology are beginning to make it possible to design chips with wires running in more than two directions. In particular, the λ-metric for $\lambda = 4$ is already being introduced into VLSI design. The literature to date on λ-metrics includes the papers [14],[3],[11],[7], and [15].

A Steiner minimum λ-tree is called *full* if all terminals are of degree 1. Moreover, as in the rectilinear Steiner problem, a λ-tree is called *fulsome* if the number of full components is maximized. Hence, for any N there is a Steiner minimum λ-tree that is fulsome. By the topology of a network we mean its graph structure. The topology of a (full) Steiner tree is called a (full) Steiner topology. Unlike Euclidean Steiner trees, not all edges in a Steiner minimum λ-tree are necessarily straight, and not all angles at degree 3 Steiner points must equal 120°. Therefore, a λ-tree is not uniquely determined by its Steiner topology but also depends on a knowledge of the assignments of nonstraight edges and unequal angles.

In this paper we use the variational argument [10] to make a systematic study of the Steiner λ-tree problem. Our methods provide a powerful theoretical framework from which we can easily derive the known basic properties of Steiner minimum λ-trees [11] and prove a number of significant new results. The paper is organized as follows. In section 2 we prove angle conditions for Steiner minimum λ-trees, which are similar to those that apply in the Euclidean case. In particular, we show that there exists a Steiner minimum λ-tree whose minimum angles at Steiner points are $\lfloor 2\lambda/3 \rfloor \pi/\lambda$ and whose maximum angles are $\lceil 2\lambda/3 \rceil \pi/\lambda$. It follows that degree 4 Steiner points exist only for $\lambda = 2$, 3, 4, or 6. In section 3 we investigate the assignment of nonstraight edges and unequal angles in Steiner minimum λ-trees. We prove that there exists a Steiner minimum λ-tree in which every full component has at most one nonstraight edge. Consequently, there exists a Steiner minimum λ-tree such that all its Steiner points are grid points of $GG_{n-2}(N)$, a grid point set defined recursively from the terminal set N. This implies that algorithms for constructing graphical Steiner minimum trees can be applied to Steiner minimum λ-trees. Moreover, we also prove some restrictions on angle assignments in Steiner minimum λ-trees. Finally, in section 4 the Melzak method for Euclidean Steiner trees [9] is generalized to Steiner λ-trees. It follows that a Steiner λ-tree with a specified assignment of nonstraight edges and unequal angles can be constructed in linear time, independently of λ.

2. Angle conditions. The basic elements in a geometry are line segments between points and angles between lines. Let p, q be distinct points in the plane. Let $|pq|$ and $|pq|_\lambda$ denote the length of pq in the Euclidean geometry and λ-geometry, respectively. Recall that $\omega = \pi/\lambda$, the smallest positive angle between line segments in λ-geometry. If the Euclidean line segment pq lies in a legal direction, then pq is a straight edge in λ-geometry and $|pq|_\lambda = |pq|$. Otherwise, there is an infinite number of shortest paths joining p, q in λ-geometry. The union of all these paths forms a parallelogram $prqr'$ whose interior angles at r and r' are $\pi - \omega$ (see Figure 2.1(b)) [14],[11]. In this case the edge pq in λ-geometry is a *nonstraight edge*, which can be represented by either of the two *critical paths* prq and $pr'q$. Therefore, $|pq|_\lambda = |pr| + |rq| = |pr'| + |r'q|$. As in the rectilinear metric, the points r, r' are referred to as *corner points*. The following lemma follows immediately from the above discussion.

LEMMA 2.1. *Any angle in λ-geometry is a multiple of ω. Furthermore, the angle at any corner point is $(\lambda - 1)\omega$.*

Given three distinct points p, q, p', let $\angle qpp'$ and $\angle_\lambda qpp'$ denote the angle at p in Euclidean geometry and λ-geometry, respectively. If both pq and pp' are straight

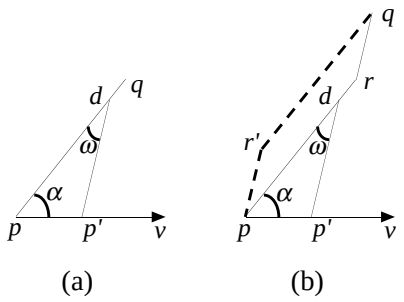


FIG. 2.1. *A one-point variation for a straight edge* (a) *and a nonstraight edge* (b).

edges, then $\angle_\lambda qpp' = \angle qpp'$; otherwise, $\angle_\lambda qpp'$ depends on the positions of the corner points. To most easily accommodate our variational methods, it is convenient to define $\angle_\lambda qpp'$ to be the smallest angle between the shortest paths of pq and pp' (see Figure 2.1(b)).

Let p, q be distinct points in the plane. Suppose q is fixed and p is perturbed to p' in a given legal direction. Let $\alpha = \angle_\lambda qpp'$. By the above definition of angles, if pq is a straight edge, then $\alpha = \angle qpp'$ (Figure 2.1(a)), whereas if pq is a nonstraight edge, then $\alpha = \angle rpp'$, where r is the corner point lying on the same side of the straight line pq as p' (Figure 2.1(b)). Assume the perturbation is sufficiently small that there exists a point d on pq (or pr) such that $\angle p'dp = \omega$. Then dp' is in a legal direction and $|qp'|_\lambda = |qd|_\lambda + |dp'|$. Let $l = |pq|_\lambda, v = |pp'|$. We have

$$\begin{aligned} \dot{l} = \frac{d}{dv}|pq|_\lambda &= \lim_{v\to 0}\frac{|qp'|_\lambda - |qp|_\lambda}{v} = \lim_{v\to 0}\frac{|dp'| - |dp|}{v} \\ &= \lim_{v\to 0}\frac{v\sin\alpha - v\sin(\alpha+\omega)}{v\sin\omega} \\ &= -\frac{\cos(\alpha+\omega/2)}{\cos(\omega/2)}. \end{aligned} \tag{2.1}$$

Remark 2.1. When λ goes to infinity, ω goes to 0 and $\dot{l}$ approaches $-\cos\alpha$, which corresponds to the derivative of l in the Euclidean plane [10].

Remark 2.2. Although we have assumed above that p is perturbed in a legal direction, it is easy to see that formula (2.1) still holds if pp' is not in a legal direction. Note, however, that in that case α is no longer a multiple of ω.

LEMMA 2.2. *Define*

$$f(k) = -\frac{\cos(k+1/2)\omega}{\cos(\omega/2)}.$$

Then, for any fixed λ, f *is an increasing function for all positive* $k \le \lambda - 1/2$, *and*

$$f(m-1) < -1/2, \quad f(2m-1) < 1/2 \text{ if } \lambda = 3m, \tag{2.2}$$

$$f(m-1) + f(m) < -1, \quad f(2m) < 1/2 \text{ if } \lambda = 3m+1, \tag{2.3}$$

$$f(m) < -1/2, \quad f(2m) + f(2m+1) < 1 \text{ if } \lambda = 3m+2. \tag{2.4}$$

Proof. The first statement follows immediately from a consideration of the derivative of f.

The three sets of inequalities have similar proofs to each other. We prove inequalities (2.3) as an example. When $\lambda = 3m+1$,

$$\begin{aligned} f(m-1) + f(m) &= -\frac{\cos((m-1/2)\omega) + \cos((m+1/2)\,\omega)}{\cos(\omega/2)} \\ &= -2\cos(m\omega) = -2\cos\left(\frac{m\pi}{3m+1}\right) < -1, \end{aligned}$$

and

$$f(2m) = -\frac{\cos((2m+1/2)\omega)}{\cos(\omega/2)} < 1/2$$

because $\frac{df(2m)}{dm} > 0$ and $\lim_{m\to\infty} f(2m) = 1/2$. □

It is well known that any angle in a Euclidean Steiner minimum tree is at least $2\pi/3$, and hence there are exactly three angles at each Steiner point, each of which is $2\pi/3$ [6]. We will show that in the λ-plane, there are similar angle conditions. Let $\lambda = 3m+i,\ i = 0, 1, 2$. Note that $2\pi/3 \geq \lfloor 2\lambda/3 \rfloor \omega$.

For a given value of λ, define $\phi_{\min}$ to be the minimum angle that can occur in any Steiner minimum λ-tree.

LEMMA 2.3. *For every (finite) λ, $\phi_{\min} < 2\pi/3$.*

Proof. Since the sum of the angles at a Steiner point of degree 3 is 2π, it immediately follows that $\phi_{\min} \leq 2\pi/3$. It remains to show that $\phi_{\min} \neq 2\pi/3$ when $\lambda = 3m$. Let T be the unique Steiner minimum λ-tree (with $\lambda = 3m$) for the vertices of an equilateral triangle with horizontal axis. This tree contains three straight edges meeting at a single Steiner point. By perturbing one of the terminals at right angles to the incident edge, the resulting Steiner minimum λ-tree will, by continuity, contain a nonstraight edge incident to the Steiner point. One of the choices of a critical path for this nonstraight edge must result in an angle at the Steiner point being strictly less than $2\pi/3$. □

THEOREM 2.4. 1. *Let $\lambda = 3m+i$. Then*

$$\phi_{\min} = \begin{cases} (\lfloor 2\lambda/3 \rfloor - 1)\omega & = (2m-1)\omega & \text{if } i = 0, \\ \lfloor 2\lambda/3 \rfloor \omega & = (2m)\omega & \text{if } i = 1, \\ \lfloor 2\lambda/3 \rfloor \omega & = (2m+1)\omega & \text{if } i = 2. \end{cases}$$

2. *If $\lambda = 3m$, then for any terminal set N there exists a Steiner minimum λ-tree T' such that the minimum angle in T' is at least $\lfloor 2\lambda/3 \rfloor \omega = 2\pi/3$.*

Proof. Let T be a Steiner minimum λ-tree, and let $\angle qpr$ be an angle between two straight line segments in a Steiner minimum λ-tree T. We begin by proving statement 1 for the case where $i = 0$.

So assume $\lambda = 3m$. By Lemma 2.1 the largest legal angle less than $(\lfloor 2\lambda/3 \rfloor - 1)\omega$ is $2(m-1)\omega$. Suppose, contrary to the theorem, that $\angle qpr \leq 2(m-1)\omega$. We regard p as being a (possibly degenerate) Steiner point, so that all variations on p result in a point of degree 3 connected to p, q, and r. Then we can choose a variation of p to p' in a legal direction between the two legs of the angle such that $\angle qpp' \leq (m-1)\omega$

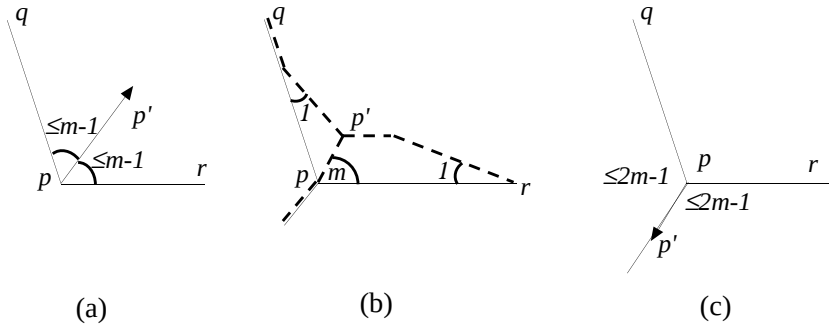


FIG. 2.2. *Variations for $\phi_{\min}$ and $\phi_{\max}$. The labels at angles represent multiples of ω.*

and $\angle p'pr \le (m-1)\omega$ (Figure 2.2(a)). It follows from Lemma 2.2 that, under this perturbation,

$$\dot{T} = \frac{d|pq|_\lambda}{dv} + \frac{d|pr|_\lambda}{dv} + \frac{d|pp'|_\lambda}{dv} \le 2f(m-1) + 1 < 0,$$

contradicting the minimality of T. This shows $\phi_{\min} \ge (\lfloor 2\lambda/3 \rfloor - 1)\omega$, and equality immediately follows from Lemma 2.3. The other two cases, $i = 1$ and $i = 2$, follow by a similar argument.

To prove statement 2, suppose $\angle qpr = (2m-1)\omega$. Consider the variation of p to p' such that $\angle qpp' = (m-1)\omega$ and $\angle p'pr = m\omega$ (Figure 2.2(a)). Note that, under this perturbation,

$$\dot{T} = f(m-1) + f(m) + 1 = -2\cos(m\omega) + 1 = 0,$$

and hence the length of T remains unchanged. Now choose critical paths for the nonstraight edges qp' and rp' such that the corner point of qp' lies on qp but the corner point of rp' does not lie on rp (as in Figure 2.2(b)). Then it follows that the angles at p' all equal $2m\omega = 2\pi/3$. In this way we can transform T into an equally long Steiner λ-tree T' in which any angle is at least equal to $2\pi/3$. □

COROLLARY 2.5. *Any Steiner point in a Steiner minimum λ-tree has degree* 3 *or* 4. *Degree* 4 *Steiner points exist only if $\lambda =$* 2, 4, *or* 6. *If $\lambda =$* 2, 4, *or* 6, *then the angles at degree* 4 *Steiner points are all equal to $\pi/2$.*

For any given value of λ, we define $\phi_{\max}$ to be the maximum angle that can occur at a Steiner point in any Steiner minimum λ-tree.

THEOREM 2.6. 1. *Let $\lambda = 3m + i$. Then*

$$\phi_{\max} = \begin{cases} (\lceil 2\lambda/3 \rceil + 1)\omega & = (2m+1)\omega \quad \text{if } i = 0, \\ \lceil 2\lambda/3 \rceil \omega & = (2m+1)\omega \quad \text{if } i = 1, \\ \lceil 2\lambda/3 \rceil \omega & = (2m+2)\omega \quad \text{if } i = 2. \end{cases}$$

2. *If $\lambda = 3m$, then for any terminal set N there exists a Steiner minimum λ-tree T' such that the maximum angle in T' at a Steiner point is $\lceil 2\lambda/3 \rceil \omega = 2\pi/3$.*

Proof. Clearly, this theorem is symmetric to Theorem 2.4. Hence, it holds by symmetry and by using the second inequalities in formulae (2.2)–(2.4) (Figure 2.2(c)). □

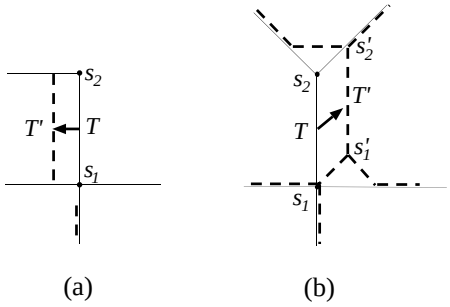


FIG. 3.1. *Two variations for a degree* 4 *Steiner point* s_1 *and its neighboring Steiner point* s_2.

If the angles at a degree 3 Steiner point are α, β, γ, and $\alpha \leq \beta \leq \gamma$, then we define the *angle set* at this point to be the triple (α, β, γ). Combining Theorems 2.4 and 2.6, we have the following corollary.

COROLLARY 2.7. *The only possible angle sets at a degree* 3 *Steiner point in a Steiner minimum* λ*-tree are*

1. $(2m-1, 2m, 2m+1)\omega$ *and* $(2m, 2m, 2m)\omega$ *if* $\lambda = 3m$,
2. $(2m, 2m+1, 2m+1)\omega$ *if* $\lambda = 3m+1$,
3. $(2m+1, 2m+1, 2m+2)\omega$ *if* $\lambda = 3m+2$.

There are also extra restrictions on which angles can appear next to a given nonstraight edge.

COROLLARY 2.8. *Let* ps *be a nonstraight edge in a Steiner minimum* λ*-tree, such that* s *is a Steiner point. Let* c *be the corner point of* ps*, and let* q *and* r *be the other two vertices or corner points immediately adjacent to* s*. If* c *lies on the same side of* ps *as* q *(that is,* $\angle_\lambda psq = \angle csq$*), then* $\angle csq \neq \phi_{\max}$ *and* $\angle csr \neq \phi_{\min}$.

3. Possible structures of Steiner minimum λ-trees. First, we prove a theorem involving degree 4 Steiner points.

THEOREM 3.1. *For any* N *there exists a Steiner minimum* λ*-tree on* N *such that the vertices adjacent to degree* 4 *Steiner points are all terminals.*

Proof. By Corollary 2.5, $\lambda = 2$, 3, 4, or 6. This theorem is well known for $\lambda = 2$ (see, for example, [6]) and follows from statement 2 of Theorem 2.4 when $\lambda = 3$ or 6. So let $\lambda = 4$, and let T be a Steiner minimum λ-tree. By Corollary 2.7, the angle set at any degree 3 Steiner point of T is $(\pi/2, 3\pi/4, 3\pi/4)$.

Assume that s_1 is a degree 4 Steiner point in T. Suppose one of the adjacent vertices of s_1, say s_2, is not a terminal. Clearly, $s_1 s_2$ must be a straight edge, since otherwise the other choice of corner point for that edge would give a contradiction to Corollary 2.4. There are two cases to consider. If s_2 is Steiner point with a $\pi/2$ angle sharing $s_1 s_2$ (Figure 3.1(a)), then we can slide $s_1 s_2$ between two parallel line segments without changing the length of T. This results in a degree 3 Steiner point whose angle set contradicts Corollary 2.7. If a $\pi/2$ angle does not share $s_1 s_2$, then s_2 is a degree 3 Steiner point as in Figure 3.1(b). In this case we can transform T into an equally long Steiner λ-tree T' (shown in dotted lines in the figure) by moving s_2 along one of the legs of this angle and moving s_1 an equal distance in the same

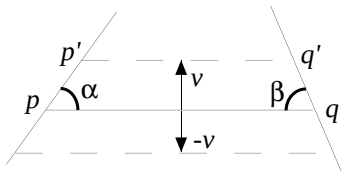


FIG. 3.2. *A parallel shift.*

direction. In this way the degree 4 Steiner point s_1 is eliminated since it has been split into two degree 3 Steiner points s_1, s_1'. □

This theorem means we can often restrict our attention to full Steiner minimum λ-trees with degree 3 Steiner points. Hence it can help reduce the number of tree topologies that need to be considered for a given set of terminals. Unlike the Euclidean case, however, there may be many Steiner minimum λ-trees with the same topology but with different assignments of nonstraight edges and unequal angles. Using the variational argument, we can significantly reduce the number of possible assignments.

We first consider how to minimize the number of nonstraight edges when $\lambda = 3$. (See [3] for details of the case when $\lambda = 3$.) Note that the process of removing degree 4 Steiner points (as illustrated in Figure 3.1(b)) may create extra bent edges. Hence it is still necessary to consider Steiner trees with degree 4 Steiner points when $\lambda = 4$ or 6.

Define a *shift* v of a straight edge pq to be a move of p to p' and a simultaneous move of q to q' such that $p'q' \| pq$ (Figure 3.2). Note that the rate of change in the length of pq under a shift, $d|pq|/dv$, is independent of $|pq|$ but depends only on the angles $\angle p'pq$ and $\angle pqq'$. Clearly, the concept of a shift can be generalized to a path P composed of straight line segments and is well defined once one has specified the directions in which the endpoints of line segments are permitted to move. A variation is said to be *reversible* if $\dot{P}_v = -\dot{P}_{-v}$. It is easy to see that a shift is reversible if each endpoint of a line segment of P is restricted to moving along the same line through that point for both shifts.

LEMMA 3.2. *Let $\lambda = 4$ or 6. Let T be a Steiner minimum λ-tree containing at least two nonstraight edges. If the path, P, between the two nonstraight edges contains a Steiner point s of degree 4, then the angle between the two edges of P incident with s is π.*

Proof. Clearly we can choose the path P in T such that ps_1 and s_kq are nonstraight edges in T and such that all other edges on the path $P = ps_1s_2 \dots s_kq$ are straight.

Suppose, for the moment, that, for any Steiner point on P with degree 4 in T, the two incident edges in P meet at an angle of π. For each s_i on P, let r_i be a vertex or corner point adjacent to s_i not lying on P. Let the corner points on ps_1 and s_kq be c and d, respectively. Let v be the variation of shifting $cs_1s_2 \dots s_kd$ to one side of the path such that c moves towards p, each point s_i moves along the line through s_ir_i, and d moves along the line through dq. This is illustrated in Figure 3.3. In this case the variation is clearly reversible, that is, $\dot{T}_v = -\dot{T}_{-v}$.

Now suppose, contrary to the lemma, that P contains a Steiner point s with

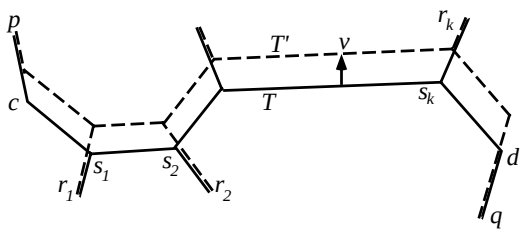


FIG. 3.3. *Shifting a path in T to reduce the number of corner points.*

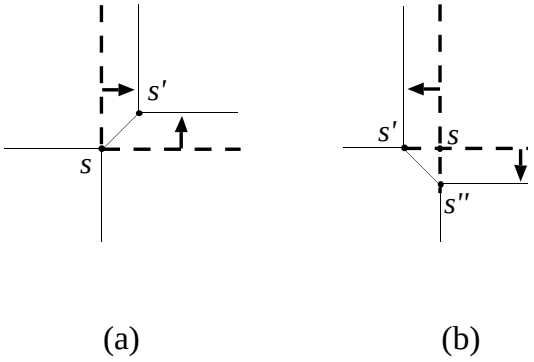


FIG. 3.4. *Two variations for a degree* 4 *Steiner point s and two of its incident edges.*

degree 4 in T such that the two incident edges in P meet at an angle of $\pi/2$. By Figure 3.4 there exist two perturbations on s and its two incident edges in P such that these perturbations have the following properties.

(1) They each split s into two Steiner points (s and s' in Figure 3.4(a), and s' and s'' in Figure 3.4(b)).

(2) One perturbation shifts the two incident edges to one side of P, while the other shifts them to the opposite side of P.

(3) In each case they reduce the length of T in the vicinity of s.

By property (2) we can incorporate these perturbations into the shift variations v and $-v$ described above. It then follows, by property (3), that $\dot{T}_v < 0$ or $\dot{T}_{-v} < 0$, contradicting the minimality of T. □

THEOREM 3.3. *For any N there exists a fulsome Steiner minimum λ-tree on N such that in each component at most one edge is nonstraight.*

Proof. Assume that $\lambda > 3$, and suppose that T is a full component of a fulsome Steiner minimum λ-tree on N containing more than one nonstraight edge. We begin with a similar construction to the previous lemma. There exists a pair of nonstraight edges ps_1 and s_kq in T such that all other edges on the path $P = ps_1s_2 \dots s_kq$ are straight. For each s_i on P let r_i be a vertex or corner point adjacent to s_i not lying

on P. Let the corner points on ps_1 and s_kq be c and d, respectively. Let v be the variation of shifting $cs_1s_2 \ldots s_kd$ to one side of the path such that c moves towards p, each point s_i moves along the line through s_ir_i, and d moves along the line through dq. By Lemma 3.2, v is well defined and is reversible, that is, $\dot{T}_v = -\dot{T}_{-v}$.

It follows that $\dot{T}_v = 0$ since T is minimal. Therefore, such a shift does not change the length of T. We can continue to perform this shift without changing the length of T until one of the following occurs.

(1) Two points s_i and s_{i+1} coincide, or a point s_i coincides with a Steiner point r_i.

(2) A point s_i coincides with a terminal r_i.

(3) A point s_i coincides with a corner point r_i, or c or d coincides with a terminal or Steiner point.

If (1) occurs and $\lambda \neq 4$ or 6, then we have a contradiction to the fact that T is minimum. If (1) occurs and $\lambda = 4$ or 6, then by Lemma 3.2 the two edges in P incident to this point must meet at an angle of π, and hence we can continue to perform the shift. If (2) occurs, we have a contradiction to the fact that T is fulsome. Hence, one of the possibilities in (3) must eventually occur, reducing the number of nonstraight edges. In this way T can be transformed into a λ-tree of equal length containing at most one nonstraight edge. □

Remark 3.1. We conjecture that Theorems 3.1 and 3.3 can always hold simultaneously; that is, if $\lambda = 4$ or 6, there exists for any terminal set a Steiner minimum tree such that each full component contains at most one nonstraight edge, and each degree 4 Steiner point is adjacent to four terminals. In light of Theorem 3.1 and this conjecture, it will be convenient for the remainder of this paper to use the term "Steiner topology" to refer to a tree topology in which all nonterminals have degree no more than 3.

Suppose N is a set of n terminals. Let $GG_0(N) = N$, and recursively define $GG_i(N)$ to be the set of grid points that are intersections of the legal lines through all points in $GG_{i-1}(N)$. The so-called multilevel grid theorem in [8] is now a simple corollary of the above theorem.

COROLLARY 3.4. *For each set N of n terminals, there exists a Steiner minimum λ-tree T for N such that all Steiner points in T are grid points of $GG_{n-2}(N)$.*

Proof. If all edges of T are straight edges, then each Steiner point adjacent to two terminals lies on $GG_1(N)$, and each Steiner point adjacent to two vertices of T on $GG_{i-1}(N)$ lies on $GG_i(N)$. Since we can assume, in general, that T contains at most one nonstraight edge, it follows that all Steiner points in the two connected components of T minus that nonstraight edge lie on $GG_{n-2}(N)$. □

The shift technique also helps us to reduce the number of feasible angle assignments. Consider again the shift of pq shown in Figure 3.2. Let $\alpha = \angle p'pq$, $\beta = \angle pqq'$, and $l = |pq|$. Then we have

$$\frac{d}{dv}|pq| = \dot{l}(v) = -\cos\alpha - \cos\beta.$$

Now assume all edges are in λ-geometry and hence $\alpha = k_1\omega, \beta = k_2\omega$. Define a function $g(k_1, k_2)$ as follows:

$$g(k_1, k_2) = -\cos(k_1\omega) - \cos(k_2\omega).$$

This is similar to $f(k)$ which expresses the variation of moving a point, except that g expresses the variation of shifting a straight edge. Just as for the inequalities in Lemma 2.2, it is easy to show that the following inequalities hold.

1. If $\lambda = 3m$, then

$$2f(m-1) + g(m,m) < -2, \tag{3.1}$$

$$2f(2m) + g(2m-1, 2m-1) < 2, \tag{3.2}$$

$$2f(2m-1) + g(2m, 2m) < 2, \tag{3.3}$$

$$f(2m) + f(2m-1) + g(2m, 2m-1) < 2. \tag{3.4}$$

2. If $\lambda = 3m+1$, then

$$2f(m) + g(m,m) < -2. \tag{3.5}$$

3. If $\lambda = 3m+2$, then

$$2f(2m+1) + g(2m+1, 2m+1) < 2. \tag{3.6}$$

Suppose that s_1s_2 is a straight edge joining two Steiner points. The angles at s_1, s_2, sharing s_1s_2 and lying on the same side of s_1s_2, are referred to as *neighboring angles*. Some restrictions occur on neighboring angles when they are either both $\phi_{\min}$ or both $\phi_{\max}$.

THEOREM 3.5. *In a Steiner minimum λ-tree T two neighboring angles cannot both be*

1. $\phi_{\min}$ *or both be* $\phi_{\max}$ *if* $\lambda = 3m$,
2. $\phi_{\min}$ *if* $\lambda = 3m+1$,
3. $\phi_{\max}$ *if* $\lambda = 3m+2$.

Proof. Consider the case of $\lambda = 3m$. First, suppose, to the contrary, that there are two neighboring angles $\angle ps_1s_2 = \angle s_1s_2q = \phi_{\min} = 2m-1$ in T. Then perform a variation by shifting s_1s_2 so that s_1 and s_2 each move at an angle of $m\omega$ to s_1s_2, as shown in Figure 3.5(a). We have $\dot{T} \le 2f(m-1) + g(m,m) + 2 < 0$ by inequality (3.1). Next, if $\angle ps_1s_2 = \angle s_1s_2q = \phi_{\max} = 2m+1$, then there are 3 cases as shown in Figures 3.5(b)–3.5(d) corresponding to the different assignments of angles around s_1 and s_2. In each case, we perform the shift by shrinking the third edges at s_1 and s_2, effectively stretching s_1s_2. Then $\dot{T} < 0$ by inequalities (3.2)–(3.4). The other two statements of the theorem can be proved similarly by inequalities (3.5) and (3.6) using the shifts illustrated in Figures 3.5(e) and 3.5(f). □

Remark 3.2. The above proof demonstrates that in order to show $\dot{T} < 0$ for the given cases it suffices to shrink s_1s_2 for two neighboring angles $\phi_{\min}$ and to stretch s_1s_2 for two neighboring angles $\phi_{\max}$. Note, however, that there are other possible shifts we could have chosen in the proof. For example, in the case shown in Figure 3.5(b), the nonminimality of T can be derived from the fact that it is covered by Figure 3.5(a).

Remark 3.3. Theorem 3.5 does not exhaust all possibilities for assignments of angles that cannot occur at two neighboring Steiner points in a Steiner minimum λ-tree. There is one remaining infeasible configuration, which occurs for $\lambda = 3m$. In this case the neighboring angles at s_1, s_2 are, respectively, $(2m-1)\omega$, $2m\omega$ on one side of s_1s_2 and $2m\omega$, $(2m+1)\omega$ on the other, as shown in Figure 3.6. Now perform a shift variation as shown in the figure, where s_1 and s_2 both move in the same direction and the critical paths are chosen so that the new corner points p' and

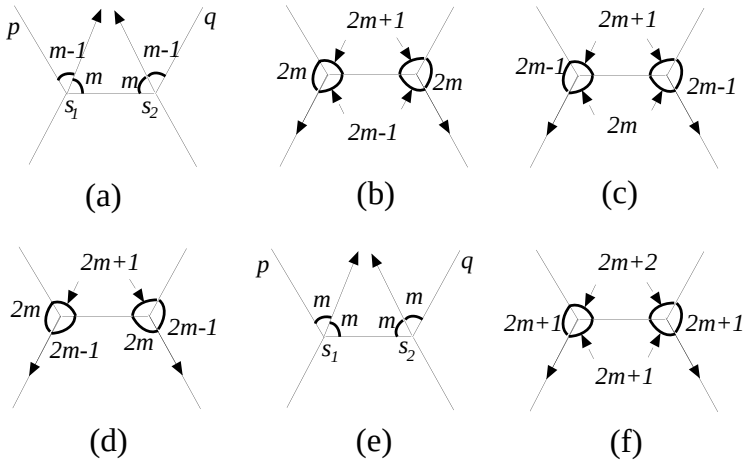


FIG. 3.5. *Infeasible pairs of neighboring angles, both of which are $\phi_{\min}$ or $\phi_{\max}$. The labels at angles represent multiples of ω.*

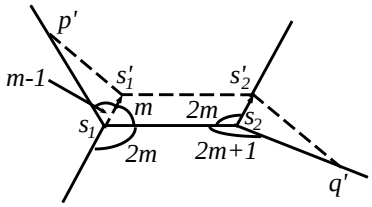


FIG. 3.6. *Another infeasible angle configuration for $\lambda = 3m$.*

q' both lie on edges of the original λ-tree. Then $|p's_1| + |s_2q'| > |p's_1'| + |s_2'q'|$ because $p's_1' \parallel s_2'q'$. Hence, the original tree is not minimal.

The λ-metric for $\lambda = 3$ (hexagonal metric) is a special metric in λ-metrics because $\omega = \phi_{\min} = \pi - \phi_{\max}$. When $\lambda = 3$, for any point set N in the λ-plane there is a Steiner minimum λ-tree such that all Steiner points are vertices of $GG_{\lceil (n-2)/2 \rceil}$ [15], which is a significant improvement on Corollary 3.4. It is not difficult, however, to see that this result cannot be generalized to larger values of λ. Moreover, it has been proved in [12] that for any set N and any given Steiner topology there exists a Steiner λ-tree (with $\lambda = 3$) such that the unique nonstraight edge in each full component is incident to a terminal, and that all angles at Steiner points are $2\pi/3$. Therefore, it directly follows that Melzak's construction for Euclidean Steiner trees [9] can also be used to construct Steiner λ-trees for $\lambda = 3$. An algebraic algorithm, which is equivalent to Melzak's construction but much more suitable for computer implementation, was developed [13],[5]. In the next section we show that in a certain sense Melzak's construction can apply to λ-trees for any λ.

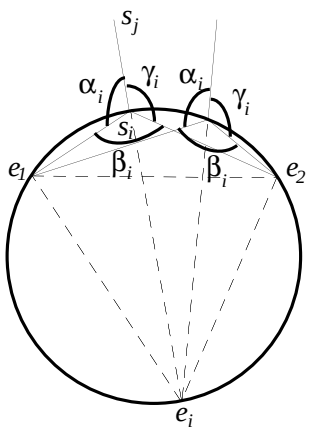


FIG. 4.1. *Merging two terminals.*

4. Steiner trees with fixed angles. To generalize Melzak's construction for Euclidean Steiner trees, which is a geometric construction using only ruler and compass, to Steiner λ-trees, we first give a brief description of the method in the Euclidean setting. Suppose the topology of a Steiner tree T spanning a terminal set N in the Euclidean plane is given. Since each full component is constructed independently, we can assume T is full. Melzak's construction is divided into two stages. The basic operation at the first stage (orientation stage) is as follows. Suppose a Steiner point s_i in T is adjacent to two terminals e_1, e_2, and suppose the third adjacent vertex is s_j. Let e_i be the third vertex of the equilateral triangle $e_1e_ie_2$ that lies on the side of e_1e_2 opposite to s_i. (The correct side can be determined by the other terminals, as shown in [4].) Without loss of generality, assume e_1, e_i, e_2 are in counterclockwise order. An easy trigonometric argument shows that s_i must lie on the minor arc $\overset{\frown}{e_1e_2}$ of the circle circumscribing e_1, e_2, and e_i, and that the point e_i lies on the major arc $\overset{\frown}{e_1e_2}$ (Figure 4.1) and satisfies the properties

(P1) e_i lies on the extension of s_js_i, and

(P2) $|e_1s_i| + |e_2s_i| = |e_is_i|$.

From these properties it follows that we can replace e_1, e_2 with e_i. (Hence, e_i is referred to as a *merging point*.) Repeat this operation until only two points are left. Then the line segment joining the two points, called a *Simpson line* of T, gives

1. the orientation of an edge of T by (P1), and
2. the length of T, which equals the length of the Simpson line, by (P2).

Once the Simpson line is determined, all Steiner points can be determined by backtracking at the second stage (reconstruction stage).

Related to this method, a notation describing a full Steiner topology was proposed [1]. Since e_1, e_2 are replaced by a merging point in this construction, the association of e_1, e_2 can be denoted as (e_1e_2). In this way, the topology of T can be represented by a sequence of terminals that are bracketed in pairs according to the order of the association of terminals. For example, the topology in Figure 4.2(b) can be represented as $(a(bc))(de)$.

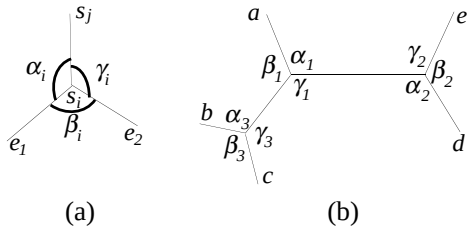


FIG. 4.2. *Steiner patterns.*

Property (P1) is very much the core of Melzak's construction, whereas property (P2), though useful, is not essential. Without using (P2), we can still find the length of T since it is the sum of all edge lengths. These edge lengths can be computed after the Steiner points are determined at the reconstruction stage. Thus, Melzak's construction can be generalized to a wide range of Steiner trees in which all angles at Steiner points are fixed.

A full Steiner topology accompanied by information about the size of all angles at Steiner points will be referred to as a *Steiner pattern*. A suitable notation for Steiner patterns can be developed by generalizing the notation for a Steiner topology. As before, suppose s_i is a Steiner point adjacent to two terminals e_1, e_2, and the third adjacent vertex is s_j. Suppose the angles at s_i are $\angle e_1 s_i s_j = \alpha_i$, $\angle e_2 s_i e_1 = \beta_i$, and $\angle s_j s_i e_2 = \gamma_i$ (Figure 4.2(a)). Then, the association of e_1, e_2 is denoted as $(\alpha_i(e_1 \beta_i e_2)\gamma_i)$. In this way the pattern in Figure 4.2(b) can be represented as

$$(\alpha_1(a\beta_1(\alpha_3(b\beta_3 c)\gamma_3))\gamma_1)(\alpha_2(d\beta_2 e)\gamma_2).$$

It is easy to see that this pattern can be simplified by removing all brackets because the insertion of angles makes it possible to find the order of association of terminals. For example, the above pattern can be easily retrieved from the *simplified pattern*

$$\alpha_1 a \beta_1 \alpha_3 b \beta_3 c \gamma_3 \gamma_1 \alpha_2 d \beta_2 e \gamma_2.$$

In fact, this pattern is the sequence of terminals and angles occurring in turn in a counterclockwise tour of the topology.

Suppose T is a full Steiner tree with a given pattern. Then the basic operation in constructing the tree can be modified as follows. Let e_i be the third vertex of the triangle $e_1 e_i e_2$ in which $\angle e_2 e_1 e_i = \pi - \gamma_i, \angle e_i e_2 e_1 = \pi - \alpha_i$. Clearly, e_1, e_i, e_2, s_i lie on a circle (as in Figure 4.1). Since the angles at s_i are fixed, property (P1) holds: that is, the extension of $s_j s_i$ must go through e_i no matter where s_i lies on $\overset{\frown}{e_1 e_2}$. Hence we still can find the orientation of an edge of T, and we can construct T by backtracking. However, (P2) does not hold unless all angles at each s_i equal $2\pi/3$. Otherwise, the length of the tree can be computed after all Steiner points are determined, as mentioned above.

We can now generalize the Steiner pattern and Melzak's construction to Steiner λ-trees. Suppose T is a full Steiner λ-tree in which $s_1 s_2$ is the unique nonstraight edge by Theorem 3.3. Then the angles at all Steiner points other than s_1, s_2 are fixed.

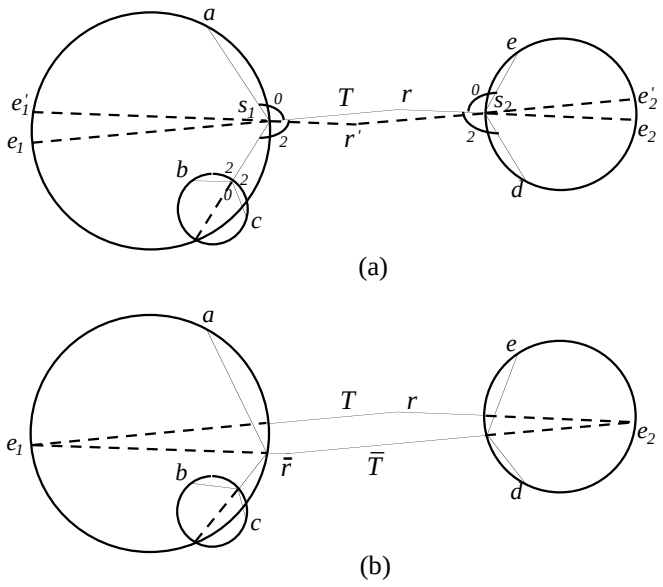


FIG. 4.3. *Melzak's construction and Steiner patterns in the λ-plane.*

If one of the critical paths of s_1s_2, say, s_1rs_2, is chosen as the nonstraight edge, then the angles at s_1, s_2 are also fixed. Thus T can be regarded as the union of two full subtrees T_1, T_2 joined at the corner point r. Let w_1, w_2 be the patterns of the subtrees T_1, T_2, respectively. Then we define $w_1 \frown w_2$ to be the pattern of the λ-tree T. Note that all edges in T_1, T_2 are straight and that all angles correspond to the possibilities given in Corollary 2.7. In the simplified pattern of λ-trees, let $\phi_{\min}$ be denoted by 0, let $\phi_{\max}$ be denoted by 2, and let the angle $2m$ for $\lambda = 3m$, which is between $\phi_{\min}$ and $\phi_{\max}$, be denoted by 1. Then, for example, the pattern of the λ-tree in Figure 4.3(a), whose topology is the same as the tree in Figure 4.2(b), is

$$0a22b0c22 \frown 2d2e0.$$

Remark 4.1. If the other critical path $s_1r's_2$ is chosen as the nonstraight edge, then the pattern becomes $2a22b0c20 \frown 0d2e2$ (Figure 4.3(a)). Although the merging points are different for the two patterns, as shown in the figure, the Steiner points s_1, s_2 are the same. Therefore, the two patterns can be regarded as equivalent.

Because both T_1, T_2 are full Steiner trees with fixed angles, they can be constructed by Melzak's method as stated above. Hence, as shown in Figure 4.3, we can find merging points e_1, e_2 after merging all terminals in T_1, T_2, respectively. Now there may exist two choices of the nonstraight edge e_1e_2 as shown in Figure 4.3(b), and two λ-trees $T, \overline{T}$ can be constructed. Because (P2) does not hold, $|T|_\lambda$ may not equal $|\overline{T}|_\lambda$ though two paths of e_1e_2 (e_1re_2 and $e_1\bar{r}e_2$) are equally long. Only by comparing the lengths of T and $\overline{T}$ are we able to determine which one is shorter.

5. Concluding remarks. Sections 3 and 4 essentially give two finite exact algorithms for solving the Steiner problem in a λ-plane. The strategy suggested by

Corollary 3.3 is to treat the problem as a graphical Steiner tree problem on the grid $GG_{n-2}(N)$, and then use standard techniques for Steiner trees on graphs (see, for example, [6]). Unfortunately, the number of grid points in $GG_{n-2}(N)$ has order $O(n^{2^{n-2}})$, making such an approach impractical for large n. The alternative algorithm suggested by section 4 is to run through all possible patterns for a Steiner minimum λ-tree. The modified version of Melzak's theorem then allows the minimum λ-tree for each pattern to be constructed in linear time (independently of λ). The practical disadvantage of this approach lies in the fact that the number of Steiner topologies increases at an exponential rate as n increases. Furthermore, for each Steiner topology the number of different patterns that needs to be considered is potentially also an exponential function of n. Noting that a Steiner minimum λ-tree can be found in linear time for a given Steiner topology if $\lambda = 2, 3$, or ∞, we have the following tantalizing but apparently difficult open question.

Open question. Given $\lambda > 3$ and a terminal set N, does there exist a polynomial-time algorithm for finding a Steiner minimum λ-tree for any given Steiner topology on N?

We will consider a number of issues connected with this question in a forthcoming paper, in which we will also develop a more practical exact algorithm for solving the Steiner minimum problem in the λ-plane.

REFERENCES

[1] E. J. Cockayne, *On the efficiency of the algorithm for Steiner minimal trees*, SIAM J. Appl. Math., 18 (1970), pp. 150–159.

[2] D. Z. Du and F. K. Hwang, *A proof of the Gilbert-Pollak conjecture on the Steiner ratio*, Algorithmica, 7 (1992), pp. 121–135.

[3] D. Z. Du and F. K. Hwang, *Reducing the Steiner problem in a normed space*, SIAM J. Comput., 21 (1992), pp. 1001–1007.

[4] F. K. Hwang, *A linear time algorithm for full Steiner trees*, Oper. Res. Lett., 5 (1986), pp. 235–237.

[5] F. K. Hwang and J. F .Weng, *Hexagonal coordinate systems and Steiner minimal trees*, Discrete Math., 62 (1986), pp. 49–57.

[6] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem,* Ann. Discrete Math. 53, North-Holland, Amsterdam, 1992.

[7] C. K. Koh, *Steiner Problem in Octilinear Routing Model*, Master's thesis, National University of Singapore, Singapore, 1995.

[8] D. T. Lee and C. F. Shen, *The Steiner minimal tree problem in the λ-geometry plane*, in Algorithms and Computation, Lecture Notes in Comput. Sci. 1178, Springer-Verlag, New York, 1996, pp. 247–255.

[9] Z. A. Melzak, *On the problem of Steiner*, Canad. Math. Bull., 4 (1961), pp. 143–148.

[10] J. H. Rubinstein and D. A. Thomas, *A variational approach to the Steiner network problem*, Ann. Oper. Res., 33 (1991), pp. 481–499.

[11] M. Sarrafzadeh and C. K. Wong, *Hierarchical Steiner tree construction in uniform orientations*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1095–1103.

[12] J. F. Weng, *Steiner Problem in Hexagonal Metric*, manuscript.

[13] J. F. Weng, *Generalized Steiner problem and hexagonal coordinate system*, Acta Math. Appl. Sinica, 8 (1985), pp. 383–397 (in Chinese).

[14] P. Widmayer, Y. F. Wu, and C. K. Wong, *On some distance problems in fixed orientations*, SIAM J. Comput., 16 (1987), pp. 728–746.

[15] G. Y. Yan, A. Albrecht, G. H. F. Young, and C. K. Wong, *The Steiner tree problem in orientation metrics*, J. Comput. System Sci., 55 (1997), pp. 529–546.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1594–1623

GENERAL DYNAMIC ROUTING WITH PER-PACKET DELAY GUARANTEES OF O(DISTANCE + 1/SESSION RATE)*

MATTHEW ANDREWS†, ANTONIO FERNÁNDEZ‡, MOR HARCHOL-BALTER§, TOM LEIGHTON¶, AND LISA ZHANG†

Abstract. A central issue in the design of modern communication networks is that of providing performance guarantees. This issue is particularly important if the networks support real-time traffic such as voice and video. The most critical performance parameter to bound is the delay experienced by a packet as it travels from its source to its destination.

We study dynamic routing in a connection-oriented packet-switching network. We consider a network with arbitrary topology on which a set of sessions is defined. For each session i, packets are injected at a rate r_i to follow a predetermined path of length d_i. Due to limited bandwidth, only one packet at a time may advance on an edge (link). Session paths may overlap subject to the constraint that the total rate of sessions using any particular edge is at most $1-\varepsilon$ for any constant $\varepsilon \in (0,1)$.

We address the problem of scheduling the sessions at each switch, so as to minimize worst-case packet delay and queue buildup at the switches. We show the existence of a periodic schedule that achieves a delay bound of $O(1/r_i + d_i)$ with only constant-size queues at the switches. This bound is asymptotically optimal for periodic schedules.

A consequence of this result is an asymptotically optimal schedule for the static routing problem, wherein all packets are present at the outset. We obtain a delay bound of $O(c_i + d_i)$ for packets on path P_i, where d_i is the number of edges in P_i and c_i is the maximum congestion along edges in P_i. This improves upon the previous known bound of $O(c+d)$, where $d = \max_i d_i$ and $c = \max_i c_i$.

We also present a simple distributed algorithm that, with high probability, delivers every session-i packet to its destination within $O(1/r_i + d_i \log(m/r_{\min}))$ steps of its injection, where $r_{\min}$ is the minimum session rate and m is the number of edges in the network. Our results can be generalized to (leaky-bucket constrained) bursty traffic, where session i tolerates a burst size of b_i. In this case, our delay bounds become $O(b_i/r_i + d_i)$ and $O(b_i/r_i + d_i \log(m/r_{\min}))$, respectively.

Key words. communication networks, packet routing, scheduling, delay bounds

AMS subject classifications. 68M20, 68M10, 68W40

PII. S009753979935061X

1. Introduction.

1.1. Motivation.

Motivated by the need for quality-of-service guarantees, network designers today offer *connection-oriented* service in many networks, e.g., ATM (asynchronous transfer mode) networks. In this medium, a user requests a particular share of the bandwidth and injects a stream of packets along one particular session at the agreed-upon rate. An important consequence of the user's predictability is that the network can, in return, guarantee the user an *end-to-end delay bound*, i.e.,

*Received by the editors January 29, 1999; accepted for publication (in revised form) July 17, 2000; published electronically November 28, 2000. Supported by Army grant DAAH04-95-1-0607 and ARPA contract N00014-95-1-1246.

http://www.siam.org/journals/sicomp/30-5/35061.html

†Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974 (andrews@research.bell-labs.com, ylz@research.bell-labs.com). The work of the first author was supported by NSF contract 9302476-CCR. The work of these authors was performed while they were at MIT.

‡Universidad Rey Juan Carlos, Móstoles, Spain (a.fernandez@escet.urjc.es). The work of this author was done while he was at MIT and was supported by the Spanish Ministry of Education.

§Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA (harchol@cs.cmu.edu). The work of this author was done while she was at MIT and was supported by an NSF Postdoctoral Fellowship in the Mathematical Sciences.

¶Department of Mathematics and Laboratories for Computer Science, MIT, Cambridge, MA (ftl@math.mit.edu).

an upper bound on the time that any packet takes to move from its source to its destination. In order to provide this delay guarantee, the network must determine how to schedule the packets that contend for the same edge simultaneously. Apart from delay bounds, it is also important to guarantee small queues at each switch due to limited buffer size. In this paper we show how to design schedules that guarantee asymptotically optimal per-session delay bounds as well as small queues.

1.2. Model and problem. Consider a network $\mathcal{N}$ of arbitrary topology and a set of sessions defined on this network. A session i is associated with a source node, a destination node, and a simple path from the source to the destination. (A path is *simple* if it uses each edge at most once.) Packets are injected to the network $\mathcal{N}$ in sessions. A packet injected in session i enters the system at the source node of i, traverses the path associated with i, and then is absorbed at its destination. The length d_i is the number of edges on the path from the source to the destination of session i.

Each session i has an associated injection rate r_i. This rate constrains the injection of new packets from the session so that, during any interval of t consecutive steps, at most $tr_i + 1$ packets can be injected in session i for any t.

We assume that all packets have the same size and all edges have the same bandwidth. We also assume a synchronized store-and-forward routing, where at each step at most one packet can traverse each edge. When two packets simultaneously contend for the same edge, one packet has to wait in a queue. During the routing, packets wait in two different kinds of queues. After a packet has been injected but before it leaves its source, the packet is stored in an *initial queue*. Once the packet has left its source, during any time it is waiting to traverse an edge, the packet is stored in an *edge queue*. The *end-to-end delay* (delay for short) for a packet is the total time from the packet injection until it reaches its destination. This includes the total time the packet spends waiting in both types of queues, plus the time it spends traversing edges.

Our goal is to minimize both the end-to-end delay for each packet and the length of all edge queues. In order to achieve delay guarantees and bounded queue sizes, it is necessary to require that, for all edges e, the sum of the rates of the sessions that use edge e is at most 1. Throughout, we shall assume that the sum of the rates of the sessions using any edge e is at most $1 - \varepsilon$ for a constant $\varepsilon \in (0, 1)$. This constant ε will appear throughout our subsequent bounds.

Our paper focuses on the problem of timing the movements of the packets along their paths. A *schedule* specifies which packets move and which packets wait in queues at each time step. Most of the schedules obtained in this paper are *template-based*. The schedule defines a fixed *template* for each edge in advance. A template of size M is a wheel with M slots, each of which contains at most one token. Each token is affiliated with some session. The wheel spins at the speed of one slot per time step. A session-i packet can traverse the edge if and only if a session-i token appears. For each session-i token, the session-i packet that uses it will be the one that has been waiting to cross the edge for the longest amount of time, i.e., the session-i packets use the session-i tokens in a first-come-first-served manner. The template size and associated tokens do not change over time.

We show that per-session delay bounds that are asymptotically optimal for template-based schedules can be achieved. Meanwhile, constant-size edge queues can be achieved as well.

1.3. Lower bounds. We observe that d_i is always a lower bound on the delay for session i, since every session-i packet has to cross d_i edges. It is also easy to see that $\Omega(1/r_i)$ is an existential lower bound. For instance, consider n sessions, all of which have the same rate $r = (1-\varepsilon)/n$ and the same initial edge e. If a packet is injected in each session simultaneously, one of the packets requires $n = \Omega(1/r)$ steps to cross e.

Furthermore, for *any* given set of sessions, $\Omega(1/r_i)$ is a lower bound for some session i in template-based schedules. Consider the template for an edge e where $\sum_e r_i = 1-\varepsilon$. By the pigeon-hole principle, the tokens for some session i can occupy at most an $r_i/(1-\varepsilon)$ fraction of the slots on the template. Hence, there exist two session-i tokens that are separated by at least $(1-\varepsilon)/r_i$ slots. As a result, an adversary can make sure a session-i packet arrives just after the first token has passed, thereby forcing the packet to wait $\Omega(1/r_i)$ steps.

If the schedule is not restricted to being template-based, the scheduler is more powerful. The scheduler does not have to decide on a fixed schedule in advance, but rather can make a new decision at each step, based on seeing the adversary's injections. In this case it is unknown if for *any* given set of sessions $\Omega(1/r_i)$ is a lower bound.

1.4. Previous work. The problem of dynamic packet routing in the above setting is well studied. Until recently, the best delay bound known was $O(d_i/r_i)$ for packets of session i. It is tempting to believe that this is the best possible delay bound, since a session-i packet may need to wait $\Omega(1/r_i)$ steps to cross each of the d_i edges on its route. However, this upper bound of $O(d_i/r_i)$ can be much improved.

In 1990, Demers, Keshav, and Shenker [8] proposed a widely studied routing algorithm called Weighted Fair Queueing (WFQ). WFQ is a packetized approximation of the idealized fluid model algorithm Generalized Processor Sharing (GPS). WFQ is simple and distributed. This same algorithm was proposed independently by Parekh and Gallager [14, 15] in 1992 under the name of Packet-by-Packet Generalized Processor Sharing (PGPS). Parekh and Gallager prove that the algorithm has an end-to-end delay guarantee of $2d_i/r_i$ [15, p. 148] in the case when all packets have the same size.

In their 1996 paper, Rabani and Tardos [16] produce an algorithm that routes every packet to its destination with probability $1-p$ in time

$$O(1/r_{\min}) + (\log^* p^{-1})^{O(\log^* p^{-1})} d_{\max} + poly(\log p^{-1}),$$

where $r_{\min} = \min_i r_i$ and $d_{\max} = \max_i d_i$. Ostrovsky and Rabani improve the bound to $O(1/r_{\min} + d_{\max} + \log^{1+\varepsilon} p^{-1})$ [13]. These bounds are not session-based, meaning that if one session has a small rate or a long path, then the delay bounds for all sessions will suffer. The algorithms of [13, 16] are distributed, where knowledge of the entire network is not assumed, but each packet carries some information.

The main technique of [13, 16] is based on "delay-insertion." The intuition here is that if each packet receives a large random initial delay, then the packets are sufficiently spread out to ensure that they only need to wait $O(1)$ steps at each successive edge rather than $\Omega(1/r_i)$ steps. This delay-insertion technique is used extensively by Leighton et al. in [10, 11] in the context of static routing. (In the *static* routing problem, all packets are present in the network initially.) Since our main result employs many techniques from [10], we give a detailed summary of [10] in section 4.1.

A contrasting model, the *connectionless adversarial queueing model*, is also much studied, e.g., [4, 1]. Here the paths on which packets are injected can change over

time, giving the adversary more power. In the adversarial queueing model the best delay bound known is polynomial in the maximum path length [1].

1.5. Our results. We first provide a randomized, distributed scheduler that achieves a delay bound for session-i packets of $O(1/r_i + d_i \log(m/r_{\min}))$ and a bound on the queue size of $O(\log(m/r_{\min}))$, where m is the number of edges in the network and $r_{\min} = \min_i r_i$. While this bound is not optimal, it nevertheless conveys some intuition for our main result.

The main contribution of this paper is an asymptotically optimal template-based schedule. We prove that a schedule exists for the dynamic routing problem such that the end-to-end delay of each session-i packet is bounded by $O(1/r_i + d_i)$.[1] Our result improves upon previous work in several aspects.

- We provide a session-based delay guarantee. That is, packets from sessions with short paths and high injection rates reach their destinations quickly. This is a big improvement over the previous bounds, which are stated in terms of $r_{\min} = \min_i r_i$ and $d_{\max} = \max_i d_i$. We also guarantee that every packet always reaches its destination within the delay bound, without dropping any packets.
- We guarantee constant-size edge queues. This is interesting because edge queues are much more expensive than initial queues in practice.
- A consequence of our result is a packet-based bound, which improves upon the $O(c + d)$ bound in [10] for the static problem. (See section 4.1 for the problem and parameter definitions.) We show that if packet p_i follows a route P_i, then p_i can be routed to its destination within $O(c_i + d_i)$ steps, where c_i is the maximum congestion along P_i and d_i is the number of edges on P_i. This result trivially follows from our result by creating a different session i for each packet p_i, and defining $r_i = (1-\varepsilon)/c_i$, where ε is a positive constant used to ensure that the load on any edge is under 1.

For a template-based schedule, even if the computation of the schedule is time-consuming, it needs to be done only once. Packets can then be scheduled indefinitely as long as the sessions do not change.

Leaky-bucket injection model. Our results above can be generalized to bursty traffic streams that are *leaky-bucket regulated*. Here, each session i has a maximum burst size (or bucket size) of $b_i \geq 1$ and an average arrival rate of r_i. During any t consecutive time steps at most $r_i t + b_i$ session-i packets are injected. Leaky-bucket regulated traffic is widely used in the literature, e.g., [6, 7, 9, 14, 15, 18].

Leaky-bucket regulated injections allow traffic shaping. When session-i packets are injected, they first enter the session-i bucket at the source. These packets then leave the bucket one at a time at the rate of r_i. In this way, the end-to-end delay is separated into two components: delay in the bucket and delay in the network. Since delay in the bucket is at most b_i/r_i, the end-to-end delay is increased by at most b_i/r_i steps, and the size of the edge queues is unchanged.

The rest of the paper is divided into sections as follows. We first give some definitions and preliminary results in section 2. Then, in section 3, we describe a simple distributed scheduler that has a delay bound of $O(1/r_i + d_i \log(m/r_{\min}))$. In section 4, we overview the major techniques employed to achieve the main result: a bound of $O(1/r_i + d_i)$ and constant-size edge queues. In section 5 we define a set of

[1] In this paper, we concentrate on proving the existence of such a schedule. However, the proof can be made constructive using ideas of Leighton, Maggs, and Richa [11] that are based on Beck's algorithm [3]. For details, see [19].

parameters used in the proof of the main result, and in section 6 we provide a detailed proof of the main result.

2. Preliminaries. In this section we present some preliminary results. Section 2.1 proves a generic fact about "token sequences" for template-based schedules. Section 2.2 presents two lemmas for probabilistic analysis that will be used extensively throughout the paper.

2.1. Token sequences. Throughout the paper we define template-based schedules in terms of *token sequences*. A token sequence for session i consists of d_i session-i tokens, $\mathcal{K}_1$, ..., $\mathcal{K}_{d_i}$, one from each template along the session-i path, where $\mathcal{K}_{j+1}$ appears $x_j > 0$ steps after $\mathcal{K}_j$. Then x_j is the *token lag* for these two tokens and $\sum_{j=1}^{d_i-1} x_j$ is the end-to-end delay for this token sequence. Two token sequences cannot have tokens in common.

In the following, we show that in any template-based schedule, bounding the delay for token sequences is sufficient to bound the packet delays and that bounding the token lag is sufficient to bound the edge queues. Our proof relies on Lemma 2.1. A vector $\vec{v} = [v_1, v_2, \ldots, v_n]$ is *sorted* if $v_1 \le v_2 \le \cdots \le v_n$. We define $perm(\vec{v})$ to be a sorted vector whose components form a permutation of the components of $\vec{v}$. We also use the notation $\vec{u} < \vec{v}$ to indicate that the jth component of $\vec{u}$ is smaller than the jth component of $\vec{v}$ for each j.

LEMMA 2.1. *Let $\vec{u} = [u_1, u_2, \ldots, u_n]$ and $\vec{v} = [v_1, v_2, \ldots, v_n]$ be two vectors, each of which consists of n distinct numbers. Suppose $\vec{u}$ is sorted, i.e., $perm(\vec{u}) = \vec{u}$, and suppose $\vec{u} < \vec{v}$. Then, the following hold.*

1. *$perm(\vec{u}) < perm(\vec{v})$.*
2. *If $\vec{v} < \vec{u} + \vec{z}$, then $perm(\vec{v}) < perm(\vec{u}) + \vec{z}$, where $\vec{z} = [z, \ldots, z]$ is a vector of n z's for a scalar z.*
3. *Let $|\vec{v}|$ represent the maximum component of $\vec{v}$; then $|perm(\vec{v}) - perm(\vec{u})| \le |\vec{v} - \vec{u}|$.*

Proof. Let $perm(\vec{v}) = [v_{\sigma(1)}, \ldots, v_{\sigma(n)}]$, where σ represents the sorted permutation of $\vec{v}$.

1. Let us compare u_j and $v_{\sigma(j)}$. There are two cases to consider. If $j \le \sigma(j)$, then $u_j \le u_{\sigma(j)} < v_{\sigma(j)}$. These inequalities hold since $\vec{u}$ is sorted by assumption and $\vec{u} < \vec{v}$. If $j > \sigma(j)$, then there exists $j' \ge j$ such that $v_{j'} \le v_{\sigma(j)}$. (Otherwise, for all $j' \ge j$, $v_{j'} > v_{\sigma(j)}$. However, only $n - j$ components of $\vec{v}$ can be greater than $v_{\sigma(j)}$.) Combining the fact that $\vec{u}$ is sorted and $\vec{u} < \vec{v}$, we have $u_j \le u_{j'} < v_{j'} \le v_{\sigma(j)}$. Therefore, $perm(\vec{u}) < perm(\vec{v})$ in both cases.
2. Since $perm(\vec{u} + \vec{z}) = perm(\vec{u}) + \vec{z}$ for $\vec{z} = [z, \ldots, z]$, property 1 implies $perm(\vec{v}) < perm(\vec{u} + \vec{z}) = perm(\vec{u}) + \vec{z}$.
3. Suppose $|perm(\vec{v}) - perm(\vec{u})| = v_{\sigma(j)} - u_j$. There are two cases to consider. If $v_{\sigma(j)} \le v_j$, then $v_{\sigma(j)} - u_j \le v_j - u_j$, which implies $|perm(\vec{v}) - perm(\vec{u})| \le |\vec{v} - \vec{u}|$. If $v_{\sigma(j)} > v_j$, then there exists $j' < j$ such that $v_{\sigma(j)} \le v_{j'}$. (Otherwise, for all $j' \le j$, $v_{\sigma(j)} > v_{j'}$. However, only $j - 1$ components of $\vec{v}$ can be smaller than $v_{\sigma(j)}$.) Since $u_{j'} < u_j$ by the assumption that $\vec{u}$ is sorted, we have $v_{\sigma(j)} - u_j \le v_{j'} - u_{j'}$, which implies $|perm(\vec{v}) - perm(\vec{u})| \le |\vec{v} - \vec{u}|$. Property 3 follows. ∎

We are ready to transform a token-sequence-based bound into a packet-based bound. Although it might seem straightforward, the difficulty is that a packet is unable to identify a token sequence. This means if a session-i token appears, then the session-i packet that has been waiting the longest has to move. The first token in a

token sequence is called an *initial token.*

THEOREM 2.2. *Consider any template-based schedule. If the end-to-end delay for each session-i token sequence is bounded by X, then each session-i packet reaches its destination within X steps after it obtains an initial token. If the token lag is bounded by x for* all *token sequences for* all *sessions, then the edge queue size is also bounded x.*

Proof. It suffices to show the following. For any $y \geq 1$, consider the first y session-i packets injected. After obtaining its initial token, each of these y packets reaches the destination within X steps, and it waits at most x steps to advance each edge.

Let T_{k1} be the time that the kth packet catches an initial token $\mathcal{K}_k$ and advances its first edge. Let T_{kj} be the time that the kth packet would cross the jth edge if it followed the token sequence initiated at $\mathcal{K}_k$. Note that T_{kj} is not necessarily the time that the kth packet crosses the jth edge in a template-based schedule. However, T_{kj} does represent the time that a token would appear on the jth edge. We have $T_{11} < T_{21} < \cdots < T_{y1}$, and $T_{k1} < T_{k2} < \cdots < T_{kd_1}$ for $1 \leq k \leq y$.

We first apply property 1 of Lemma 2.1 to show that packets 1 through y are able to cross the jth edge by times $perm(T_{1j}, T_{2j}, \ldots, T_{yj})$ for $1 \leq j \leq d_i$. Take an example of the second edge. Let $perm(T_{12}, T_{22}, \ldots, T_{y2}) = [T_{\sigma(1),2}, T_{\sigma(2),2}, \ldots, T_{\sigma(y),2}]$, where σ represents the sorted permutation. Property 1 of Lemma 2.1 implies

$$[T_{11}, T_{21}, \ldots, T_{y1}] < [T_{\sigma(1),2}, T_{\sigma(2),2}, \ldots, T_{\sigma(y),2}].$$

Since packet 1 has left its first edge by time T_{11} and an unused token for the second edge appears by $T_{\sigma(1),2}$, packet 1 is able to advance its second edge by $T_{\sigma(1),2}$. Since packet 1 has left by $T_{\sigma(1),2}$, packet 2 is able to obtain an unused token by $T_{\sigma(2),2}$ and advance its second edge. Similar reasoning applies to packets 3 through y for the second edge. Inductively, packets 1 through y are able to advance their last edge by $perm(T_{1d_i}, T_{2d_i}, \ldots, T_{yd_i})$. This quantity is bounded by $[T_{11}+X, T_{21}+X, \ldots, T_{y1}+X]$ by property 2 of Lemma 2.1. Hence, all the session-i packets reach their destination within X steps after they obtain the initial tokens.

Let us bound the queue size now. Consider the jth edge, where $1 \leq j \leq d_i$. Suppose packet k, for $1 \leq k \leq y$, uses token $\mathcal{K}_{kj}$ to cross its jth edge at time t_{kj}. Let $\mathcal{K}_{k,j+1}$ be the $(j+1)$st token on the same token sequence as $\mathcal{K}_{kj}$, and let $t_{k,j+1}$ be the time that $\mathcal{K}_{k,j+1}$ appears. (Note that $\mathcal{K}_{kj}$ is not necessarily on the same token sequence as the initial token that packet k used to cross its first edge, and that $\mathcal{K}_{k,j+1}$ is not necessarily the token that packet k would use to cross the $(j+1)$st edge.) Since $t_{kj} < t_{k,j+1}$, property 1 of Lemma 2.1 and our argument for the delay bound above imply that packets 1 through y are able to cross the $(j+1)$st edge by $perm(t_{1,j+1}, t_{2,j+1}, \ldots, t_{y,j+1})$. Property 3 of Lemma 2.1 shows that $|perm(t_{1,j+1}, t_{2,j+1}, \ldots, t_{y,j+1}) - [t_{1j}, t_{2j}, \ldots, t_{yj}]|$ is bounded by x, the token lag. Hence, a packet waits at most x steps to advance each edge once it obtains an initial token. □

2.2. Lemmas for probabilistic analysis. Throughout the construction of our schedules, we use the Lovász local lemma [17, pp. 57–58] and a Chernoff bound [5] for probabilistic analysis. We include them here for easy reference.

Lovász local lemma. *Let $E_1, \ldots, E_n$ be a set of "bad events," each occurring with probability at most p and with dependence at most d (i.e., every bad event is*

mutually independent of some set of $n-d$ other bad events). If $4pd < 1$, then

$$\Pr\left[\bigcap_{i=1}^{n} \bar{E}_i\right] > 0.$$

In other words, no bad event occurs with a nonzero probability.

Chernoff bound. *Let X_i be n independent Bernoulli random variables with probability of success p_i. Let $X = \sum_{i=1}^{n} X_i$, and let the expectation $\mu = \sum_{i=1}^{n} p_i$. Then for $0 < \delta < 1$, we have*

$$\Pr[\ X > (1+\delta)\mu\ ] \le e^{-\delta^2\mu/3}.$$

We also prove a variation of the Chernoff bound.

LEMMA 2.3. *Let X_i be n independent Bernoulli random variables with probability of success p_i. Let $X = \sum_i^n X_i$ and the expectation $E[X] = \sum_i^n p_i$. Then for $u \ge E[X]$ and $0 < \delta < 1$, we have*

$$\Pr[\ X > (1+\delta)u\ ] \le e^{-\delta^2 u/3}.$$

Proof. We prove the lemma by amplifying the success probabilities. If $u \ge n$, then $\Pr[\ X \ge (1+\delta)u\ ] = 0$ and we are done. Otherwise, let p_i' be a value such that $p_i \le p_i' \le 1$ and $\sum_i^n p_i' = u$. We have

$$\begin{aligned}&\Pr[\ X > (1+\delta)u \mid \text{success probabilities } p_1, \ldots, p_n\ ]\\ &\le \Pr[\ X > (1+\delta)u \mid \text{success probabilities } p_1', \ldots, p_n'\ ].\end{aligned}$$

The Chernoff bound implies that the above probability is bounded by $e^{-\delta^2 u/3}$. ∎

3. Suboptimal schedules. We present in this section a simple randomized distributed scheduler that, with high probability, produces a delay bound of

$$O\left(\frac{1}{r_i} + d_i \log \frac{m}{r_{\min}}\right)$$

and edge queues of size $O(\log \frac{m}{r_{\min}})$, where m is the number of edges and $r_{\min} = \min_i r_i$. This preliminary result is substantially simpler to prove than the optimal result of $O(1/r_i + d_i)$ because of the relaxed bounds. Nevertheless, it illustrates the basic ideas necessary to prove the main result. We begin with a *centralized* scheme in section 3.1 that achieves these bounds, and then we convert it to a *distributed* scheme in section 3.2.

3.1. A simple centralized scheduler. As stated above, we now present a centralized scheduler that achieves the desired delay bound of $O(\frac{1}{r_i} + d_i \log \frac{m}{r_{\min}})$ with edge queues of size $O(\log \frac{m}{r_{\min}})$.

The structure of the proof is as follows. Each session-i packet must traverse d_i edges. We prove that the time the packet spends waiting for a token at each edge along the path (after the first edge) is $O(\log \frac{m}{r_{\min}})$. Hence the time to traverse all edges (but the first) is $O(d_i \log \frac{m}{r_{\min}})$. It turns out that the time spent waiting to receive the very first token (what we refer to as *initial* waiting time) is $O(\frac{1}{r_i} + d_i \log \frac{m}{r_{\min}})$. Hence the result follows.

The difficulty is to come up with a placement of tokens that achieves the above bounds. To do this, we will first come up with an *illegal* placement of tokens, where we place more than one token in some slots and zero tokens in other slots. We will prove delay bounds on the illegal placement. We will then apply a *smoothing* procedure which "smooths" out the bumps in the illegal placement, making it legal. We will prove that the smoothing process does not change the bounds very much.

Template size. We first decide the template size $\mathcal{T}$. Roughly speaking, $\mathcal{T}$ needs to be sufficiently large so that enough tokens can be placed to accommodate arrivals from all sessions every $\mathcal{T}$ steps. We express the injection rate for session i in terms of $\hat{r}_i = s_i/\ell_i$, a fraction slightly larger than r_i. If $\mathcal{T}$ is the least common multiple of ℓ_i for all i, then we can place s_i session-i tokens every ℓ_i consecutive slots on each template along the path of session i. The quantities of ℓ_i and s_i are defined as follows:

$$\ell_i = 2^{\lceil \log \frac{2}{\varepsilon r_i} \rceil}, \tag{3.1}$$

$$s_i = \lfloor \ell_i r_i (1 + \varepsilon/2) \rfloor, \tag{3.2}$$

$$\hat{r}_i = s_i/\ell_i, \tag{3.3}$$

where ε is a constant to ensure that the sum of the rates of the sessions using any edge is less than 1. In other words, ℓ_i is the smallest power of 2 that is larger than or equal to $2/(\varepsilon r_i)$, and s_i is the largest integer that is less than or equal to $\ell_i r_i(1+\varepsilon/2)$. The template size $\mathcal{T}$ is the least common multiple of ℓ_i. Since all the ℓ_i's are powers of 2, $\mathcal{T} = O(1/r_{\min})$.

LEMMA 3.1. *We have the following properties for $\hat{r}_i$.*

1. *$r_i \le \hat{r}_i \le r_i(1+\varepsilon/2)$ for each session i.*
2. *$\sum_{i \in S_e} \hat{r}_i \le 1 - \varepsilon/2$ for each edge e, where S_e is the set of sessions that cross edge e.*

Proof. Property 1 is equivalent to

$$\ell_i r_i \le s_i \le \ell_i r_i (1 + \varepsilon/2).$$

The difference between the lower bound and the upper bound is $\ell_i r_i \varepsilon/2$, which is at least 1 by the definition of ℓ_i. Therefore, there exists an integer in the range of $[\ell_i r_i, \ell_i r_i(1+\varepsilon/2)]$, and s_i is such an integer by definition. Property 1 follows.

Given $\sum_{i \in S_e} r_i \le 1 - \varepsilon$, we have $\sum_{i \in S_e} \hat{r}_i \le (1-\varepsilon)(1+\varepsilon/2) < 1 - \varepsilon/2$. Property 2 follows. ∎

We now define the template size $\mathcal{T}$ to be $\max_i \ell_i$, which is $\Theta(\frac{1}{r_{\min}})$. Since all the ℓ_i's are powers of 2, $\mathcal{T}$ is also the least common multiple of the ℓ_i's.

Token placement. We describe a procedure to place the tokens for all sessions. We start with an *illegal placement* of tokens. For each session i, we first place s_i *initial tokens* in one slot every ℓ_i slots on the template that corresponds to the first edge of session i. We then delay each initial token of session i by an amount chosen uniformly and independently at random from $[L+1, L+\ell_i]$, where

$$L = 2^{\lceil \log\left(\frac{\alpha}{2} \log(m\mathcal{T})\right) \rceil} \tag{3.4}$$

for a constant α. In other words, L is a power of 2 that is greater than or equal to $\frac{\alpha}{2}\log(m\mathcal{T})$. As we shall see, this is enough randomness to spread out the tokens. For every session-i token a placed on the template corresponding to the jth edge, we place a session-i token b on the template corresponding to the $(j+1)$st edge such that b appears exactly $2L$ steps after a. In this way, we have partitioned all the session-i tokens into $\mathcal{T}\hat{r}_i$ sequences, where each token sequence has d_i tokens and two neighboring tokens in each sequence are $2L$ apart. In the following we show that the tokens cannot be too clustered.

LEMMA 3.2. *At most L tokens appear in any consecutive L slots on any template with probability at least $1 - 1/(m\mathcal{T})$, where L is defined in* (3.4) *for a sufficiently large constant α.*

Proof. Since s_i initial tokens for session i are placed in one slot every ℓ_i slots and each is delayed by an amount chosen independently and uniformly at random from $[L+1, L+\ell_i]$, the expected number of session-i tokens in a single slot is s_i/ℓ_i, which is $\hat{r}_i$. Hence by linearity of expectations and property 2 of Lemma 3.1, the expected number of tokens over all sessions in L consecutive slots is $\sum_i \hat{r}_i L \le (1-\varepsilon/2)L$. For a particular interval of L consecutive slots on a particular template, let the random variable X be the number of tokens in these slots. Whether or not a token lands in these L slots is a Bernoulli event. Since the delays to the initial tokens are chosen independently and all session paths are simple, these Bernoulli events are independent. Since $E[X] \le (1-\varepsilon/2)L$, we have the following by Lemma 2.3.

$$\Pr[\ X > L\ ] \le \Pr[\ X > (1+\varepsilon/2)(1-\varepsilon/2)L\ ] \le e^{-\varepsilon^2(1-\varepsilon/2)L/12}.$$

In m templates there are at most $m\mathcal{T}$ intervals of L consecutive slots. Therefore, by a union bound the probability that more than L tokens appear in *any* L consecutive slots is bounded by

$$m\mathcal{T} \Pr[\ X > L\ ] \le m\mathcal{T} e^{-\varepsilon^2(1-\varepsilon/2)L/12} = m\mathcal{T} e^{-\varepsilon^2(1-\varepsilon/2)\alpha \log(m\mathcal{T})/24}.$$

By choosing a sufficiently large constant α, we can bound the above probability by $1/(m\mathcal{T})$. □

If the first pass of the delay insertion does not produce a token assignment that satisfies the condition of at most L tokens every L slots, we simply try another pass until the condition is met.

Smoothing. In order to guarantee one token per slot, we carry out a *smoothing process*. Since there are at most L tokens in any consecutive L slots, we partition each template into intervals of L consecutive slots and arbitrarily place at most one token in each slot within each interval. (Note the template size $\mathcal{T}$ is a multiple of L since $\mathcal{T}$ and L are both powers of 2.) Recall we have defined a token sequence for each session in the token placement process.

LEMMA 3.3. *Let $\mathcal{K}_1, \ldots, \mathcal{K}_{d_i}$ be any token sequence for session i; then, after the smoothing process, we have the following.*

1. *Token $\mathcal{K}_j$ appears after $\mathcal{K}_{j-1}$ for $1 < j \le d_i$.*
2. *The end-to-end delay of the token sequence is bounded by $2d_iL + 2L$, and the token lag is bounded by $4L$.*

Proof. Before the smoothing, $\mathcal{K}_j$ appears exactly $2L$ steps after $\mathcal{K}_{j-1}$ for $1 < j \le d_i$, i.e., the token lag is $2L$. Since the smoothing process shifts each token by at most $L-1$ slots, $\mathcal{K}_j$ still appears after $\mathcal{K}_{j-1}$ after the smoothing. The token lag therefore increases to at most $4L$. The end-to-end delay for the token sequence increases from $2d_iL$ to at most $2d_iL + 2L$ due to the shift of the first and the last tokens. □

THEOREM 3.4. *With high probability, the above randomized centralized scheme generates a template-based schedule that produces a delay bound of $O(\frac{1}{r_i} + d_i \log \frac{m}{r_{\min}})$ and edge queues of size $O(\log \frac{m}{r_{\min}})$.*

Proof. We first show that each session-i packet, p, is able to catch an initial token within $2L + 2\ell_i$ steps of its injection. Before the initial session-i tokens are delayed, we have exactly s_i tokens every ℓ_i slots. Since at most s_i session-i packets can be injected during ℓ_i steps, packet p would be able to obtain an initial token, say $\mathcal{K}$, in fewer than ℓ_i steps if the tokens were not delayed or shifted. Let p be injected at time t, and let $\mathcal{K}$ appear at T before $\mathcal{K}$ is delayed and shifted; then $t \le T < t + \ell_i$. Each initial token is delayed by an amount in the range of $[L+1, L+\ell_i]$ during the token

placement process and is shifted by at most $L-1$ slots during the smoothing process. Therefore, after the smoothing process, $\mathcal{K}$ appears after t but before $t+2L+2\ell_i$.

By Theorem 2.2 and Lemma 3.3, any session-i packet p is able to reach its destination within $2d_iL+2L$ steps after it obtains its initial token. Therefore, the end-to-end delay for session-i packets is $(2L+2\ell_i)+(2d_iL+2L)$, which is $O(\frac{1}{r_i}+d_i\log\frac{m}{r_{\min}})$. The edge queue size is bounded by the token lag $4L$, which is $O(\log\frac{m}{r_{\min}})$. □

3.2. A simple distributed scheduler. The above scheme is centralized since the session-i tokens on one template are dependent on the previous template. However, it suggests the following simple *distributed* scheme for scheduling packets so as to achieve small delay. As in section 3.1, we place initial tokens on the first edge of session i; however, this time we delay each initial token by an amount chosen independently and uniformly at random from $[1,\ell_i]$, where ℓ_i is defined in (3.1). (Note that the delay is from $[L+1,L+\ell_i]$ in the centralized scheme.) Suppose that a session-i packet p now obtains its initial token at time T. Then for the jth edge on the session-i path, p is given a deadline of $T+2L(j-1)+L$, where L is defined in (3.4). Whenever two or more packets contend for the same edge simultaneously, the packet with the earliest deadline moves. Ties are broken arbitrarily. We call this scheme Earliest-Deadline-First (EDF). Note that EDF is no longer template based. We show in Lemma 3.5 that the deadlines do not cluster together with high probability, and we show in Lemma 3.6 that every packet meets its deadlines.

Lemma 3.5. *For any edge, at most L deadlines appear in any consecutive L time steps with probability at least $1-1/(m\mathcal{T})$, where L is defined in* (3.4) *for a sufficiently large constant α.*

Proof. The deadlines for a packet p are $T+L$, $T+3L$, $T+5L$, ..., which correspond to the times that the tokens in a sequence appear. Hence, the proof is identical to that of Lemma 3.2. □

Lemma 3.6. *If for any edge at most L deadlines appear in any consecutive L time steps, then each packet crosses every edge by its deadline by* EDF.

Proof. For the purpose of contradiction, let D be the first deadline that is missed. This implies all deadlines earlier than D are met. Let p be the packet that misses deadline D for edge e. Since packet p makes its previous deadlines, p must have crossed its previous edge by time $D-2L$, or else e must be p's first edge and p must have obtained its initial token at time $D-L$. Hence, at every time step from time $D-L+1$ to D, packet p is held up by another packet with a deadline no later than D. Furthermore, these deadlines must be later than $D-L$ since all deadlines earlier than D are met. Therefore, at least $L+1$ packets have deadlines for edge e from time $D-L+1$ to D. Our lemma follows from the contradiction. □

By an argument similar to that in Theorem 3.4, a session-i packet obtains its initial token within $2\ell_i$ steps of its injection. Combined with Lemmas 3.5 and 3.6, we have the following theorem.

Theorem 3.7. *With high probability, the randomized distributed scheme* EDF *generates a schedule that produces an end-to-end delay bound of $O(\frac{1}{r_i}+d_i\log\frac{m}{r_{\min}})$.*

In [2], simulations were carried out to compare the end-to-end delays produced by our EDF scheme against those produced by WFQ. The former outperformed the latter in a range of simulations.

4. Overview of the main result. Our main result for the dynamic routing problem parallels an earlier result on static routing. In section 4.1 we review the method used for solving the static case, and in section 4.2 we give an overview of the additional complexities that need to be addressed in the dynamic case.

TABLE 4.1
Frame-refinement for static routing in [10].

Schedule	Frame size	Relative congestion
$\mathcal{S}^{(q)}$	$I^{(q)}$	$c^{(q)}$
Refinement	$\log^5 I^{(q)}$	$(1+o(1))c^{(q)}$
$\mathcal{S}^{(q+1)}$	$I^{(q+1)}$	$c^{(q+1)}$

4.1. A bound of $O(c+d)$ for static routing. Leighton, Maggs, and Rao consider the static routing problem for arbitrary networks in [10]. For static routing, all packets are present in the network initially. Each packet is associated with a source, a destination, and a route. The *congestion* on each edge is the total number of routes that require that edge, and the *dilation* of a route is the number of edges on the route. Leighton, Maggs, and Rao show that for any set of routes with maximum congestion c (over all edges) and maximum dilation d (over all routes), there is a schedule of length $O(c+d)$ and edge queue size $O(1)$. In this schedule, at most one packet traverses each edge at each time step. A packet waits $O(c+d)$ steps initially before leaving its source, and it waits $O(1)$ steps to cross each edge thereafter.

We summarize here the techniques in [10]. The strategy for constructing an efficient schedule is to make a succession of *refinements* to an initial schedule $\mathcal{S}^{(0)}$. In $\mathcal{S}^{(0)}$, each packet moves at every step until it reaches its destination. This schedule has length d, but as many as c packets may traverse the same edge at the same step. Each refinement brings the schedule closer and closer to the requirement that at most one packet uses one edge per time step.

A *T-frame* is a time interval of length T. The *frame congestion*, C, in a T-frame is the largest number of packets that use any edge during the frame. The *relative congestion* in a T-frame is the ratio C/T. The frame congestion (resp., relative congestion) on an edge e during a T-frame is defined to be the frame congestion (resp., relative congestion) associated with edge e.

It is obvious that the initial schedule $\mathcal{S}^{(0)}$ has relative congestion at most 1 for any c-frame. A *refinement* transforms a schedule $\mathcal{S}^{(q)}$ with relative congestion at most $c^{(q)}$ in any frame of size $I^{(q)}$ or larger into a schedule $\mathcal{S}^{(q+1)}$ with relative congestion at most $c^{(q+1)}$ in any frame of size $I^{(q+1)}$ or larger. The resulting frame size $I^{(q+1)}$ is much smaller than $I^{(q)}$, whereas the relative congestion $c^{(q+1)}$ is only slightly bigger than $c^{(q)}$. In particular, $I^{(q+1)} = \log^5 I^{(q)}$ and $c^{(q+1)} = (1+o(1))c^{(q)}$. After a series of $O(\log^* c)$ refinements, a schedule $\mathcal{S}^{(\zeta)}$ is obtained, where the relative congestion is $O(1)$ for any $O(1)$-frame. A final schedule, in which at most one packet at a time crosses each edge, can be constructed by replacing each step of $\mathcal{S}^{(\zeta)}$ by a constant number of steps. Each refinement is achieved by inserting delays to the packets. It is the central issue in [10] to show that a set of delays always exists satisfying the criteria in Table 4.1.

4.2. A bound of $O(1/r_i + d_i)$ for dynamic routing. Our result for the dynamic routing problem is parallel to that in [10]. For an arbitrary network where paths (sessions) are defined, we show that there is a schedule such that every session-i packet reaches its destination within $O(1/r_i + d_i)$ steps of its injection, where r_i and d_i are the injection rate and path length for session i, respectively. A session-i packet waits $O(1/r_i + d_i)$ steps initially before leaving its source, and it waits $O(1)$ steps to cross each edge afterwards.

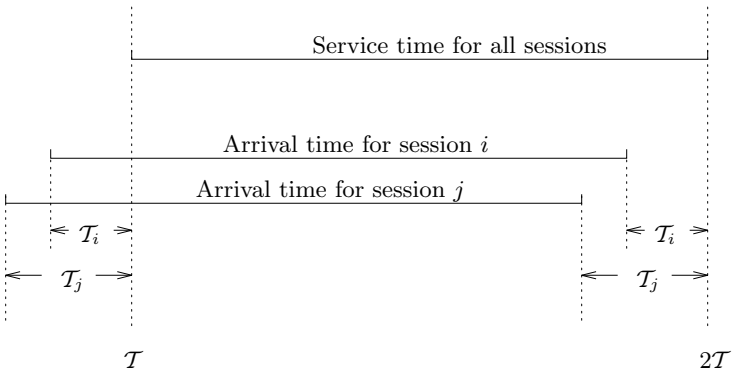


FIG. 4.1. *All the session-i packets that arrive during* $[k\mathcal{T}-\mathcal{T}_i,(k+1)\mathcal{T}-\mathcal{T}_i)$ *are serviced during* $[k\mathcal{T},(k+1)\mathcal{T})$. *In this figure,* $k=1$.

To achieve a session-based, end-to-end delay bound of $O(1/r_i+d_i)$ for our dynamic routing problem, we adopt the general approach in [10]. However, there are three major problems in transforming the solution for the static problem into a solution for the dynamic problem. In the remainder of this section we present these three problems and their solutions.

In the remainder of the paper we use the language of "scheduling packets" rather than "placing tokens." At the end of the presentation we show how to transform the packet schedule into a template-based schedule. Although the actual packet arrivals are not be periodic, the times at which the packets cross the first edge *are* periodic. This is the key to the transformation.

Problem 1: Infinite time. In [10] all the packets to be scheduled are present initially. In the dynamic model, packets are injected over an infinite time line. We would like to partition the infinite time line into finite time intervals which can be scheduled independently of each other. We divide time into intervals of length $\mathcal{T}$, where $\mathcal{T}=\Theta(1/r_{\min}+d_{\max})$. We then independently schedule the time intervals $[0,\mathcal{T})$, $[\mathcal{T},2\mathcal{T})$, $[2\mathcal{T},3\mathcal{T})$, etc.

We associate each session i with a quantity $\mathcal{T}_i=\Theta(1/r_i+d_i)$. For any integer $k\geq 0$ consider all the session-i packets that are injected during interval $[k\mathcal{T}-\mathcal{T}_i,(k+1)\mathcal{T}-\mathcal{T}_i)$. We provide a schedule in which all these packets leave their sources no earlier than time $k\mathcal{T}$ and reach their destinations before time $(k+1)\mathcal{T}$. (See Figure 4.1.) From now on, we concentrate on scheduling the arrivals that would be serviced during interval $[\mathcal{T},2\mathcal{T})$.

The quantity $\mathcal{T}$ will also serve as the size of all templates in the template-based schedule.

Problem 2: Session-based delay guarantees. Once we restrict ourselves to the interval $[\mathcal{T},2\mathcal{T})$, it seems that the dynamic routing problem is similar to the static problem. However, we cannot simply proceed with the successive refinements as in section 4.1, since some sessions need tighter delay bounds than others. Session-i packets can only tolerate a delay proportional to $1/r_i+d_i$. We group sessions according to their associated $1/r_i+d_i$ value. We start by inserting delays to sessions having large values of $1/r_i+d_i$, reducing the frame size, and bounding the relative congestion. When the frame size becomes small enough, sessions with smaller $1/r_i+d_i$ join in.

TABLE 4.2
Refinement and conversion for dynamic routing.

Schedule	Integral sessions	Frame size	Relative congestion
$\mathcal{S}^{(q)}$	$A^{(q)}$	$I^{(q)}$	$c^{(q)}$
Refinement	$A^{(q)}$	$\log^5 I^{(q)}$	$(1+o(1))c^{(q)}$
Conversion	$A^{(q)} \cup B^{(q+1)}$	$\log^5 I^{(q)}$	$(1+o(1))^2 c^{(q)}$
$\mathcal{S}^{(q+1)}$	$A^{(q+1)}$	$I^{(q+1)}$	$c^{(q+1)}$

More precisely, we introduce the concept of *integral* and *fractional* sessions. When session i is *integral*, packets of size 1 are injected at rate r_i. When session i is *fractional*, a packet of size $\hat{r}_i$ is injected at every time step, where $\hat{r}_i$ is a value slightly larger than r_i. A packet from a fractional session always crosses one edge at a time, whether or not other packets are crossing the edge at the same time. Therefore, a fractional packet from session i always contributes exactly $\hat{r}_i$ to the congestion. Integral sessions are those to which we can afford to insert delays in order to bound the congestion. Fractional sessions are those to which we cannot insert delays. However, congestion due to a fractional session i is only $\hat{r}_i$, which is small.

As before, $\mathcal{S}^{(q)}$ represents the schedule in the qth iteration. The set of integral sessions for $\mathcal{S}^{(q)}$ is denoted by $A^{(q)}$. For the initial schedule $\mathcal{S}^{(0)}$, all the sessions are fractional and we show that the relative congestion is less than 1. For schedule $\mathcal{S}^{(q)}$ we inductively assume that the relative congestion due to the current integral and fractional sessions is at most $c^{(q)}$ for any frame of size $I^{(q)}$ or larger. To create a schedule $\mathcal{S}^{(q+1)}$ from schedule $\mathcal{S}^{(q)}$ we carry out a frame-refinement step and a conversion step.

The frame-refinement step reduces the frame size from $I^{(q)}$ to $I^{(q+1)} = \log^5 I^{(q)}$, while slightly increasing the relative congestion from $c^{(q)}$ to $(1+o(1))c^{(q)}$. This step is achieved by delaying the integral packets by up to $\Theta\left((I^{(q)})^2\right)$ steps. We make sure that if session i is in $A^{(q)}$, then $1/r_i + d_i \geq (I^{(q)})^2$, and therefore the delays inserted can be tolerated. The conversion step converts some sessions from fractional to integral, while maintaining the frame size of $I^{(q+1)}$ and slightly increasing the relative congestion to $c^{(q+1)} = (1+o(1))^2 c^{(q)}$. These newly-converted sessions form a set $B^{(q+1)}$ and have associated values $1/r_i + d_i \geq (I^{(q+1)})^2$. This bound is chosen so that the sessions in $A^{(q+1)}$, which is $A^{(q)} \cup B^{(q+1)}$, will be able to tolerate the delays inserted during the next iteration of frame refinement. During the conversion step we delay the packets in $B^{(q+1)}$ by up to $\Theta(1/r_i + d_i)$ steps. We are able to show the existence of "good" delays for both frame refinement and conversion steps. Table 4.2 summarizes our approach.

At the termination of our algorithm we have a schedule $\mathcal{S}^{(\zeta)}$ in which every session is integral and the relative congestion is at most 1 for all frames of size larger than a certain constant. In $\mathcal{S}^{(\zeta)}$ all session-i arrivals during $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ are serviced during $[\mathcal{T}, 2\mathcal{T})$. Furthermore, all session-i packets reach their destination within $O(\mathcal{T}_i)$ steps of their injections.

Problem 3: Constant-factor stretching in the final schedule. As discussed above, we repeat the process of refinement and conversion until we have a schedule, $\mathcal{S}^{(\zeta)}$, in which all sessions are integral and in which the relative congestion is 1 for all frames of size larger than a certain constant w. In the static problem, a final schedule can easily be obtained by stretching $\mathcal{S}^{(\zeta)}$ by a constant factor. However, we cannot

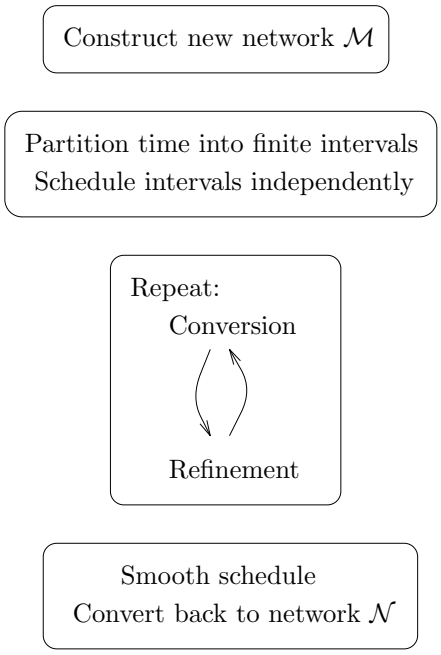


FIG. 4.2. *An overview of our approach for the dynamic routing problem.*

afford to have a constant blowup in our final schedule for the dynamic problem. This is because we need to independently schedule all time intervals $[0, \mathcal{T})$, $[\mathcal{T}, 2\mathcal{T})$, etc., and a constant blowup would make these time intervals overlap.

To overcome this problem, we first devise a schedule for a new network $\mathcal{M}$ that is constructed from the original network $\mathcal{N}$ as follows. Each edge e of $\mathcal{N}$ is replaced by $2w$ consecutive edges $e_1, \ldots, e_{2w}$, where w is the constant introduced above. The rates and routes of the sessions are unaffected. In $\mathcal{M}$, session i has length $D_i = 2wd_i = O(d_i)$.

All the techniques described earlier are applied to the network $\mathcal{M}$. We carry out successive conversion and refinement steps for $\mathcal{M}$ and obtain a schedule $\mathcal{S}^{(\zeta)}$, where the relative congestion is 1 for any frame whose size is larger than w. We then "smooth" $\mathcal{S}^{(\zeta)}$ and convert it to a schedule for $\mathcal{N}$ where only one packet at a time traverses any edge.

The idea behind the smoothing process is as follows. In $\mathcal{S}^{(\zeta)}$, more than one packet may require some edge of $\mathcal{M}$ during a given time step, but at most w packets can require any given edge f in $\mathcal{M}$ within w time steps. This means we can shuffle each packet that requires edge f by at most w time steps, so that exactly one packet traverses f at any step. Unfortunately, this shuffling in time can lead to an illegal schedule for $\mathcal{M}$, in which a packet can be scheduled to traverse the edges on its path out of order (timewise). However, one can prove that if we consider the schedule with respect to the packets traversing edge e_{2w} for all e, then this schedule *is* legal, i.e., the packets cross *these* edges in order. Hence, we schedule edge e in $\mathcal{N}$ in exactly the same way that the corresponding edge e_{2w} is scheduled in $\mathcal{M}$.

Figure 4.2 is a schematic picture of our overall approach.

5. Parameter definitions.

Interval length $\mathcal{T}$ and $\mathcal{T}_i$. As discussed in section 4.2, we independently schedule intervals $[0, \mathcal{T})$, $[\mathcal{T}, 2\mathcal{T})$, etc. Our proof will concentrate on the interval $[\mathcal{T}, 2\mathcal{T})$. All the session-$i$ packets that arrive during $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ are serviced during $[\mathcal{T}, 2\mathcal{T})$. We define $\mathcal{T}$ and $\mathcal{T}_i$ for session i as follows. Recall $D_i = 2wd_i$, where w is a constant defined at the end of this section.

$$\mathcal{T}_i = 4D_i + 2 + (8/\varepsilon + 2)/r_i,$$
$$\mathcal{T} = \left\lceil \frac{(1 + 4/\varepsilon) \max_i \mathcal{T}_i}{w} \right\rceil w.$$

In other words, $\mathcal{T}$ is the smallest multiple of w that is greater than or equal to $(1 + 4/\varepsilon) \max_i \mathcal{T}_i$. Clearly $\mathcal{T}_i = O(1/r_i + D_i) = O(1/r_i + d_i)$.

In the template-based schedule, all template sizes will be $\mathcal{T}$.

Packet size for a fractional session. In this section we define $\hat{r}_i$, the packet size for a fractional session i. For reasons that will become clear in the conversion step of section 6.3, we need $\hat{r}_i$, to be slightly larger than r_i, and we shall need to express $\hat{r}_i$ as the ratio of two integers. Let

$$\ell_i = \lceil 8/(\varepsilon r_i) \rceil,$$
$$s_i = \lfloor \ell_i r_i (1 + \varepsilon/2) \rfloor,$$
$$\hat{r}_i = s_i/\ell_i.$$

The following lemma is analogous to Lemma 3.1.

LEMMA 5.1. *We have the following properties for $\hat{r}_i$.*

1. *$r_i(1 + \varepsilon/4) \le \hat{r}_i \le r_i(1 + \varepsilon/2)$ for each session i.*
2. *$\sum_{i \in S_e} \hat{r}_i \le 1 - \varepsilon/2$ for each edge e, where S_e is the set of sessions that cross edge e.*

Note that the definition of ℓ_i and property 1 of Lemma 5.1 are different from the ones in section 3.1. We need this stronger lower bound on $\hat{r}_i$ to handle the extra complexity in the conversion step. In particular, $\hat{r}_i$ is also used to indicate the rate at which the initial tokens for session i appear. During the conversion step, the initial tokens for session i are placed in the interval $[\mathcal{T}, 2\mathcal{T} - \mathcal{T}_i)$. Since these tokens are to accommodate all the session-i arrivals during $[\mathcal{T}, 2\mathcal{T})$, we need $\hat{r}_i(\mathcal{T} - \mathcal{T}_i) \ge r_i\mathcal{T}$. This condition is guaranteed by the choices of $\mathcal{T}$ and $\mathcal{T}_i$ and property 1 of Lemma 5.1. (See Lemma 6.8.)

Parameters for schedule $\mathcal{S}^{(q)}$. We shall show later that, in schedule $\mathcal{S}^{(q)}$, the relative congestion, due to all integral and fractional sessions, is at most $c^{(q)}$ for any frame of size $I^{(q)}$ or larger. For $\mathcal{S}^{(q)}$, the set $A^{(q)}$ consists of all the integral sessions. As we construct schedule $\mathcal{S}^{(q+1)}$ from $\mathcal{S}^{(q)}$, sessions in $B^{(q+1)}$ become integral and join $A^{(q)}$. The schedule at the end of the refinement and the conversion is $\mathcal{S}^{(\zeta)}$. The parameters $I^{(q)}$, $c^{(q)}$, $A^{(q)}$, and $B^{(q+1)}$ are defined by the following recurrences. Let $X_i = D_i + 1/r_i$ for session i, and let $X_{\max} = \max_i X_i$.

$$I^{(0)} = e^{\log^{2/5} X_{\max}},$$
$$I^{(q+1)} = \log^5 I^{(q)},$$
$$c^{(0)} = 1 - \varepsilon/2,$$
$$c^{(q+1)} = (1 + \delta^{(q)})^2 c^{(q)},$$
$$\delta^{(q)} = \beta/\sqrt{\log I^{(q)}},$$

$$
\begin{aligned}
A^{(0)} &= \emptyset, \\
A^{(q+1)} &= A^{(q)} \cup B^{(q+1)}, \\
B^{(q+1)} &= \left\{ i \notin A^{(q)} : \left(I^{(q+1)}\right)^2 \le X_i \le e^{\sqrt{I^{(q+1)}}} \right\} \qquad \text{for } q \neq \zeta - 1, \\
B^{(q+1)} &= \left\{ i \notin A^{(q)} : X_i \le e^{\sqrt{I^{(q+1)}}} \right\} \qquad \text{for } q = \zeta - 1.
\end{aligned}
$$

The parameter β is a sufficiently large positive constant. Note that $I^{(q)}$ decreases polylogarithmically and $c^{(q)}$ increases by a factor of $1+o(1)$. One can verify that $B^{(q)}$ forms a partition of all the sessions and that sessions with large X_i values become integral first. We make use of the bound $X_i \ge \left(I^{(q+1)}\right)^2$ in the frame refinement step, and we use the bound $\log^2 X_i \le I^{(q+1)}$ in the conversion step.

Definition of w. We define a constant w that has two purposes. First, the process of refinement and conversion terminates when the frame size becomes smaller than or equal to w. Second, the intermediate network $\mathcal{M}$ is constructed from the original network $\mathcal{N}$ by replacing each edge in $\mathcal{N}$ with $2w$ edges. We define w to be a constant that satisfies the following two bounds:

1. $w \ge x$, where x satisfies $(1 - \frac{\alpha}{\sqrt{\log x}})^2 = 1 - \varepsilon/2$, i.e., $x = e^{\alpha^2(1-\sqrt{1-\varepsilon/2})^{-2}}$,
2. $w \ge 2\log^{15} w + 2\log^{10} w - \log^5 w$.

The first bound ensures that the relative congestion $c^{(\zeta)}$ is at most 1. (See Lemma 6.11.) The second bound is to maintain an invariant throughout the frame refinement steps. (See section 6.2.)

6. An asymptotically optimal schedule. In this section we show the existence of an asymptotically optimal schedule. Sections 6.1 through 6.4 concentrate on problem 2 of section 4.2. We begin with an initial schedule $\mathcal{S}^{(0)}$ and transform it to schedule $\mathcal{S}^{(\zeta)}$ through a process of refinement and conversion. All these schedules are designed for the intermediate network $\mathcal{M}$. Section 6.5 concentrates on problem 3 of section 4.2. We describe how to obtain an optimal schedule $\mathcal{S}_{\mathcal{N}}$ for the original network $\mathcal{N}$ from $\mathcal{S}^{(\zeta)}$.

We first define or recall several basic concepts. Given some schedule $\mathcal{S}$, a *region* R of the schedule is some interval of contiguous time steps in the schedule. A *T-frame* is a region of length T. The *congestion* C in a T-frame is the maximum number of packets that cross any edge in that interval, and the *relative congestion* is the ratio C/T. A fractional packet from session i always contributes exactly $\hat{r}_i$ to the relative congestion of any frame.

6.1. An initial schedule $\mathcal{S}^{(0)}$. In $\mathcal{S}^{(0)}$, all sessions are fractional, i.e., $A^{(0)} = \emptyset$. Each packet (of a fractional size) crosses one edge per time step whether or not other packets are using the same edge at the same time. Since the relative congestion is entirely due to fractional sessions, the relative congestion is at most $\sum \hat{r}_i < 1 - \varepsilon/2 = c^{(0)}$ on any edge e. (See Lemma 5.1.)

Note that the above relative congestion holds for any frame size. We choose the initial frame size $I^{(0)} = e^{\log^{2/5} X_{\max}}$, so that $I^{(1)} = \log^2 X_{\max}$, which implies $X_{\max} = e^{\sqrt{I^{(1)}}}$. This allows the sessions with the largest X_i value to be converted in the first iteration of the algorithm (see definition of $B^{(1)}$).

6.2. Frame refinement for schedule $\mathcal{S}^{(q)}$. In this section we describe the frame-refinement process. For each schedule, a frame refinement delays the packets

from integral sessions in a way that dramatically reduces the frame size but does not increase the relative congestion and the length of the schedule by much.

To be more precise, for schedule $\mathcal{S}^{(q)}$, we inductively assume that the relative congestion is at most $c^{(q)}$ for frames of size $I^{(q)}$ or larger and that each integral packet waits at most once every $I^{(q-1)}$ steps after leaving its source. In this frame refinement step we show that there is a way to delay (by an amount related to the frame size) the packets from $A^{(q)}$ so that, in the resulting schedule $\mathcal{S}^{(q+\frac{1}{2})}$, the relative congestion is at most $(1+\delta^{(q)})c^{(q)}$ for any frame of size $I^{(q+1)} = \log^5 I^{(q)}$ or larger, where $\delta^{(q)} = \beta/\sqrt{\log I^{(q)}}$, and each integral packet waits at most once every $I^{(q)}$ steps.

The base case of the initial schedule $\mathcal{S}^{(0)}$ is described in section 6.1. Since there are no integral sessions, no delays are inserted in this step. Trivially, the resulting relative congestion is at most $(1+\delta^{(0)})c^{(0)}$ for any frame of size $I^{(1)}$ or larger at the end of this step, and no packet ever waits.

Let us now consider refining schedule $\mathcal{S}^{(q)}$ for $q > 0$. The refinement is divided into two steps. In the first refinement step we divide the current schedule into blocks of length $2(I^{(q)})^3 + 2(I^{(q)})^2 - I^{(q)}$, and we insert delays into each block so that its length increases to $2(I^{(q)})^3 + 2(I^{(q)})^2$. We show that these delays can be introduced in such a way that in the central $2(I^{(q)})^3$ steps of each block the relative congestion of frames of at least length $I^{(q+1)}$ is only a little larger than $c^{(q)}$. (See Figure 6.1.) At the beginning and end of each block there are "fuzzy" regions of length $(I^{(q)})^2$ each. In the second step we move the block boundaries so that the fuzzy regions at the end and beginning of adjacent blocks are at the center of the new blocks of $2(I^{(q)})^3 + 2(I^{(q)})^2$ steps. Again, we insert delays into each block, increasing the size of the block by $(I^{(q)})^2$ steps. We show that there is a way to insert these delays so that the final conditions for refining $\mathcal{S}^{(q)}$ are indeed satisfied. (See Figure 6.2.)

In the following we present Lemma 6.2, which will be used extensively in both steps of the refinement. We continue by presenting both steps in detail.

A useful lemma. The following lemma is used to prove Lemma 6.2.

LEMMA 6.1. *Let X and Y be independent random variables. Let Y be binomially distributed with mean μ_y, and let σ_1, σ_2, and v be values such that $\sigma_2 = \sigma_1 - 1/v$. Then,*

$$\Pr[\, X + \mu_y > (1+\sigma_1)v \,] \le 2\Pr[\, X + Y > (1+\sigma_2)v \,].$$

Proof. Let $z = (1+\sigma_1)v - \mu_y$. We have

$$\Pr[\, X + \mu_y > (1+\sigma_1)v \,] = \Pr[\, X > z \,], \tag{6.1}$$

$$\Pr[\, X + Y > (1+\sigma_2)v \,] = \Pr[\, X + Y > \mu_y + z - 1 \,]. \tag{6.2}$$

Note also that

$$\begin{aligned}\Pr[\, X + Y > \mu_y + z - 1 \,] &\ge \Pr[\, X > \mu_y + z - 1 - \lfloor \mu_y \rfloor \text{ and } Y \ge \lfloor \mu_y \rfloor \,] \\ &= \Pr[\, X > z - 1 + \mu_y - \lfloor \mu_y \rfloor \,] \Pr[\, Y \ge \lfloor \mu_y \rfloor \,].\end{aligned}$$

This last equality follows from the independence of X and Y. Theorem B.1 in [12] shows that $\Pr[\, Y \ge \lfloor \mu_y \rfloor \,] \ge 1/2$. Since $\mu_y - \lfloor \mu_y \rfloor < 1$, we have

$$\Pr[\, X + Y > \mu_y + z - 1 \,] \ge \frac{1}{2}\Pr[\, X > z \,].$$

Our lemma follows from equalities (6.1) and (6.2) and the above inequality. ∎

We say that a packet is *active* during some region of a schedule if the packet belongs to some integral session and it traverses at least one edge during the region. Since we maintain the invariant that a packet waits at most once every $I^{(q-1)}$ steps after leaving its source, an *inactive* packet is either at its source or its destination during the entire region. Lemma 6.2 below is a stepping stone that allows us to reduce the frame size from $I^{(q)}$ to $poly \log I^{(q)}$. We invoke this lemma for various values of s, t, r, and R.

LEMMA 6.2. *Consider some region R of a schedule where the relative congestion is at most $r = \Theta(1)$ for frames of length s or more, where $\log^3 I^{(q)} \le s \le (I^{(q)})^2$. Consider any edge e and any t-frame, where $\log^2 I^{(q)} \le t \le 2\log^2 I^{(q)}$. Assume each active packet in the region is delayed between the beginning of R and the beginning of the t-frame by a number of steps randomly, independently, and uniformly chosen from $[1, s]$. Then, for any constant k there is some value $\gamma = \Theta(1)/\sqrt{\log I^{(q)}}$ such that the probability of having a relative congestion larger than $r(1+\gamma)$ on e during the t-frame is at most $(I^{(q)})^{-k}$.*

Proof. Let the random variable X be the frame congestion on e during the t-frame due to the active packets after they are delayed. If the relative congestion due to fractional sessions is r_f, the frame congestion due to fractional sessions in the t-frame is exactly $r_f t$. Since the active packets are the only integral-session packets that can cross e during the region, the frame congestion on e during the t-frame is $X + r_f t$ after the delay.

Consider now a binomial random variable Y with parameters $(r_f s, t/s)$ and mean $E[Y] = r_f t$. From Lemma 6.1, the probability p that the congestion in the t-frame is larger than $(1+\gamma)rt$ after the packets have been delayed is

$$p = \Pr[\, X + r_f t > (1+\gamma)rt \,] \le 2\Pr[\, X + Y > (1+\sigma)rt \,],$$

where $\sigma = \gamma - 1/rt$. Since $t \ge \log^2 I^{(q)}$ and $r = \Theta(1)$, $\gamma = \Theta(1)/\sqrt{\log I^{(q)}}$ if and only if $\sigma = \Theta(1)/\sqrt{\log I^{(q)}}$. Let $\sigma = v/\sqrt{\log I^{(q)}}$, where v is a constant. We shall choose an appropriate value v so that the lemma is satisfied.

We first concentrate on X. Since the active packets are delayed up to s steps, an active packet that crosses e in the t-frame after the delay could cross e in an interval of $t + s$ steps before the delay. The relative congestion due to active packets is at most $r - r_f$ in that interval before the delay. Hence, at most $(t+s)(r-r_f)$ active packets can cross e in the t-frame after the delay, and each of them has a probability of at most t/s of doing so.

Recall that Y is a binomial random variable with parameters $(r_f s, t/s)$. We define Z to be a binomial random variable with parameters $(n, t/s)$, where $n = r(t+s) > (r - r_f)(t+s) + r_f s$. It is easy to see that

$$p \le 2\Pr[\, X + Y > (1+\sigma)rt \,] \le 2\Pr[\, Z > (1+\sigma)rt \,].$$

Therefore, we bound the probability p as follows:

$$p \le 2 \sum_{i=(1+\sigma)rt}^{r(t+s)} \binom{r(t+s)}{i} (t/s)^i (1 - t/s)^{r(t+s)-i}.$$

We bound the sum by observing that $(1+\sigma)rt$ is larger than $E[Z] = (t+s)rt/s$, since $t/s \le 2/\log I^{(q)}$. Thus, the first term of the sum is the largest. Hence, from the fact

that there are at most $r(t+s)$ terms in the sum, we have

$$p \le 2r(s+t)\binom{r(t+s)}{(1+\sigma)rt}(t/s)^{(1+\sigma)rt}(1-t/s)^{r(t+s)-(1+\sigma)rt}.$$

By applying the inequality $\binom{a}{b} \le (ae/b)^b$ for $0 < b < a$, we get

$$p \le 2r(s+t)\left(\frac{(t+s)e}{(1+\sigma)t}\right)^{(1+\sigma)rt}(t/s)^{(1+\sigma)rt}(1-t/s)^{r(t+s)-(1+\sigma)rt}.$$

Now applying the inequality $\ln(1+x) \ge x - x^2/2$ for $0 \le x \le 1$, for the case $x = \sigma$,

$$p \le 2r(s+t)\left((1+t/s)e^{1-\sigma+\sigma^2/2}\right)^{(1+\sigma)rt}(1-t/s)^{r(t+s)-(1+\sigma)rt}.$$

Finally, by applying the inequality $(1+x) \le e^x$ for $1+x = 1+t/s$ in one case and for $1+x = 1-t/s$ in the other, we obtain

$$p \le 2r(t+s)e^{-rt\sigma^2(1/2-\sigma/2-t/\sigma^2 s-2t/\sigma s)}.$$

The bounds on s and t and the definitions of r and σ imply that we can choose a constant v large enough so that $p < (I^{(q)})^{-k}$ for any constant $k > 0$. □

The first refinement step for schedule $\mathcal{S}^{(q)}$. We first divide the interval $[\mathcal{T}, \mathcal{T}+|\mathcal{S}^{(q)}|)$ into blocks of length $2(I^{(q)})^3 + 2(I^{(q)})^2 - I^{(q)}$. We shall reschedule each block B independently. During a block B we only delay active packets.

For each block B, each active packet in B is assigned a delay randomly, uniformly, and independently chosen from $[1, I^{(q)}]$. An active packet p, whose assigned delay is x, is delayed in the first $xI^{(q)}$ steps of B once every $I^{(q)}$ steps. In order to independently reschedule the next block, packet p is also delayed in the last $(I^{(q)} - x)I^{(q)}$ steps of B once every $I^{(q)}$ steps. Therefore, a rescheduled block has length $2(I^{(q)})^3 + 2(I^{(q)})^2$.

Before the delays are inserted to reschedule block B, an active packet p is delayed at most once within the block, provided that $2(I^{(q)})^3 + 2(I^{(q)})^2 - I^{(q)} < I^{(q-1)}$, which holds as long as $I^{(q)}$ is larger than some constant. Prior to inserting any new delay to a block, we check if it is within $I^{(q)}$ steps of the single old delay. If the new delay would be too close to the old delay, then it is simply not inserted. The loss of one delay in a block has a negligible effect on the probability analysis that follows.

Lemma 6.4 shows that with the insertion of delays we can dramatically reduce the frame size in the center of the block and increase the relative congestion only slightly. In order to prove Lemma 6.4, we need the following fact.

LEMMA 6.3. *If the relative congestion in every frame of size T to $2T-1$ is at most r, then the relative congestion in any frame of size T or greater is at most r.*

Proof. Consider a frame of size T', where $T' > 2T-1$. The first $\lfloor T'/T \rfloor T - T$ steps of the frame can be broken into T-frames, each with relative congestion r. The remainder of the T'-frame consists of a single frame of size between T and $2T-1$ steps in which the relative congestion is also at most r. □

LEMMA 6.4. *There exists a way of choosing delays so that in between the first and last $(I^{(q)})^2$ steps of block B, the relative congestion of any frame of size $\log^2 I^{(q)}$ or larger is at most $(1+\gamma_1)c^{(q)}$ for some $\gamma_1 = \Theta(1)/\sqrt{\log I^{(q)}}$.*

Proof. With each edge e, we associate a bad event. A bad event on e happens when the frame congestion on edge e is more than $(1+\gamma_1)c^{(q)}I$ during *any* I-frame

of size $\log^2 I^{(q)}$ or larger. Due to Lemma 6.3, it is sufficient to prove the statement for all frames of size between $\log^2 I^{(q)}$ and $2\log^2 I^{(q)}$. We shall use the Lovász local lemma to show that the probability that no bad event occurs is nonzero.

We first bound the dependence, d, of bad events. Two bad events on two edges are dependent only if a packet from a session $i \in A^{(q)}$ can use both edges. At most $c^{(q)}(2(I^{(q)})^3 + 2(I^{(q)})^2 - I^{(q)})$ packets (from sessions in $A^{(q)}$) can cross the same edge in block B, and each packet crosses at most $2(I^{(q)})^3 + 2(I^{(q)})^2 - I^{(q)}$ edges in B. As we shall show later, $c^{(q)} \leq 1$. Therefore, a bad event can be dependent on at most $O((I^{(q)})^6)$ other bad events.

We now bound the probability, p, that a bad event happens on e. Consider a particular I-frame, where $\log^2 I^{(q)} \leq I \leq 2\log^2 I^{(q)}$, that lies completely between the first and last $(I^{(q)})^2$ steps of B. By setting $R = B$, $r = c^{(q)}$, $s = I^{(q)}$, and $t = I$, we apply Lemma 6.2 to show that for any constant k_1 there is some value $\gamma_1 = \Theta(1)/\sqrt{\log I^{(q)}}$ such that the probability p_1 of a bad event happening on e in the I-frame is smaller than $(I^{(q)})^{-k_1}$.

Since there are $O((I^{(q)})^3 \log^2 I^{(q)})$ possible I-frames in B, the probability that a bad event happens on e during any I-frame is $p < p_1 O((I^{(q)})^3 \log^2 I^{(q)})$. We can set the value k_1 appropriately so that this probability is upper bounded by $O((I^{(q)})^{-7})$.

Therefore, we have $4pd < 1$, and our lemma follows from the Lovász local lemma. □

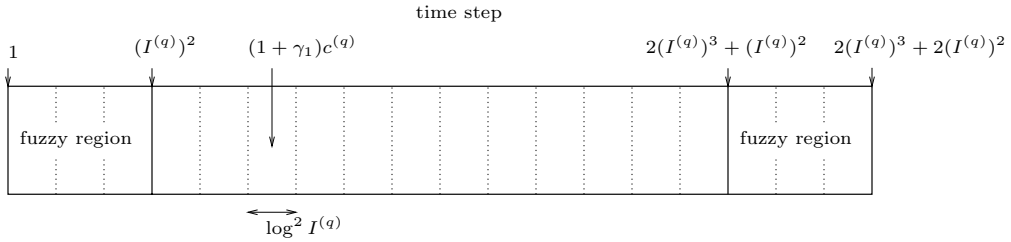


FIG. 6.1. *Situation after the first refinement step.*

At the end of the first refinement step, the center of each block has small relative congestion for small frame sizes. However there are regions of $(I^{(q)})^2$ steps at the beginning and end of each block that may have very large relative congestion. We call these "fuzzy" regions, and we deal with them in the second refinement step.

The second refinement step for schedule $\mathcal{S}^{(q)}$. We start the second step of the refinement by relocating the block boundaries so that blocks still have $2(I^{(q)})^3 + 2(I^{(q)})^2$ steps, but now the fuzzy regions that were at the beginning and end of adjacent blocks are in the center of new blocks. Then, a new block has two "clean" regions of $(I^{(q)})^3$ steps each at the beginning and the end, and a fuzzy region of length $2(I^{(q)})^2$ steps in the center.

As in the first step of the refinement we now concentrate on individual blocks. We first show that the relative congestion is not very large for frames of size $(I^{(q)})^2$ or larger (even in the fuzzy region).

LEMMA 6.5. *For any choice of delays in the first step of the refinement, the relative congestion in any frame of size* $(I^{(q)})^2$ *or larger is at most* $(1 + 2/I^{(q)})c^{(q)}$.

Proof. Without loss of generality we shall assume that all the sessions are integral. Consider an I-frame with I_1 steps before the center of the block and I_2 steps after the center. ($I = I_1 + I_2$, and either I_1 or I_2 could be zero.) A packet crosses some edge e in

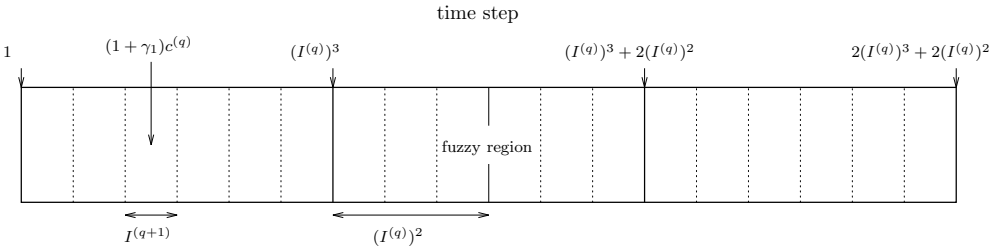


FIG. 6.2. *Situation after relocating block boundaries.*

the I_1-frame only if it did so in some frame of length $I_1 + I^{(q)}$ before the delays where inserted. Therefore, at most $(I_1 + I^{(q)})c^{(q)}$ packets can cross edge e in the I_1-frame. Similarly, at most $(I_2 + I^{(q)})c^{(q)}$ packets can cross edge e in the I_2-frame. Therefore, the congestion in the I-frame can be at most $(I_1 + I_2 + 2I^{(q)})c^{(q)} = (I + 2I^{(q)})c^{(q)}$, and for $I \geq (I^{(q)})^2$ the relative congestion is at most $(1 + 2/I^{(q)})c^{(q)}$. □

Now, in order to reduce the frame size in the fuzzy region, we consider only the active packets in each block B, and we assign a delay randomly, independently, and uniformly chosen from $[1, (I^{(q)})^2]$ to each active packet. A packet p with delay x waits once every $(I^{(q)})^3/x$ at the beginning of the block and once every $(I^{(q)})^3/((I^{(q)})^2 - x)$ at the end. As in the first step a delay is not inserted if it is going to be within $I^{(q)}$ steps of an existing delay for a moving packet.

The block length after the delay insertion is $2(I^{(q)})^3 + 3(I^{(q)})^2$, and the fuzzy region can be $(I^{(q)})^2$ steps longer, spanning steps $(I^{(q)})^3$ to $(I^{(q)})^3 + 3(I^{(q)})^2$.

The next lemma shows that there is some way of inserting delays so that the frame size in the fuzzy region is reduced, and the frame size and relative congestion in the rest of the block are increased by only a small amount.

LEMMA 6.6. *In a block B, there exists a way of choosing delays so that in the fuzzy region (i.e., interval $[(I^{(q)})^3, (I^{(q)})^3 + 3(I^{(q)})^2]$) the relative congestion of any frame of size $\log^2 I^{(q)}$ or larger is at most $(1 + \gamma_2)c^{(q)}$ for some $\gamma_2 = \Theta(1)/\sqrt{\log I^{(q)}}$, and so that in the intervals $[I^{(q)} \log^3 I^{(q)}, (I^{(q)})^3]$ and $[(I^{(q)})^3 + 3(I^{(q)})^2, 2(I^{(q)})^3 + 3(I^{(q)})^2 - I^{(q)} \log^3 I^{(q)}]$ the congestion of any frame of size $\log^2 I^{(q)}$ or larger is at most $(1 + \gamma_3)c^{(q)}$ for some $\gamma_3 = \Theta(1)/\sqrt{\log I^{(q)}}$.*

Proof. As in Lemma 6.4, we will use the Lovász local lemma to prove the claim. We associate a bad event with every edge e, so that a bad event happens on e if, for any $I \geq \log^2 I^{(q)}$,

- more than $(1 + \gamma_2)c^{(q)}I$ packets cross e in any I-frame in $[(I^{(q)})^3, (I^{(q)})^3 + 3(I^{(q)})^2]$ (the fuzzy region), or
- more than $(1+\gamma_3)c^{(q)}I$ packets cross e in any I-frame in $[I^{(q)} \log^3 I^{(q)}, (I^{(q)})^3]$ or $[(I^{(q)})^3 + 3(I^{(q)})^2, 2(I^{(q)})^3 + 3(I^{(q)})^2 - I^{(q)} \log^3 I^{(q)}]$.

The dependency, d, of the bad events is bounded as in Lemma 6.4. Two bad events on two edges are dependent if packets from some session $i \in A^{(q)}$ can use both edges. At most $O((I^{(q)})^3)$ packets cross any edge in a block, and each of them can cross at most $O((I^{(q)})^3)$ other edges. Therefore, $d = O((I^{(q)})^6)$.

Now, to bound the probability p of a bad event happening on some edge e, we consider the three intervals separately and sum their respective probabilities. From Lemma 6.3 we only consider frames of length I such that $\log^2 I^{(q)} \leq I \leq 2 \log^2 I^{(q)}$.

Take first some I-frame in $[(I^{(q)})^3, (I^{(q)})^3 + 3(I^{(q)})^2]$ (the fuzzy region). From Lemma 6.5 we know that the relative congestion for frames of size $(I^{(q)})^2$ or longer

is at most $(1+2/I^{(q)})c^{(q)} = \Theta(1)$. Then, by choosing $R = B$, $r = (1+2/I^{(q)})c^{(q)}$, $s = (I^{(q)})^2$, and $t = I$, we can use Lemma 6.2 to show that, for any constant k_2, there is some $\sigma_2 = \Theta(1)/\sqrt{\log I^{(q)}}$ such that the probability p_1 of having relative congestion on e in the I-frame larger than $c^{(q)}(1+2/I^{(q)})(1+\sigma_2) = c^{(q)}(1+\gamma_2)$ is smaller than $(I^{(q)})^{-k_2}$. Note that $\gamma_2 = \Theta(1)/\sqrt{\log I^{(q)}}$.

Take now some I-frame in $[I^{(q)}\log^3 I^{(q)}, (I^{(q)})^3]$ which starts at step j. Given the way delays are inserted, by the jth step an active packet with delay x has been delayed $jx/(I^{(q)})^3$ steps. Thus, the delay of an active packet at the jth step is essentially a random value uniformly chosen from $[1, j/I^{(q)}]$. For $j \geq I^{(q)}\log^3 I^{(q)}$ the value $j/I^{(q)} \geq \log^3 I^{(q)}$.

Note that before inserting delays, from Lemma 6.4 the relative congestion in any frame of length $\log^2 I^{(q)}$ or larger in the interval $[1, (I^{(q)})^3]$ was at most $(1+\gamma_1)c^{(q)}$. Then, we can make $R = [1, (I^{(q)})^3]$, $r = (1+\gamma_1)c^{(q)}$, $s = \log^3 I^{(q)}$, and $t = I$, and we use Lemma 6.2 to show, for any constant k_3, the existence of some $\sigma_3 = \Theta(1)/\sqrt{\log I^{(q)}}$ such that the probability p_2 of having relative congestion larger than $(1+\sigma_3)(1+\gamma_1)c^{(q)} = (1+\gamma_3)c^{(q)}$ on e in the I-frame is smaller than $(I^{(q)})^{-k_3}$. Again, $\gamma_3 = \Theta(1)/\sqrt{\log I^{(q)}}$.

By symmetry, the same value γ_3 makes the probability of a bad event happening on e in some I-frame in $[(I^{(q)})^3 + 3(I^{(q)})^2, 2(I^{(q)})^3 + 3(I^{(q)})^2 - I^{(q)}\log^3 I^{(q)}]$ smaller than $(I^{(q)})^{-k_3}$.

There are $O((I^{(q)})^3 \log^2 I^{(q)})$ possible I-frames as described in total. Hence, we can choose values for k_2 and k_3 such that the probability of a bad event is bounded as $p \leq (p_1 + 2p_2)O((I^{(q)})^3 \log I^{(q)}) < O((I^{(q)})^7)$. Therefore, we can guarantee $4pd < 1$ and invoke the Lovász local lemma to prove the claim. □

Finally, we bound the frame size and the relative congestion in the remaining intervals of the block in the following lemma.

LEMMA 6.7. *The relative congestion in any frame of size* $\log^4 I^{(q)}$ *or larger in the intervals* $[1, I^{(q)}\log^3 I^{(q)}]$ *and* $[2(I^{(q)})^3 + 3(I^{(q)})^2 - I^{(q)}\log^3 I^{(q)}, 2(I^{(q)})^3 + 3(I^{(q)})^2]$ *is at most*

$$(1+\gamma_1)(1+1/\log I^{(q)})c^{(q)} = (1+\gamma_4)c^{(q)}.$$

Proof. Let us first consider some I-frame in $[1, I^{(q)}\log^3 I^{(q)}]$. Recall that, before inserting delays, the relative congestion for frames of size $\log^2 I^{(q)}$ or more was at most $(1+\gamma_1)c^{(q)}$. In the interval no packet is delayed more than $\log^3 I^{(q)}$ steps. Therefore, the packets crossing some edge e in the I-frame could have crossed e in some interval of at most $I + \log^3 I^{(q)}$ steps, and the congestion in the I-frame can be of at most $(I + \log^3 I^{(q)})(1+\gamma_1)c^{(q)}$. For $I \geq \log^4 I^{(q)}$ the claim follows. The proof for interval $[2(I^{(q)})^3 + 3(I^{(q)})^2 - I^{(q)}\log^3 I^{(q)}, 2(I^{(q)})^3 + 3(I^{(q)})^2]$ is similar. □

From the above two lemmas we have that any frame of length at least $\log^4 I^{(q)}$ in each of the different intervals has at most a relative congestion $(1+\gamma)c^{(q)}$, where $\gamma = \max(\gamma_2, \gamma_3, \gamma_4)$ and $\gamma = O(1)/\sqrt{\log I^{(q)}}$. We need to be careful now with the relative congestion in frames that overlap several intervals or several blocks. We can safely say that for any frame of size $I^{(q+1)} = \log^5 I^{(q)}$ or larger in the schedule $\mathcal{S}^{(q+\frac{1}{2})}$ obtained after the frame refinement, the relative congestion is at most $(1+\delta^{(q)})c^{(q)}$ for some $\delta^{(q)} = \beta/\sqrt{\log I^{(q)}}$ large enough.

6.3. Conversion for schedule $\mathcal{S}^{(q)}$. In the conversion process we transform the schedule $\mathcal{S}^{(q+\frac{1}{2})}$, obtained from the frame refinement step, into a new schedule $\mathcal{S}^{(q+1)}$. In this new schedule, all the sessions in $B^{(q+1)}$ which were fractional in $\mathcal{S}^{(q)}$

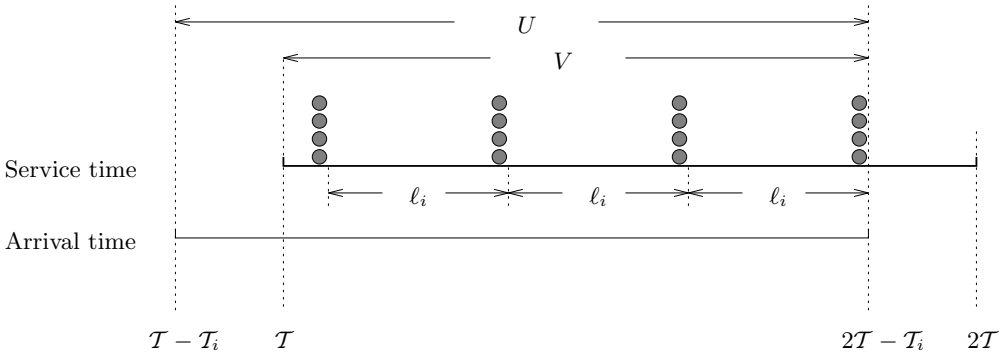


FIG. 6.3. *Session-i packets that are injected in interval U are assigned initial tokens in interval V. The interval V is divided into consecutive intervals of length ℓ_i, each of which has s_i initial tokens. The initial tokens are shown in solid dots.*

have been made integral, and the relative congestion of frames of size $I^{(q+1)}$ or larger is at most $c^{(q+1)} = (1 + \delta^{(q)})^2 c^{(q)}$.

At the beginning of this step, we inductively assume that the relative congestion is at most $(1 + \delta^{(q)})c^{(q)}$ for any frame of size $I^{(q+1)}$ or larger, where $\delta^{(q)} = \beta/\sqrt{\log I^{(q)}}$. If the set $B^{(q+1)}$ is empty, then we skip this conversion step; clearly, the relative congestion is at most $c^{(q+1)}$ for any frame of size $I^{(q+1)}$, and we are done.

On the other hand, if the set $B^{(q+1)}$ is not empty, then for each session $i \in B^{(q+1)}$ we apply the following two processes. (a) In the discretization process we convert the schedule for fractional session-i packets into a schedule for integral packets in which no packet has to wait too long before it starts moving. (b) In the delay-insertion process we delay the time at which packets start moving (i.e., we insert *initial delays*) in such a way that the relative congestion requirements are satisfied.

Discretization. We first show how to transform a fractional session in $B^{(q+1)}$ into an integral session. Consider a session i in $B^{(q+1)}$. When session i is fractional, a packet of size $\hat{r}_i = s_i/\ell_i$ is injected at every time step, where ℓ_i and s_i are integer constants defined in section 5. A fractional packet marches to its destination one edge at a time with no delay.

We want to replace these fractional packets by integral packets. An integral packet waits at its source until it finds an unused *initial token*. Then, it crosses one edge every time step until it reaches its destination. The number of initial tokens and their distribution have to be carefully chosen so that no packet waits at its source for too long.

To transform session i, we consider the two intervals shown in Figure 6.3, $U = [\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ and $V = [\mathcal{T}, 2\mathcal{T} - \mathcal{T}_i)$. When session i is converted, we distribute enough *initial tokens* in the interval V to accommodate all the session-i arrivals during U. Integral packets arrive at a rate r_i during U, and initial tokens will appear at a rate roughly equal to $\hat{r}_i$ during V. Recall from section 5 that $\hat{r}_i$ is slightly larger than r_i. By choosing the interval U long enough (i.e., $\mathcal{T}$ large enough), we guarantee that there are more initial tokens than arrivals.

Let $|V| = \mathcal{T} - \mathcal{T}_i$ be the length of interval V. We divide V into consecutive intervals of length ℓ_i (starting from the end), and we put s_i initial tokens in the last slot of each ℓ_i-interval. Note that if $|V|$ is not an integer multiple of ℓ_i, then the first ℓ_i-interval is "incomplete." (See Figure 6.3.) We show that there are enough initial

tokens and that no packet waits too long for an unused one.

LEMMA 6.8. *For a converted session $i \in B^{(q+1)}$, every session-i packet that is injected during U finds an unused initial token in V within $\mathcal{T}_i + \ell_i = O(1/r_i + D_i)$ steps of its injection.*

Proof. Let $x = \mathcal{T}/(\mathcal{T} - \mathcal{T}_i)$ be the ratio of the length of interval U to the length of interval V. It suffices to show that s_i, the number of initial tokens in an ℓ_i-interval (shown in Figure 6.3), is as large as the number of session-i arrivals during an interval of length $x\ell_i$. At most $n = x\ell_i r_i + 1$ packets can arrive during $x\ell_i$ steps. Since $\mathcal{T} \geq (1 + 4/\varepsilon) \max_i \mathcal{T}_i$ by definition, we have $x \leq 1 + \varepsilon/4$ and $n \leq \ell_i r_i (1 + \varepsilon/4) + 1$. By the left-hand side of property 1 of Lemma 5.1, we have $n \leq s_i$. Therefore, we have enough initial tokens. Since the initial tokens are at the end of an ℓ_i-interval, each packet can use an initial token that appears after the packet arrival time. It is also easy to verify that an unused initial token appears within $\mathcal{T}_i + \ell_i = O(\mathcal{T}_i)$ steps of the packet injection. ∎

Delay insertion. Before any delay is inserted for a packet from session $i \in B^{(q+1)}$, the packet leaves its source at the time of its initial token and marches to its destination with no more waiting. Now we insert an initial delay for each session-i packet, which has the effect of deferring the start time of the packet. We choose the delays uniformly from $[1, \ell_i]$. After the initial delay each packet travels to its destination without further delay.

LEMMA 6.9. *Consider a particular edge e and a particular t-frame during interval $[\mathcal{T}, 2\mathcal{T})$. Suppose session i requires edge e; then the expected number of session-i packets that use e in the t-frame is at most $ts_i/\ell_i = t\hat{r}_i$.*

Proof. Let us assume first that delays have not been inserted yet. Due to the way initial tokens are distributed, session-i packets cross edge e in a very synchronous manner: a batch of at most s_i packets crosses every ℓ_i steps. Since we want an upper bound on the expectation, we assume that exactly s_i packets cross e every ℓ_i steps.

Let us now partition time in ℓ_i-intervals, so that each interval ends with a step in which packets cross e (i.e., all packets cross e in the last step of the intervals). Observe that, once delayed, all the packets that crossed e in the last step of some ℓ_i-interval will cross it in the following interval. Then, the total number of packets crossing e in an ℓ_i-interval after the delay insertion is exactly s_i. Also, after the insertion of delays, the expected number of packets crossing e in some subinterval of length ℓ of an ℓ_i-interval is exactly $\ell s_i/\ell_i$.

Take now the t-frame, and consider the incomplete ℓ_i-intervals it contains. There can be at most one at the beginning and one at the end. Assume they have lengths t_1 and t_2, respectively. From the above observations, the expected number of packets crossing e in the t_1 (resp., t_2) subinterval is $t_1 s_i/\ell_i$ (resp., $t_2 s_i/\ell_i$). In the remainder of the t-frame the number of packets crossing is exactly $(t - t_1 - t_2)s_i/\ell_i$. Hence, the expected number of packets crossing e in the t-frame is $(t - t_1 - t_2)s_i/\ell_i + t_1 s_i/\ell_i + t_2 s_i/\ell_i = ts_i/\ell_i$. ∎

We now use a Chernoff bound and the Lovász local lemma to show the following.

LEMMA 6.10. *There exists a way of choosing the initial delays for sessions in $B^{(q+1)}$ such that the relative congestion in any frame of size $I^{(q+1)}$ or bigger is at most $c^{(q+1)}$ after the delays are inserted.*

Proof. Due to Lemma 6.3, it is sufficient to prove the result for all frames of size $I^{(q+1)}$ to $2I^{(q+1)}$. We associate a bad event with each edge e and each I-frame, where $I^{(q+1)} \leq I \leq 2I^{(q+1)}$. A bad event $E_{\{e,I\}}$ happens when more than $Ic^{(q+1)}$ packets use e during frame I. We use the Lovász local lemma to show that with nonzero

probability no bad event occurs. Let $D_{\max} = \max_{i \in B^{(q+1)}} D_i$, $r_{\min} = \min_{i \in B^{(q+1)}} r_i$, $X = \max_{i \in B^{(q+1)}} D_i + 1/r_i$, and $\ell_{\max} = \max_{i \in B^{(q+1)}} \ell_i$.

We first bound the dependency d of bad events. Note that the probability space is given by the delays assigned to packets from sessions in $B^{(q+1)}$. Hence, a bad event $E_{\{e,I\}}$ is dependent on another bad event $E_{\{e',I'\}}$ only if there is a packet p from a session $i \in B^{(q+1)}$ such that there is a nonzero probability that p uses e during the I-frame and there is a nonzero probability that p uses e' during the I'-frame.

There are at most $1/r_{\min}$ sessions in $B^{(q+1)}$, each of which is at most $D_{\max}$ long. Therefore, $E_{\{e,I\}}$ depends on $E_{\{e',I'\}}$ for at most $D_{\max}/r_{\min} = O(X^2)$ choices of e'. Furthermore, intervals I and I' cannot be more than $D_{\max} + \ell_{\max}$ steps apart. (Otherwise any session-i packet either has probability 0 of crossing edge e during I or probability 0 of crossing e' during I'.) Therefore, the starting point of I' is limited to $2D_{\max} + 2\ell_{\max} + 4I^{(q+1)}$ locations, and the total possible choices for I' is at most $(2D_{\max} + 2\ell_{\max} + 4I^{(q+1)})I^{(q+1)} = O(X(I^{(q+1)})^2)$. We conclude that the dependency d is $O(X^3(I^{(q+1)})^2)$.

We now bound the probability p that a bad event $E_{\{e,I\}}$ happens. By our inductive assumption, the frame congestion on edge e during the I-frame is at most $(1 + \delta^{(q)})c^{(q)}I$ before the conversion. Let S be the set of sessions in $B^{(q+1)}$ that use edge e. When sessions in $B^{(q+1)}$ are fractional, they contribute exactly $I \sum_{i \in S} \hat{r}_i$ to the frame congestion. Lemma 6.9 implies that the expected frame congestion due to the sessions in $B^{(q+1)}$ is at most $I \sum_{i \in S} \hat{r}_i$ after the initial delays are inserted. The congestion due to sessions not in $B^{(q+1)}$ does not change during the conversion. Hence, the expected frame congestion on edge e during the I-frame is at most $(1 + \delta^{(q)})c^{(q)}I = \mu$. We bound the probability of $E_{\{e,I\}}$ as follows.

$$\begin{aligned}
p &= \Pr\left[\text{ Frame congestion on } e \text{ in } I > c^{(q+1)}I \right] \\
&= \Pr\left[\text{ Frame congestion on } e \text{ in } I > (1+\delta^{(q)})\mu \right] \\
&\le e^{-(\delta^{(q)})^2\mu/3} \\
&\le e^{-(1-\varepsilon)\beta^2 I^{(q+1)}/(3\log I^{(q)})} \\
&\le e^{-(1-\varepsilon)\frac{\beta^2}{3}(I^{(q+1)})^{1/5}(I^{(q+1)})^{3/5}} \\
&\le e^{-(1-\varepsilon)\frac{\beta^2}{3}(I^{(q+1)})^{1/5}\log^{6/5} X}.
\end{aligned}$$

The first inequality follows from Lemma 2.3. The second inequality holds since $\mu > (1-\varepsilon)I \ge (1-\varepsilon)I^{(q+1)}$ and from the definition of $\delta^{(q)}$. The third inequality follows from the recurrence for $I^{(q+1)}$. The last inequality follows from the fact that $\log^2 X \le I^{(q+1)}$. (This explains the need for $\log^2 X_i \le I^{(q+1)}$ in the definition of $B^{(q+1)}$.)

When β is a sufficiently large constant, we have $4dp < 1$. Hence, the Lovász local lemma implies that with nonzero probability no bad events occur. That is, there exists a way to choose the initial delays for sessions in $B^{(q+1)}$ such that for all frames of size $I^{(q+1)}$ or larger the relative congestion is at most $c^{(q+1)}$.

Note that in the proof of this lemma we associate a bad event with each edge e and each interval I. Why couldn't we associate a bad event with each edge e only and then use a union bound on the number of intervals, as in Lemma 6.4? This is because we are considering all the session-i packets during an interval of length $\mathcal{T}$, which can be much bigger than $1/r_i + D_i$ for some sessions i. □

6.4. Termination at schedule $\mathcal{S}^{(\zeta)}$. The succession of refinement and conversion terminates at schedule $\mathcal{S}^{(\zeta)}$ when the frame size $I^{(\zeta)}$ becomes smaller than

or equal to w, a constant defined in section 5. The following lemma shows that the relative congestion of $\mathcal{S}^{(\zeta)}$ is small.

LEMMA 6.11. *In the schedule $\mathcal{S}^{(\zeta)}$ all sessions are integral and the relative congestion is at most $c^{(\zeta)} < 1$ for any frame of size $I^{(\zeta)}$ or larger.*

Proof. One can verify that $B^{(q+1)}$ forms a partition of all the sessions. Therefore, all the sessions are integral in the schedule $\mathcal{S}^{(\zeta)}$. By our induction, the relative congestion is at most $c^{(\zeta)}$ for all frames of size $I^{(\zeta)}$ or larger. Hence, we only need to show that $c^{(\zeta)} < 1$.

Due to the termination conditions, $x \le I^{(\zeta-1)}$, where x is defined in section 5. Let $\Delta = \beta/\sqrt{\log x}$, and observe that $\delta^{(\zeta-1)} \le \Delta < 1$. By the recursive definition of $c^{(\zeta)}$, we have

$$\begin{aligned}
c^{(\zeta)} &= (1+\delta^{(\zeta-1)})^2(1+\delta^{(\zeta-2)})^2 \dots (1+\delta^{(0)})^2 c^{(0)} \\
&< (1+\Delta)^2(1+\Delta^2)^2(1+\Delta^4)^2(1+\Delta^8)^2 \dots c^{(0)} \\
&\le (1-\Delta)^{-2}\left\{(1-\Delta)^2(1+\Delta)^2(1+\Delta^2)^2(1+\Delta^4)^2(1+\Delta^8)^2 \dots\right\} c^{(0)} \\
&\le (1-\Delta)^{-2} c^{(0)} \\
&= \left(1-\beta/\sqrt{\log x}\right)^{-2} c^{(0)} \\
&= \frac{1-\varepsilon/2}{1-\varepsilon/2} \\
&= 1.
\end{aligned}$$

The first inequality holds since $\delta^{(q)} < (\delta^{(q+1)})^2$ for all q by the recurrence defined in section 5. The third inequality holds since $\Delta < 1$, and therefore the "telescope product" in the braces is less than 1. The last equality holds by the above choice of x and the definition of $c^{(0)}$ in section 5. ∎

Now, we have to make sure that in the resulting schedule $\mathcal{S}^{(\zeta)}$ no packet waits too long. The conversion step guarantees that when a session i becomes integral, no packet waits more than $O(D_i + 1/r_i)$ steps before it starts moving, and it does not wait anymore. The last frame refinement step also guarantees that a moving packet never waits more than once every $I^{(\zeta-1)}$ steps. However, all the frame refinement steps that an integral packet has to go through can, in fact, delay the time it starts moving. The following lemma shows that this delay does not add up to a large amount, and therefore that a session-i packet reaches its destination in at most $O(D_i + 1/r_i)$ steps in the schedule $\mathcal{S}^{(\zeta)}$.

LEMMA 6.12. *During frame-refinement a session-i packet is delayed by at most $2(D_i + 1/r_i)$ steps before it starts moving.*

Proof. Suppose session i first becomes integral in schedule $\mathcal{S}^{(q')}$. Consider a session-i packet p. For schedule $\mathcal{S}^{(q)}$, where $q \le q'-1$, p is never delayed during frame refinement. For schedule $\mathcal{S}^{(q)}$, where $q \ge q'$, p is delayed by at most $I^{(q)} + (I^{(q)})^2$ steps before it starts moving. Therefore, the total delay inserted during all the frame refinement steps is at most $\sum_{q \ge q'} I^{(q)} + (I^{(q)})^2$. Since session i becomes integral for schedule $\mathcal{S}^{(q')}$, we must have $i \in B^{(q')}$. By the definition of $B^{(q')}$, $D_i + 1/r_i \ge (I^{(q')})^2$. Since $I^{(q)}$ decreases polylogarithmically, a session-i packet is delayed during frame refinement by at most $2(D_i + 1/r_i)$ steps before it starts moving. ∎

We proceed to prove that $\mathcal{S}^{(\zeta)}$ has all the properties.

THEOREM 6.13. *Given network $\mathcal{M}$ and a set of sessions as defined in section 1.2, there is a schedule $\mathcal{S}^{(\zeta)}$ such that the following hold.*

1. *The relative congestion is at most* 1 *for any frame of size larger than a certain constant.*
2. *After leaving its source, each packet waits at most once every* $O(1)$ *steps, which implies that all edge queues in* $\mathcal{M}$ *have size* $O(1)$.
3. *For all sessions* i*, any session-*i *packet reaches its destination within* $O(1/r_i + D_i)$ *steps of its injection.*
4. *All session-*i *arrivals during* $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ *are serviced during* $[\mathcal{T}, 2\mathcal{T})$*, i.e., all packets leave their source no earlier than* $\mathcal{T}$ *and reach their destination before* $2\mathcal{T}$.

Proof.

1. By Lemma 6.11, the relative congestion is at most 1 for any frame of size $I^{(\zeta)}$ or larger. Due to the termination conditions $I^{(\zeta)}$ is a constant.
2. By the invariant maintained throughout the frame refinement steps, a packet waits at most once every $I^{(\zeta-1)}$ steps once it leaves its source. In addition, by property 1 above, at most $I^{(\zeta)}$ packets cross an edge during any time step. Therefore, the edge queues have size at most $2I^{(\zeta)}$.
3. We first show that a session-i packet reaches its destination within $\mathcal{T}_i$ steps after it obtains an initial token. After the initial token, a session-i packet is deferred by an initial delay during the conversion step and other delays during the frame refinement step before it could leave its source. The initial delay is at most $\ell_i < 1 + 8/(\varepsilon r_i)$, and the delay during the refinement is at most $2(D_i + 1/r_i)$ by Lemma 6.12. Once the packet starts moving, it reaches its destination in at most $2D_i$ steps by property 2. Therefore, a session-i packet reaches its destination within $4D_i + 1 + (8/\varepsilon + 2)/r_i < \mathcal{T}_i$ steps after obtaining its initial token.

   Since any session-i packet obtains an initial token within $\mathcal{T}_i + \ell_i$ steps of its injection by Lemma 6.8, the packet reaches its destination within $2\mathcal{T}_i + \ell_i = O(1/r_i + D_i)$ steps of its injection.
4. For all session-i arrivals during $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$, the initial tokens are in $[\mathcal{T}, 2\mathcal{T} - \mathcal{T}_i)$. From the discussion of property 3, a session-i packet reaches its destination within $\mathcal{T}_i$ steps after it obtains an initial token. Therefore, all packets leave their sources no earlier than $\mathcal{T}$ and reach their destinations before $2\mathcal{T}$. ∎

6.5. The final schedule for the original network $\mathcal{N}$. We now describe how to create a schedule $\mathcal{S}_{\mathcal{N}}$ for network $\mathcal{N}$ from $\mathcal{S}^{(\zeta)}$. In $\mathcal{S}_{\mathcal{N}}$ at most one packet at a time crosses each edge in $\mathcal{N}$. Recall that in the construction of $\mathcal{M}$ from $\mathcal{N}$, each edge e in $\mathcal{N}$ is replaced by $2w$ consecutive edges $e_1, \ldots, e_{2w}$, where w is a constant defined in section 5.

We first partition the time interval $[\mathcal{T}, 2\mathcal{T})$ into consecutive intervals of length w (recall that by definition $\mathcal{T}$ is a multiple of w). For each w-interval and each edge f in $\mathcal{M}$, as many as w packets, $p_1, p_2, \ldots, p_w$, can cross f during the w-interval by schedule $\mathcal{S}^{(\zeta)}$. We *smooth out* $\mathcal{S}^{(\zeta)}$ so that p_j is the jth packet to cross f in the w-interval, where $p_1, \ldots, p_w$ represents an arbitrary ordering. After smoothing, a packet may not be scheduled to cross the edges on its route in order. For example, a packet may be scheduled to cross edge f before g, whereas f follows g on the route in $\mathcal{M}$. A packet may also be scheduled to leave its source before its injection time. However, $\mathcal{S}^{(\zeta)}$ after smoothing does have the property that at most one packet at a time crosses each edge. We define $\mathcal{S}_{\mathcal{N}}$ as follows. $\mathcal{S}_{\mathcal{N}}$ schedules a packet p to cross e in $\mathcal{N}$ at time t if and only if $\mathcal{S}^{(\zeta)}$ after smoothing schedules p to cross e_{2w} in $\mathcal{M}$ at time t.

LEMMA 6.14. *In $\mathcal{S}_{\mathcal{N}}$, each packet is scheduled to leave its source after its injection and is scheduled to cross the edges on its route in order.*

Proof. We first show that each packet crosses the edges on its route in order. Consider a packet p. Let e and $\hat{e}$ be two edges on p's route in $\mathcal{N}$, where $\hat{e}$ follows e. Let t and $\hat{t}$ be the times that p crosses e and $\hat{e}$ in schedule $\mathcal{S}_{\mathcal{N}}$. We shall show that $t < \hat{t}$.

Let e_{2w} and $\hat{e}_{2w}$ be the edges in $\mathcal{M}$ that correspond to e and $\hat{e}$. Let τ and $\hat{\tau}$ be the times that p crosses e_{2w} and $\hat{e}_{2w}$ in the schedule $\mathcal{S}^{(\zeta)}$ *before* smoothing. Since p crosses the edges in $\mathcal{M}$ in order before smoothing, we have

$$\tau + 2w \leq \hat{\tau}. \tag{6.3}$$

In schedule $\mathcal{S}_{\mathcal{N}}$, packet p crosses e at time t, which is shifted by at most $w-1$ steps from τ. Similarly, $\hat{t}$ is shifted by at most $w-1$ steps from $\hat{\tau}$. Hence we have

$$\begin{array}{rcccl} \tau - (w-1) & \leq & t & \leq & \tau + (w-1), \\ \hat{\tau} - (w-1) & \leq & \hat{t} & \leq & \hat{\tau} + (w-1). \end{array}$$

From (6.3) and the above inequalities, we have $t < \hat{t}$. Therefore, p crosses the edges on its route in order.

The proof that packet p leaves its source after its injection time is similar. Suppose that p is injected into the network at time s. Let edge e be the first edge on the route of p in network $\mathcal{N}$, and let t be the time that p crosses e in $\mathcal{S}_{\mathcal{N}}$. Also, let e_{2w} be the corresponding edge in $\mathcal{M}$, and let τ be the time that p crosses e_{2w} in $\mathcal{S}^{(\zeta)}$ before smoothing. Since in $\mathcal{S}^{(\zeta)}$ before smoothing p crosses the edges in order and leaves its source after its injection, we have

$$s + 2w \leq \tau.$$

In schedule $\mathcal{S}_{\mathcal{N}}$, packet p crosses e at time t, which is shifted by at most $w-1$ steps from τ. Hence we have

$$\tau - (w-1) \quad \leq \quad t \quad \leq \quad \tau + (w-1).$$

Therefore, $s < t$ and packet p leaves its sources in $\mathcal{N}$ after the injection time. □

We summarize the properties of $\mathcal{S}_{\mathcal{N}}$.

THEOREM 6.15. *Schedule $\mathcal{S}_{\mathcal{N}}$ satisfies the following properties.*

1. *At most one packet at a time crosses each edge in $\mathcal{N}$.*
2. *After leaving its source, each packet waits a constant number of steps to cross an edge, which implies all the edge queues in $\mathcal{N}$ have a constant size.*
3. *For all sessions i, any session-i packet reaches its destination within $O(1/r_i + d_i)$ steps of its injection.*
4. *All session-i arrivals during $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ are serviced during $[\mathcal{T}, 2\mathcal{T})$, i.e., all packets leave their source no earlier than $\mathcal{T}$ and reach their destination before $2\mathcal{T}$.*

Proof. The smoothing process guarantees property 1. Properties 2 and 3 come from properties 2 and 3 of $\mathcal{S}^{(\zeta)}$ given in Theorem 6.13, the construction of $\mathcal{M}$ from $\mathcal{N}$, and the fact that each packet is scheduled to reach its destination in $\mathcal{S}_{\mathcal{N}}$ at most w steps later than in $\mathcal{S}^{(\zeta)}$.

To see property 4, recall that the interval $[\mathcal{T}, 2\mathcal{T})$ is partitioned into intervals of size w (with one interval possibly longer than w), and that schedule $\mathcal{S}^{(\zeta)}$ is smoothed

out within each w-interval. Therefore, if a packet is scheduled to cross an edge e during $[\mathcal{T}, 2\mathcal{T})$ according to $\mathcal{S}^{(\zeta)}$, the packet must also be scheduled to cross e during $[\mathcal{T}, 2\mathcal{T})$ according to $\mathcal{S}_{\mathcal{N}}$. Property 4 follows. Property 4 of the above theorem implies that all intervals of $[0, \mathcal{T})$, $[\mathcal{T}, 2\mathcal{T})$, etc. can be scheduled independently. □

6.6. Derivation of the templates. We now describe how to transform $\mathcal{S}_{\mathcal{N}}$ into a template-based schedule. Property 4 of Theorem 6.15 says that all packets considered in schedule $\mathcal{S}_{\mathcal{N}}$ (those injected in interval $[\mathcal{T} - \mathcal{T}_i, 2\mathcal{T} - \mathcal{T}_i)$ for each session i) move from their sources to their destination during interval $[\mathcal{T}, 2\mathcal{T})$. For this reason, we choose $\mathcal{T}$ as the size of each template. Recall that in the conversion step of section 6.3, the placement of the initial tokens is *independent* of the actual packet arrival times. The placement is simply a result of randomization added onto the fixed configuration shown in Figure 6.3. As we have shown, even if each session-i initial token is owned by a session-i packet, we can schedule these packets by schedule $\mathcal{S}_{\mathcal{N}}$. Then, if we place a session-i token in the template of edge e whenever a session-i packet crosses e in $\mathcal{S}_{\mathcal{N}}$, the movement of each packet determines a token sequence, and these token sequences define the locations of all the tokens. We emphasize that the placement of these tokens is fixed as the initial tokens are.

Obviously, the token lag is $O(1)$ for all sequences and the end-to-end delay is $O(1/r_i + d_i)$ for all session-i token sequences. Since each session-i packet is able to obtain an initial token within $O(1/r_i + d_i)$ steps of its injection, Theorem 2.2 implies that the template-based schedule defined by the token sequences achieves a delay bound of $O(1/r_i + d_i)$ and constant-edge queues. Combined with Theorem 2.2, we have a template-based schedule with desired delay bounds and constant-edge queues. In summary, we have the following theorem.

THEOREM 6.16. *Consider an arbitrary network in which sessions are defined. Each session i is associated with an injection rate r_i and path length d_i. Packets are injected into the network along these sessions subject to the injection rates. If the total rate on each edge is at most $1 - \varepsilon$ for a constant $\varepsilon \in (0, 1)$, then there exists a template-based schedule such that each session-i packet reaches its destination within $O(1/r_i + d_i)$ steps of its injection and at most one packet crosses an edge at each time step. This schedule also maintains constant-edge queues.*

Acknowledgments. The authors wish to thank Bruce Maggs, Greg Plaxton, and Salil Vadhan for many helpful comments.

REFERENCES

[1] M. ANDREWS, B. AWERBUCH, A. FERNÁNDEZ, J. KLEINBERG, T. LEIGHTON, AND Z. LIU, *Universal stability results for greedy contention-resolution protocols*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 380–389.

[2] M. ANDREWS AND L. ZHANG, *Minimizing end-to-end delay in high-speed networks with a simple coordinated schedule*, in Proceedings of the IEEE Computer and Communication Societies, INFOCOM, IEEE Computer Society, Los Alamitos, CA, 1999, pp. 380–388.

[3] J. BECK, *An algorithmic approach to the Lovasz Local Lemma* I, Random Structures Algorithms, 2 (1991), pp. 343–365.

[4] A. BORODIN, J. KLEINBERG, P. RAGHAVAN, M. SUDAN, AND D. WILLIAMSON, *Adversarial queueing theory*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 376–385.

[5] H. CHERNOFF, *A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations*, Ann. Math. Stat., 23 (1952), pp. 493–509.

[6] R. L. CRUZ, *A calculus for network delay, Part* I: *Network elements in isolation*, IEEE Trans. Inform. Theory, 37 (1991), pp. 114–131.

[7] R. L. Cruz, *A calculus for network delay, Part* II*: Network analysis*, IEEE Trans. Inform. Theory, 37 (1991), pp. 132–141.

[8] A. Demers, S. Keshav, and S. Shenker, *Analysis and simulation of a fair queueing algorithm*, J. Internetworking: Research and Experience, 1 (1990), pp. 3–26.

[9] A. Elwalid, D. Mitra, and R. H. Wentworth, *A new approach for allocating buffers and bandwidth to heterogeneous, regulated traffic in an ATM node*, IEEE J. Selected Areas in Communications, 13 (1995), pp. 1115–1127.

[10] F. T. Leighton, B. M. Maggs, and S. B. Rao, *Packet routing and job-shop scheduling in* $O(congestion + dilation)$ *steps*, Combinatorica, 14 (1994), pp. 167–186.

[11] F. T. Leighton, B. M. Maggs, and A. W. Richa, *Fast Algorithms for Finding O(Congestion + Dilation) Packet Routing Schedules*, Technical report CMU-CS-96-152, Carnegie Mellon University, Pittsburgh, PA, 1996.

[12] F. T. Leighton and G. Plaxton, *Hypercubic sorting networks*, SIAM J. Comput., 27 (1998), pp. 1–47.

[13] R. Ostrovsky and Y. Rabani, *Universal O (congestion + dilation +* $\log^{1+\epsilon} N$*) local control packet switching algorithm*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 644–653.

[14] A. K. Parekh and R. G. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: The single-node case*, IEEE/ACM Trans. Networking, 1 (1993), pp. 344–357.

[15] A. K. Parekh and R. G. Gallager, *A generalized processor sharing approach to flow control in integrated services networks: The multiple-node case*, IEEE/ACM Trans. Networking, 2 (1994), pp. 137–150.

[16] Y. Rabani and E. Tardos, *Distributed packet switching in arbitrary networks*, in Proceedings of the 28th Annual ACM Symposium on Theory of Computing, Philadelphia, PA, 1996, pp. 366–375.

[17] J. Spencer, *Ten Lectures on the Probabilistic Methods*, Capital City Press, Philadelphia, PA, 1994.

[18] J. S. Turner, *New directions in communications, or which way to the information age*, IEEE Communications Magazine, 24 (1986), pp. 8–15.

[19] L. Zhang, *An Analysis of Network Routing and Communication Latency*, Ph.D. thesis, MIT, Cambridge, MA, 1997.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1624–1661

A DECOMPOSITION THEOREM FOR TASK SYSTEMS AND BOUNDS FOR RANDOMIZED SERVER PROBLEMS*

AVRIM BLUM†, HOWARD KARLOFF‡, YUVAL RABANI§, AND MICHAEL SAKS¶

Abstract. A lower bound of $\Omega(\sqrt{\log k/\log\log k})$ is proved for the competitive ratio of randomized algorithms for the k-server problem against an oblivious adversary. The bound holds for arbitrary metric spaces (having at least $k+1$ points) and provides a new lower bound for the metrical task system problem as well. This improves the previous best lower bound of $\Omega(\log\log k)$ for arbitrary metric spaces [H.J. Karloff, Y. Rabani, and Y. Ravid, *SIAM J. Comput.*, 23 (1994), pp. 293–312] and more closely approaches the conjectured lower bound of $\Omega(\log k)$. For the server problem on $k+1$ equally spaced points on a line, which corresponds to a natural motion-planning problem, a lower bound of $\Omega(\frac{\log k}{\log\log k})$ is obtained.

The results are deduced from a general decomposition theorem for a simpler version of both the k-server and the metrical task system problems, called the "pursuit-evasion game." It is shown that if a metric space $\mathcal{M}$ can be decomposed into two spaces $\mathcal{M}_{\mathcal{L}}$ and $\mathcal{M}_{\mathcal{R}}$ such that the distance between them is sufficiently large compared to their diameter, then the competitive ratio for this game on $\mathcal{M}$ can be expressed nearly exactly in terms of the ratios on each of the two subspaces. This yields a divide-and-conquer approach to bounding the competitive ratio of a space.

Key words. lower bounds, randomized algorithms, k-server, task systems, on-line algorithms, competitive analysis

AMS subject classifications. 68Q17, 68Q25, 68W20, 91A46

PII. S0097539799351882

1. Introduction and main results.

On-line computation is a setting in which randomization has been shown to have a provable advantage over determinism (see, e.g., [BE]). An on-line computation problem typically involves responding to a sequence of requests in order to minimize some cost function. The standard measure of success is the *competitive ratio* [ST, KMRS], which is, roughly, the maximum over all request sequences of the ratio of the cost charged to the algorithm on a request sequence, to the optimal off-line cost of servicing that sequence. It is useful and customary to view the request sequence as chosen by an adversary who knows the algorithm being used and seeks to force this ratio to be large. Against a deterministic algorithm, the adversary can completely predict the responses of the algorithm, and

*Received by the editors February 17, 1999; accepted for publication (in revised form) August 2, 2000; published electronically November 28, 2000. A preliminary version of this paper appeared in the Proceedings of the 33rd IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 197–207.

http://www.siam.org/journals/sicomp/30-5/35188.html

†School of Computer Science, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213 (avrim@theory.cs.cmu.edu). The work of this author was supported in part by an NSF Postdoctoral Fellowship.

‡College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 (howard@cc.gatech.edu). The work of this author was supported in part by NSF grant CCR 9107349 and by DIMACS.

§Computer Science Department, Technion—IIT, Haifa 32000, Israel (rabani@cs.technion.ac.il). The work of this author was done while he was a graduate student at Tel Aviv University, Department of Computer Science. Part of this work was done while visiting DIMACS.

¶Department of Mathematics, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854-8019 and Department of Computer Science and Engineering, University of California at San Diego, 9500 Gilman Drive, Department 0114, La Jolla, CA 92093-0114 (saks@math.rutgers.edu). The work of this author was supported in part by NSF grant CCR89-11388 and AFOSR grants 89-0512 and 90-0008.

this gives it great power for forcing the algorithm to perform badly. Against a randomized algorithm the adversary knows the algorithm but not the random choices of the algorithm. Intuitively, this can be interpreted by saying that after each successive request, the adversary "knows" only a probability distribution over states of the algorithm rather than the precise state. This restriction on the adversary provides the potential advantage of randomization. (Note that here and throughout this paper we are discussing a version of the adversary known as an "oblivious" adversary [BBKTW, RS]. There are other, more powerful, adversaries that are less vulnerable to randomization.)

In the well-known k-server problem, an algorithm controls k servers, each of which occupies some point in a metric space $\mathcal{M}$. At each time step the algorithm is given a *request*, which is a point in $\mathcal{M}$, and must serve it by moving a server to that point if none is there already. The algorithm is charged a cost equal to the total distance moved. It has been shown that for any metric space having at least $k+1$ points no deterministic on-line algorithm can achieve a competitive ratio less than k [MMS]. (Note that the problem is nontrivial only if there are at least $k+1$ points.) The well-known k-server conjecture [MMS] says that for any metric space, there is a deterministic on-line algorithm that can achieve a competitive ratio of k. In other words, if we define the competitive ratio of a metric space to be the minimum ratio achievable by any algorithm, then the conjecture is that for the k-server problem, the deterministic competitive ratio of any metric space on at least $k+1$ points is exactly k. A breakthrough result [KP] provided a deterministic algorithm with competitive ratio $2k-1$, improving on the previous exponential upper bounds [FRR, Gro].

The power of randomization in this setting was first demonstrated for the *uniform* metric space on $k+1$ points, $\mathcal{U}(k+1)$, in which all pairs of distinct points are equidistant. For this space there is an $O(\log k)$-competitive algorithm, and indeed this is a lower bound.

THEOREM 1.1 (see [FKLMSY, MS, BLS]). *The k-server problem for $\mathcal{U}(k+1)$ has a randomized competitive ratio exactly $1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{k} \sim \ln k$.* In fact, these bounds hold for uniform metric spaces of any size greater than k. Various people have speculated on the following conjecture.

CONJECTURE 1.1. *For any k and any metric space $\mathcal{M}$ on more than k points, the randomized competitive ratio of the k-server problem on $\mathcal{M}$ is $\Theta(\log k)$.*

Unlike the deterministic case, where the lower bound is relatively easy and only the upper bound seems difficult, neither bound has been proved in the randomized case. For the lower bound, the previous best result is the following.

THEOREM 1.2 (see [KRR]). *Let k be a positive integer, and let $\mathcal{M}$ be a metric space with at least $k+1$ points. Then the k-server problem on $\mathcal{M}$ has randomized competitive ratio $\Omega(\min\{\log k, \log\log|\mathcal{M}|\})$.*

If $\mathcal{M}$ is sufficiently large (exponential in k), then the lower bound in Conjecture 1.1 holds. For arbitrary spaces, in particular those whose size is polynomial in k, the lower bound is $\Omega(\log\log k)$. One of the main results of this paper is to improve this lower bound.

THEOREM 1.3. *For any metric space $\mathcal{M}$ with at least $k+1$ points, the k-server problem on $\mathcal{M}$ has randomized competitive ratio $\Omega(\sqrt{\frac{\log k}{\log\log k}})$.*

The competitive ratio of the k-server problem for $\mathcal{M}$ is at least as large as the ratio for a subspace of $\mathcal{M}$, as the adversary can restrict its requests to that subspace. Thus a lower bound on the competitive ratio of the k-server problem for the case that the space has exactly $k+1$ points implies the same lower bound for every metric space.

One way to view this special case is to think of the algorithm as occupying a single point of the space (corresponding to the unique location where there is no server) and to think of the adversary as probing points of this space. When the adversary probes the point on which the algorithm stands, the algorithm must move to a different location. We call this the *pursuit-evasion (PE) game* and call the adversary the pursuer and the algorithm the evader. This paper is about this game. It should be noted that the PE game bears a superficial resemblance to the cat-and-mouse game of [CDRS], but that game models the case of randomized algorithms against a more powerful adversary.

The PE game also models a problem in robotics. Imagine a robot walking down a long hallway of some width n (e.g., if $n = 3$, then the robot may walk either down the left side, the center, or the right side of the hallway). The hallway contains rectangular obstacles, and when the robot meets an obstacle, it must go left or right around it. Any algorithm at all must travel the length of the hallway, so we will not charge for that. Instead we look at the left/right motion of the robot and compare it to the least possible left/right motion by an algorithm that knew the placement of the obstacles in advance. If the hallway has width n, then this is the PE game for the metric space of n equally spaced points on the real line, a metric space we call $\mathcal{L}(n)$. The above lower bound of $\Omega(\sqrt{\frac{\log(n-1)}{\log\log(n-1)}})$ applies, of course, but for this case we have a better lower bound.

THEOREM 1.4. *The PE game on $\mathcal{L}(n)$ has randomized competitive ratio $\Omega(\frac{\log n}{\log\log n})$. Thus if $n > k$, the k-server problem on $\mathcal{L}(n)$ has competitive ratio $\Omega(\frac{\log k}{\log\log k})$.*

This nearly matches the conjectured bounds.

For general spaces $\mathcal{M}$, the special case of Conjecture 1.1 with $k = |\mathcal{M}| - 1$ can be stated as the following conjecture.

CONJECTURE 1.2. *For any metric space $\mathcal{M}$ on n points, the randomized competitive ratio of the PE game on $\mathcal{M}$ is $\Theta(\log n)$.*

As noted, the lower bounds of Conjectures 1.2 and 1.1 are equivalent. On the other hand, an upper bound for the PE game does not have immediate application to the upper bound conjecture for the general k-server problem. (In fact, there is some evidence that for $k = 2$ the competitive ratio might be worse for metric spaces with more than 3 points; see [LR].) In any case, we believe that a solution to the PE game would be a major step towards the solution of the more general problem and would also be interesting in its own right. (Since the appearance of a preliminary version of the present work, significant progress on this problem has been made. Bartal et al. [BBBT] gave a polylog(n) algorithm for the PE game, and more generally for any metrical task system.)

Previously, Conjecture 1.2 was known to be true only for the case of uniform (or nearly uniform) spaces mentioned earlier. Here, we establish Conjecture 1.2 for a dual situation. If $C > 1$, a metric space is *C-unbalanced* if for any three distinct points, the ratio of the largest distance to the smallest nonzero distance is at least C. For example, the metric space consisting of four points in a rectangle with side lengths 1 and C is C-unbalanced.

THEOREM 1.5. *There is a polynomial $p(n)$ such that for all n, the PE problem on any $p(n)$-unbalanced metric space with n points has randomized competitive ratio between $\ln n$ and $3 \ln n$.*

It is worth mentioning that bounds on the competitive ratio of the PE game carry over to the *task system* model of [BLS]. In particular, Theorems 1.3, 1.4, and 1.5 hold if we replace "the PE game on $\mathcal{M}$" by "the task system on $\mathcal{M}$."

1.1. Overview of the method. Theorems 1.3, 1.4, and 1.5 are proved as a consequence of a *decomposition theorem* for the competitive ratio of the PE game. (Henceforth, when we say "competitive ratio" we will mean the "randomized competitive ratio.") The theorem concerns metric spaces that can be split into two subspaces, where the diameter of each subspace is small relative to the overall diameter, and it asserts that the competitive ratio of the PE game on the whole space can be expressed almost exactly in terms of the competitive ratios of the games on the two subspaces.

To state this result, we need some notation. For a finite metric space $\mathcal{M}$, $\delta(\mathcal{M})$ denotes its diameter and $\lambda(\mathcal{M})$ denotes the competitive ratio of the associated PE game. We use the convention that the competitive ratio of a one point space is 0. A subspace $\mathcal{N}$ of $\mathcal{M}$ is said to be γ-*small* in $\mathcal{M}$ for $\gamma < 1$ if $\delta(\mathcal{N}) < \gamma\delta(\mathcal{M})$. We say that $\mathcal{M}$ is *bipartite* if it can be split into two subspaces $\mathcal{M}_L$ and $\mathcal{M}_R$ (called, respectively, the left space and the right space) such that the diameters of $\mathcal{M}_L$ and $\mathcal{M}_R$ are each less than $\delta(\mathcal{M})/2$. We call $(\mathcal{M}_L, \mathcal{M}_R)$ a *bipartition* of $\mathcal{M}$. It is easy to see that if a bipartition exists, then it is unique. If $\mathcal{M}$ is bipartite, then we say that it is γ-*bipartite* for $\gamma < 1/2$ if $\delta(\mathcal{M}_L)$ and $\delta(\mathcal{M}_R)$ are each γ-small in $\mathcal{M}$, and we call $(\mathcal{M}_L, \mathcal{M}_R)$ a γ-*bipartition* of $\mathcal{M}$.

The precise statement of our result is a bit long and is given as Theorem 1.7. We begin with a simpler version, useful when $\mathcal{M}_L$ and $\mathcal{M}_R$ have nearly the same competitive ratio.

THEOREM 1.6. *For any $\epsilon > 0$, there exists a polynomial p such that the following holds. Let $\mathcal{M}$ be a metric space with bipartition $(\mathcal{M}_L, \mathcal{M}_R)$. If $\lambda_{\min}$ and $\lambda_{\max}$ are real numbers such that $\lambda_{\min} \le \lambda(\mathcal{M}_R), \lambda(\mathcal{M}_L) \le \lambda_{\max}$, and each part is $\frac{1}{p(\lambda_{\max})}$-small in $\mathcal{M}$, then*

$$\lambda_{\min} + 1 - \epsilon \le \lambda(\mathcal{M}) \le \lambda_{\max} + 1 + \epsilon.$$

In other words, if $\mathcal{M}_R$ and $\mathcal{M}_L$ both have a competitive ratio close to λ, then the competitive ratio of $\mathcal{M}$ is close to $1 + \lambda$.

The decomposition theorem can be used to estimate the competitive ratio of the PE game on a space by partitioning it into smaller spaces and applying induction. The lower bound of the theorem can be applied to nonbipartite spaces by applying it to a bipartite subspace $\mathcal{N}$ and using $\lambda(\mathcal{N}) \le \lambda(\mathcal{M})$. For example, Theorem 1.4 can be derived as follows. Let $\epsilon = 1/2$ and let $p()$ be the increasing polynomial whose existence is given by Theorem 1.6. Let t be the greatest integer such that $\lceil p(\log n)\rceil^t \le n$ and let $n' = \lceil p(\log n)\rceil^t$. Note that $t = \Omega(\frac{\log n}{\log\log n})$. Let $j = n'/\lceil p(\log n)\rceil = \lceil p(\log n)\rceil^{t-1}$ and let $\mathcal{M}_L$ and $\mathcal{M}_R$ be the leftmost j points and rightmost j points of $\mathcal{M} = \mathcal{L}(n')$, respectively. By choice of j, either $\lambda(\mathcal{M}_L) > \log n$ and we are done, or else the condition on δ of Theorem 1.6 is satisfied. So $\lambda(\mathcal{L}(n)) \ge \lambda(\mathcal{L}(n')) \ge \lambda(\mathcal{M}_L \cup \mathcal{M}_R) \ge \lambda(\mathcal{M}_L) + 1/2$. We can continue on $\mathcal{M}_L$, letting $j' = j/\lceil p(\log n)\rceil$ and so forth until after t steps we have run out of points. The competitive ratio of $\mathcal{L}(n)$ is thus at least $t/2$, which is $\Omega(\frac{\log n}{\log\log n})$.

The full decomposition theorem provides sharp bounds on $\lambda(\mathcal{M})$ even if $\mathcal{M}_L$ and $\mathcal{M}_R$ have different competitive ratios. Define the function $Z(x)$ on nonnegative reals by $Z(x) = x/(e^x - 1)$ if $x > 0$ and $Z(0) = 1$. The theorem says that $\lambda(\mathcal{M})$ is well approximated by $\max\{\lambda(\mathcal{M}_L), \lambda(\mathcal{M}_R)\} + Z(|\lambda(\mathcal{M}_L) - \lambda(\mathcal{M}_R)|)$.

THEOREM 1.7. *Let $\mathcal{M}$ be a metric space with at least three points, having bipartition $(\mathcal{M}_R, \mathcal{M}_L)$. Let $\alpha_R, \alpha_L \ge 0$ and set $\alpha_{\max} = \max\{\alpha_R, \alpha_L\}$ and $\alpha_{\text{diff}} = |\alpha_R - \alpha_L|$. Let $\delta = \delta(\mathcal{M})$ and $\delta_{\max} = \max\{\delta(\mathcal{M}_L), \delta(\mathcal{M}_R)\}$. Suppose that $\alpha_{\max} \ge 1$ and that each part is $1/\max\{324\alpha_{\max}, \alpha_{\max}^3\}$-small in $\mathcal{M}$. Then*

1. *if* $\alpha_L \geq \lambda(\mathcal{M}_L)$ *and* $\alpha_R \geq \lambda(\mathcal{M}_R)$, *then* $\lambda \leq \alpha_{\max} + Z(\alpha_{\text{diff}})(1+\zeta)$, *and*

2. *if* $\alpha_L \leq \lambda(\mathcal{M}_L)$ *and* $\alpha_R \leq \lambda(\mathcal{M}_R)$, *then* $\lambda \geq \alpha_{\max} + Z(\alpha_{\text{diff}})(1-\zeta)$, *where* $\zeta = 23e^{\alpha_{\text{diff}}}\sqrt{\frac{\delta_{\max}}{\delta}\alpha_{\max}^3}$.

The proof of Theorem 1.7 consists of two parts. We introduce and completely analyze a new game, called the *walker-jumper* game, which abstracts the essential elements of the analysis of a decomposed problem. Then we formally demonstrate that the competitive ratio of a decomposed problem can be tightly bounded by the competitive ratio of an associated walker-jumper game.

Theorem 1.6 follows easily by applying the first part of Theorem 1.7 with $\alpha_L = \alpha_R = \lambda_{\max}$ and the second part of the Theorem with $\alpha_L = \alpha_R = \lambda_{\min}$.

Theorem 1.7 is combined with a Ramsey-type theorem for metric spaces to prove Theorem 1.3. The cases of Theorem 1.7 needed are the case where $\mathcal{M}_L$ and $\mathcal{M}_R$ have nearly the same competitive ratio, and the "highly unbalanced" case where $\mathcal{M}_L$ is large and $\mathcal{M}_R$ is a single point. The Ramsey-type theorem, which can be viewed as an extension of a theorem from [KRR], says, roughly, that any metric space of n points must contain at least one of the following *three* objects: (a) a roughly uniform space of around $2^{\sqrt{\log n/\log\log n}}$ points, (b) two highly separated spaces with small diameter, each having around $n/2^{\sqrt{\log n \log\log n}}$ points, or (c) one point very far from a small diameter subspace containing nearly all the rest (around $n - n/2^{\sqrt{\log n/\log\log n}}$ points). (For other Ramsey-like theorems for metric spaces, see [Mat].)

In section 3, we present an informal discussion of the proof to motivate the connection with the walker-jumper game. In section 4 we define the walker-jumper game and state a theorem which gives its exact competitive ratio, and we describe and prove the optimal strategies for each of the two players. The precise statement and proof of the lemma connecting the walker-jumper game to the decomposition theorem is given in section 5. The applications of the decomposition theorem needed to prove Theorems 1.3 and 1.5 are given in section 6, which can be read independently of the previous ones. It requires only the statement of the main decomposition theorem.

Section 5 is long and technical, although the underlying idea as sketched in section 3 is fairly intuitive. Two technical lemmas stated in that section, Lemmas 5.2 and 5.3, provide quantitative bounds on the additive constant that occurs in the definition of competitive ratio, and may be of independent interest. The proofs of these lemmas are deferred to the last section.

2. On-line games: Definitions and preliminary results.

2.1. Notation. As usual, $\mathbb{R}$ and $\mathbb{N}$ denote, respectively, the sets of real numbers and the set of nonnegative integers. The set $\{x \in \mathbb{R} : x \geq 0\} \cup \{\infty\}$ is denoted by $\mathbb{R}_\infty$.

We will need the following notation for sequences. Let X be a set. We denote by X^n the set of sequences consisting of n terms from X and $X^* = \bigcup_{n\geq 1} X^n$. An element of X^* is denoted in boldface as $\mathbf{x} = (x_1, x_2, \ldots, x_t)$; sequences are indexed from 1 unless otherwise noted. The number t of terms is called the *term length* of x and is denoted $|\mathbf{x}|$. If $j \leq t$, then we use $\mathbf{x}^j$ to denote the sequence consisting of the first j terms of $\mathbf{x}$. If $\mathbf{x}$ and $\mathbf{y}$ are sequences such that $\mathbf{x}^j = \mathbf{y}$ for some integer j, then we say that $\mathbf{y}$ is a *prefix* of $\mathbf{x}$ and that $\mathbf{x}$ is an *extension* of $\mathbf{y}$. If $\mathbf{x}$ and $\mathbf{y}$ are sequences, then $\mathbf{xy}$ denotes their concatenation.

If $\mathbf{x}$ is a sequence of real numbers of term length n, then $\mathbf{\Delta x}$ denotes the sequence of differences: $\Delta x_1 = x_1$ and for $2 \leq j \leq n$, $\Delta x_j = x_j - x_{j-1}$.

A metric space $\mathcal{M}$ consists of a set of points P and a symmetric nonnegative valued distance function defined on $P \times P$ that satisfies the triangle inequality and is zero only on the diagonal. We abuse notation and use $\mathcal{M}$ to denote both the metric space and the underlying set of points P. The associated metric is denoted by $d = d_{\mathcal{M}}$. $\mathcal{M}$ is assumed to be a finite set unless otherwise noted. The diameter of the space, $\delta = \delta(\mathcal{M})$, is the maximum distance between any pair of points.

A sequence of points in a metric space is referred to as a *walk* in the space. The domain of the distance function can be viewed as the set of walks of term length 2. We extend the domain of d to the set of all walks by defining $d(\mathbf{x})$ to be the sum of the distances between successive pairs of points. We refer to this as the *metric length* of the walk.

2.2. Two-player zero-sum games. The on-line algorithmic problems considered in this paper can be viewed as two-person zero-sum games. We recall some basic definitions and results about such games. A two-player zero-sum game G with players MAX and MIN is a triple $(S_{\mathrm{MAX}}, S_{\mathrm{MIN}}, C)$, where S_{MAX} and S_{MIN} are sets, and a *cost* function $C = C^G$, where $C : S_{\mathrm{MIN}} \times S_{\mathrm{MAX}} \to \mathbb{R}_\infty$. An element of S_{MAX} (resp., S_{MIN}) is called a *pure strategy* of MAX (resp., MIN). The game is *finite* if S_{MAX} and S_{MIN} are finite sets. It will be convenient to use an asymmetric notation: $C_q(r)$ denotes the value of this function for strategies $q \in S_{\mathrm{MIN}}$ and $r \in S_{\mathrm{MAX}}$. The value $C_q(r)$ is intuitively the cost to MIN given these two strategies.

In a *randomized instance* of the game, each player selects a probability distribution over its strategy set; such a distribution is called a *mixed strategy*. The set of mixed strategies for player X is denoted by $\tilde{S_X}$. Similarly, we denote a mixed strategy by a letter with a $\sim$ over it.

The cost of two mixed strategies $\tilde{q} \in \tilde{S}_{\mathrm{MIN}}$ and $\tilde{r} \in \tilde{S}_{\mathrm{MAX}}$, denoted by $C_{\tilde{q}}(\tilde{r})$, is the expected value of $C_q(r)$ with respect to the product distribution of the two strategies. The *value* of a mixed strategy $\tilde{r} \in \tilde{S}_{\mathrm{MAX}}$, denoted by $V_{\mathrm{MAX}}(\tilde{r})$, is the infimum of $C_{\tilde{q}'}(\tilde{r})$ over all $\tilde{q}' \in \tilde{S}_{\mathrm{MIN}}$ (which may be 0.) Similarly, for $\tilde{q} \in \tilde{S}_{\mathrm{MIN}}$, $V_{\mathrm{MIN}}(\tilde{q})$ is the supremum of $C_{\tilde{q}}(\tilde{r}')$ over all $\tilde{r}' \in \tilde{S}_{\mathrm{MAX}}$ (which may be $+\infty$.)

It is well known (and easy to show) that the value of a mixed strategy is determined by its cost with respect to pure strategies for the other player.

LEMMA 2.1. *Let G be a two-player zero-sum game.*

1. *For any mixed strategy $\tilde{r}$ for MAX, $V_{\mathrm{MAX}}(\tilde{r})$ is equal to the infimum of $C_q(\tilde{r})$ over all pure strategies $q \in S_{\mathrm{MIN}}$.*
2. *For any mixed strategy $\tilde{q}$ for* MIN*, $V_{\mathrm{MIN}}(\tilde{q})$ is equal to the supremum of $C_{\tilde{q}}(r)$ over all pure strategies $r \in S_{\mathrm{MAX}}$.*

The supremum of $V_{\mathrm{MAX}}(\tilde{r})$ over all $\tilde{r} \in \tilde{S}_{\mathrm{MAX}}$ is called the MAX-value of game G, and is denoted by $V_{\mathrm{MAX}}(G)$. A strategy $\tilde{r}$ that attains this supremum is called an *optimal* strategy for MAX. Similarly, the infimum of $V_{\mathrm{MIN}}(\tilde{q})$ over all $\tilde{q} \in \tilde{S}_{\mathrm{MIN}}$ is called the MIN-value of game G, and is denoted by $V_{\mathrm{MIN}}(G)$, and a strategy $\tilde{q}$ that attains this infimum is called an *optimal* strategy for MIN. In general, optimal strategies may exist for one, both, or neither player.

The following elementary result is easily proved.

LEMMA 2.2. *For any game G, $V_{\mathrm{MAX}}(G) \le V_{\mathrm{MIN}}(G)$.*

A game G is said to have the *min-max property* if (i) $V_{\mathrm{MAX}}(G) = V_{\mathrm{MIN}}(G)$ and (ii) both players have an optimal strategy. For such a game, the common value is denoted $V(G)$ and is called the *value* of the game. The following fundamental theorem of two-person games, known as the min-max theorem, was proved by von Neumann (see, e.g., [vNM]).

THEOREM 2.2.1. *Any finite two-player zero-sum game has the min-max property.*

A *subgame* of the game $G = (S_{\text{MAX}}, S_{\text{MIN}}, C)$ is a game $G' = (T_{\text{MAX}}, T_{\text{MIN}}, C)$, where $T_{\text{MAX}} \subseteq S_{\text{MAX}}$ and $T_{\text{MIN}} \subseteq S_{\text{MIN}}$. We say that G' is *equivalent* to G if (i) $V_{\text{MAX}}(G') = V_{\text{MAX}}(G)$, (ii) $V_{\text{MIN}}(G') = V_{\text{MIN}}(G)$, and (iii) MAX (resp., MIN) has an optimal strategy in G' if and only if MAX (resp., MIN) has an optimal strategy in G.

A subset $T_{\text{MAX}} \subseteq S_{\text{MAX}}$ *dominates* S_{MAX} with respect to the game G if for any strategy $r \in S_{\text{MAX}}$, there is a strategy $r' \in T_{\text{MAX}}$ such that $C_q(r') \geq C_q(r)$ for all $q \in S_{\text{MIN}}$. Similarly, a subset $T_{\text{MIN}} \subseteq S_{\text{MIN}}$ dominates S_{MIN} with respect to G if for any strategy $q \in S_{\text{MIN}}$, there is a strategy $q' \in T_{\text{MIN}}$ such that $C_{q'}(r) \leq C_q(r)$ for all $r \in S_{\text{MAX}}$. We have the following proposition.

PROPOSITION 2.1. *Let $G = (S_{\text{MAX}}, S_{MIN}, C)$ be a game, let $T_{\text{MAX}} \subseteq S_{\text{MAX}}$, and let $T_{\text{MIN}} \subseteq S_{\text{MIN}}$.*

1. *If T_{MAX} dominates S_{MAX} relative to G, then the subgame $(T_{\text{MAX}}, S_{\text{MIN}}, C)$ is equivalent to the game G.*
2. *If T_{MIN} dominates S_{MIN} relative to G, then the subgame $(S_{\text{MAX}}, T_{\text{MIN}}, C)$ is equivalent to the game G.*

Finally, we define the notion of the *competitive ratio* of a two-person game. Let G be a game whose payoff function is nonnegative. A *base-cost* function C_{BASE} for the game G is an $\mathbb{R}_\infty$-valued function defined on S_{MAX}. We say that the mixed strategy $\tilde{q}$ for MIN is *κ-competitive with respect to* C_{BASE} if there exists a constant K such that for any strategy r for MAX,

$$C_{\tilde{q}}(r) \leq \kappa C_{\text{BASE}}(r) + K.$$

The *competitive ratio* $\lambda = \lambda(G)$, with respect to C_{BASE}, is the infimum over all κ for which there is a κ-competitive algorithm, with respect to C_{BASE}.

A natural choice for a base-cost function is the *optimal cost function*, $C_{\text{OPT}}(r)$, which is defined as the infimum of $C_q(r)$ over all MIN-strategies q. (Notice that $C_{\text{OPT}}(r) = V_{\text{MAX}}(r)$.) In the context of on-line algorithms, the competitive ratio with base-cost $C_{\text{OPT}}(r)$ corresponds to the standard notion of the *randomized competitive ratio* of the associated on-line problem. We call C_{OPT} the *standard* base-cost function. In this paper, we will also have need to refer to nonstandard base-costs.

The following result gives a criterion for upper bounding the competitive ratio (which is slightly more general than the definition).

PROPOSITION 2.2. *Let G be a game with base-cost function C_{BASE}. Let $\kappa > 0$ and $f : \mathbb{R} \longrightarrow \mathbb{R}$ be a function such that $\lim_{x\to\infty} \frac{f(x)}{x} = \kappa$. Suppose that $\tilde{q}$ is a strategy for* MIN *such that for any strategy r for* MAX*,*

$$C_{\tilde{q}}(r) \leq f(C_{\text{BASE}}(r)).$$

Then $\lambda(G) \leq \kappa$.

Proof. It is easy to check that the hypothesis of the proposition implies that $\tilde{q}$ is $\kappa + \epsilon$ competitive for any positive ϵ, which implies that $\lambda \leq \kappa$. □

The following result gives a criterion for lower bounding the competitive ratio.

PROPOSITION 2.3. *Let G be a game with base-cost function C_{BASE}. Let $\kappa > 0$ and $f : \mathbb{R} \longrightarrow \mathbb{R}$ be a function such that $\lim_{x\to\infty} \frac{f(x)}{x} = \kappa$. Suppose that $\{\tilde{r_i} : i \in \mathbb{N}\}$ is a sequence of mixed strategies for* MAX *and $\{u_i : i \in \mathbb{N}\}$ is a sequence of real numbers tending to ∞, such that for each i, $C_{\text{BASE}}(\tilde{r_i}) \leq u_i$, and for each pure* MIN*-strategy q,*

$$C_q(\tilde{r_i}) \geq f(u_i).$$

Then $\lambda(G) \geq \kappa$.

Proof. It suffices to show that if $\epsilon > 0$, there is no $(\kappa - \epsilon)$-competitive algorithm $\tilde{q}$. Suppose, to the contrary, that $\tilde{q}$ is $(\kappa - \epsilon)$-competitive. Then there is a constant K such that for any pure strategy r

$$C_{\tilde{q}}(r) \leq (\kappa - \epsilon) C_{\mathrm{BASE}}(r) + K.$$

Now by taking expectation with respect to the distribution $\tilde{r}_i$, we get

$$\begin{aligned} C_{\tilde{q}}(\tilde{r}_i) &\leq (\kappa - \epsilon) C_{\mathrm{BASE}}(\tilde{r}_i) + K \\ &\leq (\kappa - \epsilon) u_i + K. \end{aligned}$$

For each i, there is a deterministic strategy q_i such that $C_{q_i}(\tilde{r}_i) \leq C_{\tilde{q}}(\tilde{r}_i)$. For that strategy we have $C_{q_i}(\tilde{r}_i) \leq (\kappa - \epsilon) u_i + K$. Thus by the hypothesis of the proposition, $f(u_i) \leq (\kappa - \epsilon) u_i + K$ for every i. This contradicts the hypothesis that $\lim_{x \to \infty} \frac{f(x)}{x} = \kappa$. □

Taking $u_i = C_{\mathrm{BASE}}(\tilde{r}_i)$ in the above Proposition yields the following corollary.

COROLLARY 2.3. *Let G be a game with base-cost function C_{BASE}. Let $\kappa > 0$ and $f : \mathbb{R} \longrightarrow \mathbb{R}$ be a function such that $\lim_{x \to \infty} \frac{f(x)}{x} = \kappa$. Suppose that $\{\tilde{r}_i : i \in \mathbb{N}\}$ is a sequence of mixed strategies for* MAX *such that $C_{\mathrm{BASE}}(\tilde{r}_i)$ tends to ∞ and such that for each i and for each pure* MIN*-strategy q*

$$C_q(\tilde{r}_i) \geq f(C_{\mathrm{BASE}}(\tilde{r}_i)).$$

Then $\lambda(G) \geq \kappa$.

2.3. The PE game: Definitions and preliminary results. The PE game for a metric space $\mathcal{M}$, denoted PE($\mathcal{M}$), is a two-person zero-sum game between two players, the pursuer (the MAX player) and the evader (the MIN player). Intuitively, the game is played as a sequence of rounds. At all times the evader is located at some point of the space. In each round, the pursuer probes some point of the metric space. If she picks the point containing the evader, then the evader must move to some other point; otherwise, the evader may stay where he is. The cost to the evader in responding to a sequence of probes is the total distance he travels.

In the language of two-player zero-sum games, the set of pure strategies of the pursuer is the set $\mathcal{M}^*$ of all finite sequences from the metric space. Such sequences are referred to as *probe* sequences. A sequence σ is said to be a *response sequence* for the probe sequence ρ if it has the same term length as ρ and $\sigma_i \neq \rho_i$ for all i. The point σ_i represents the location of the evader at time i. The pure strategies for the evader are called *deterministic response algorithms* or, simply, *deterministic algorithms*. A response algorithm A maps each probe sequence ρ to a response sequence $A(\rho)$ subject to the following consistency requirement: for any probe sequence ρ and point a, $A(\rho a)$ extends $A(\rho)$. The consistency requirement formalizes the intuition that the algorithm determines the response sequence of the evader in an on-line manner, i.e., the ith response depends only on the first i probes.

The cost function $C_A(\rho)$ is defined to be $d(A(\rho))$, the metric length of the response sequence generated by the algorithm A on input ρ. $C_{\mathrm{OPT}}(\rho)$ denotes the minimum of $C_A(\rho)$ over all algorithms A. It is easy to see that this is the same as the minimum metric length of a response sequence for ρ. (Note that in our definition, the evader is allowed to choose his own starting point σ_1 at no cost. Other authors specify a starting point σ_0 and charge the evader an additional $d(\sigma_0, \sigma_1)$. This is a matter of convention that does not affect the results.)

A mixed strategy for the pursuer is a probability distribution $\tilde{\rho}$ over probe sequences. A mixed strategy for the evader, called a *randomized response algorithm*, is a probability distribution $\tilde{A}$ over response algorithms. Since the set of pure strategies for the evader is uncountable, a probability distribution cannot be specified simply by assigning a probability value to each strategy. The general approach to mixed strategies on infinite strategy spaces requires measure theory; in our case the measure theoretic definition can be restated in algorithmic terms: in a randomized algorithm the sth move of the evader is chosen according to some probability distribution on $\mathcal{M}$, where the probability distribution may depend on the first s moves by the pursuer and the first $s-1$ responses of the evader.

We can define a randomized response algorithm formally via *decision trees*. Let $T = T_{\mathcal{M}}$ denote the infinite rooted tree of degree $|\mathcal{M}|$ where the edges from each node are in one-to-one correspondence with the points of $\mathcal{M}$ and the nodes are labeled as follows. The root is labeled P (for pursuer), the children of the root are labeled E (for evader), and the remaining nodes are labeled inductively P or E so that labels alternate along each path from the root. We can represent a (randomized) on-line response algorithm $\tilde{A}$ by assigning to each E-node v a probability distribution on $\mathcal{M}$, i.e., a nonnegative function p_v on $\mathcal{M}$ whose values sum to 1. On probe sequence $\rho_1, \rho_2, \ldots, \rho_k$, the algorithm follows the branch labeled ρ_1 from the root. It chooses its response according to the probability distribution p_v for its node, and it follows the corresponding branch from v to the next P node. It then processes each successive probe in the same way, following down the tree to depth $2k$.

As in section 2.2, we extend the definition of the cost function C to randomized strategies by defining $C_{\tilde{A}}(\tilde{\rho})$ to be the expectation of $C_A(\rho)$ with respect to the product distribution of $\tilde{A}$ and $\tilde{\rho}$. Also, $C_{\mathrm{OPT}}(\tilde{\rho})$ is the expectation of $C_{\mathrm{OPT}}(\rho)$ with respect to the distribution $\tilde{\rho}$. (It is important here to emphasize $C_{\mathrm{OPT}}(\tilde{\rho})$ is *not* the same as the minimum of $C_A(\tilde{\rho})$ over all A. In computing the former, we choose the best algorithm for each deterministic ρ and average the cost with respect to $\tilde{\rho}$, while in the latter we choose the one algorithm that minimizes the average cost with respect to $\tilde{\rho}$.)

For $s > 0$, an *s-block* of $\mathcal{M}$ is a prefix-minimal probe sequence whose optimal cost is at least $s-\delta$. In other words, ρ is an s-block if $C_{\mathrm{OPT}}(\rho) \geq s-\delta$, but $C_{\mathrm{OPT}}(\tau) < s-\delta$ for any proper prefix τ of ρ. In the nondegenerate case that $\mathcal{M}$ has at least two points, an s-block ρ satisfies $s - \delta \leq C_{\mathrm{OPT}}(\rho) < s$ since the last step can increase C_{OPT} by at most δ. In the degenerate case that $\mathcal{M}$ consists of a single point p, we define an s-block to be the singleton sequence.

Any probe sequence ρ can be parsed uniquely into subsequences $\rho_{\mathbf{1}}\rho_{\mathbf{2}}\ldots\rho_{\mathbf{k}}$ where each successive $\rho_{\mathbf{i}}$ except possibly the last is an s-block. We refer to this as the *s-block partition* of ρ.

The competitive ratio of $PE(\mathcal{M})$ is defined in terms of the standard base-function C_{OPT} and is denoted by $\lambda(\mathcal{M})$. For brevity, we often refer to this as the *competitive ratio of* $\mathcal{M}$. It is trivial that the competitive ratio of a two-point space is 1. From the previously noted result of [MMS] (which was proved for the deterministic competitive ratio), we have the following lemma.

LEMMA 2.4. *For any* $\mathcal{M}$, $\lambda(\mathcal{M}) \leq |\mathcal{M}| - 1$.

The PE game on a one-point space does not really make sense. However, it will be convenient to adopt the convention that the competitive ratio of a one-point space is 0. With that definition, the main decomposition theorem will hold when one or both spaces is a one-point space.

The following fact is both well known and easy to prove.

PROPOSITION 2.4. *For any metric space $\mathcal{M}$ and subspace $\mathcal{N}$, $\lambda(\mathcal{M}) \geq \lambda(\mathcal{N})$.*

3. An overview of the decomposition theorem. We are working in a space $\mathcal{M}$ with bipartition $(\mathcal{M}_L, \mathcal{M}_R)$, which we call the *left* space and the *right* space. In the present discussion, we assume that each space has at least two points; the degenerate case that one of the spaces consists of a single point will require special treatment, which we delay until later. The assumptions of the theorem imply that the distances within each subspace are small relative to distances between the two subspaces.

We want to express the competitive ratio of the big space in terms of the competitive ratio of each of the two subspaces. The key idea is to abstract the behaviors of the pursuer and the evader so as to focus on their movements *between* the spaces, treating their movements within each space as a "black box." This idea leads to the formulation of a new game, called the walker-jumper game, which abstracts the PE game for such a partitioned space. This game is defined and analyzed in the next section, and the proof of the decomposition theorem is then completed in the following section.

The proof is technical, but the underlying idea is natural. In this section, we provide intuition for the proof with an informal discussion that leads naturally to the definition of the walker-jumper game. Throughout the section we make various plausible but unjustified assumptions and approximations, which will be cleaned up in the proof.

At each point in the game, the evader is either "on the left" or "on the right." While the pursuer probes the opposite space, the evader need do nothing. While the pursuer probes the subspace occupied by the evader, it seems apparent that either the evader should follow his optimal randomized response algorithm for that space (achieving, over that interval of moves, a competitive ratio equal to that for that space) or he should move to the other space. By randomizing his choice of when to switch between spaces, he can hope to "fool" the pursuer as to his location.

We view the probe sequence of the pursuer as a sequence of left phases and right phases, where a left (resp., right) phase consists of moves in the left (resp., right) space. When the evader uses a randomized strategy, the pursuer will only have a probability distribution on the location of the evader. In order to maximize the competitive ratio, the pursuer wants to construct a probe sequence that (i) has a good chance of catching the evader often, and (ii) has a low off-line cost. For the first goal, it would seem that she should always probe on the side with higher probability of containing the evader, while for the second goal, it would seem that she would do well to avoid switching between spaces too often (this will make it easier for an off-line algorithm to "hide" safely in one space for long intervals of moves) and thus should tend to make each phase long.

For $s > 0$, we defined an s-block to be a prefix-minimal probe sequence of cost at least $s - \delta$, and we observed that its cost is at most s. For some large integer D (to be specified later) we define $s = \delta/D$, where δ is the diameter of $\mathcal{M}$, and define a *left block* (resp., *right block*) to be a probe sequence which is an s-block with respect to the space $\mathcal{M}_L$ (resp., $\mathcal{M}_R$). Note that an s-block for $\mathcal{M}_L$ or $\mathcal{M}_R$ is not an s-block with respect to the entire space; indeed, any such block has 0 optimal cost with respect to the entire space since an off-line evader will respond by staying at one location on the opposite space.

Recall that the s-block partition provides a canonical way to parse every probe sequence from a space as a concatenation of sequences, each of which, except possibly the last, is an s-block. For a given probe sequence of $\mathcal{M}$, parse each left phase

according to its s-block partition with respect to $\mathcal{M}_L$ and parse each right phase similarly. It is reasonable to expect that if s is small relative to the typical cost of a phase, then we may ignore the "remainder" block of the phase and simply assume that each left (resp., right) phase is a concatenation of left (resp., right) blocks.

With this assumption, we view the entire probe sequence as a sequence of left and right blocks. If the pursuer chooses to add a right block, it would seem natural that she select the right block so that, assuming the evader is on the right, the expected cost to the evader is maximized. This expected cost cannot exceed $s\lambda_R$ (by much) since the evader can follow his λ_R-competitive strategy for the space $\mathcal{M}_R$. On the other hand, the pursuer can force an evader who stays on the right to incur nearly this much (in expectation) by selecting her right block according to her optimal randomized s-block strategy $\tilde{\rho}_{\mathbf{s}} = \tilde{\rho}_{\mathbf{s}}(\mathcal{M}_R)$. Similarly, when the pursuer picks a left block, she can force an evader who stays on the left to pay roughly $s\lambda_L$.

To summarize, when the pursuer adds a block, if the evader is on the opposite side, he pays nothing. If the evader is on the same side, he either moves to the other side immediately, paying roughly δ, or he stays on the original side and incurs an expected cost of $s\lambda_R$ or $s\lambda_L$ depending on the side. We assume that δ is much larger than both $s\lambda_R$ and $s\lambda_L$ so that it would not pay for him to move to the opposite side at the beginning and back at the end of the block.

Thus we have a good approximation to the cost of each block to the evader, depending only on (i) the side from which each block is chosen by the pursuer, and (ii) the side on which the evader finishes responding to each block.

We would like to get a similar estimate for the off-line cost. For each probe sequence ρ, define $C_L(\rho)$ to be the minimum cost of a response sequence σ whose last point is on the left and $C_R(\rho)$ in the analogous way. We refer to these, respectively, as the left-optimal and right-optimal cost of ρ. The optimal cost of ρ is just the minimum of these. Since they clearly differ by at most the diameter δ of $\mathcal{M}$, we may take $C_R(\rho)$ as a good estimate of $C_{\mathrm{OPT}}(\rho)$. We want to understand how C_L and C_R change when the pursuer adds a left block or a right block. So let ρ be the sequence constructed so far and consider adding a right block β. It is easy to see that $C_L(\rho\beta) = C_L(\rho)$, as $C_L(\rho\beta) = \min\{C_L(\rho), C_R(\rho)+\delta\}$ and $C_L(\rho) \le C_R(\rho)+\delta$. On the other hand, we can estimate $C_R(\rho\beta) \approx \min\{C_L(\rho)+\delta, C_R(\rho)+s\}$, where the two terms correspond to the two choices of the off-line evader: either finish the previous block on the left and move right only at the end of β or finish the previous block on the right and respond to β on the right.

Let us summarize this discussion by considering the evolution of the parameter $w = (C_R - C_L)/s$. Note that w is always between $-D$ and D. Each right block increases w by (roughly) 1, subject to $w \le D$ and, similarly, each left block decreases w by 1, subject to $w \ge -D$. It is useful to visualize the evolution of w as a walk on the integer points between $-D$ and D. A right step corresponds to w being increased by 1, and C_R being increased by s. Similarly, a left step corresponds to w being decreased by 1, and C_L being decreased by s.

Notice that if the pursuer adds a right block that causes w to reach D, then the pursuer may add an arbitrary number of right blocks without affecting C_L, C_R, or w. Let us assume that she does this, i.e., a right step from $D-1$ to D corresponds to a huge number of right blocks. The effect of this on the evader is clear: on such a step he must move to the left space (if he is not there already) or incur a huge cost. (Note that the evader can compute C_R, C_L, and w on-line and thus recognize this situation). Similarly, we may assume that a left step from $1-D$ to $-D$ corresponds

to a huge number of left blocks and the evader must move to the right space.

Having associated the pursuer's probe sequence to a walk on the integer line, we now can make the following estimates of the off-line cost and the evader's cost. The off-line cost is estimated by s times the number of steps to the right. The evader's cost is 0 on any step that is taken in the direction opposite the space he occupies. On a step taken in the direction of the space that he occupies, his cost is sD if he chooses to move to the other space and is $s\lambda_R$ or $s\lambda_L$ (depending on his space) if he chooses to stay in his space. Whenever a right (resp., left) step is taken that reaches D (resp.,$-D$), the evader must move to the left (resp., right) space.

This idealization suggests that we can model our problem by a game between two players, the *walker* who walks on the line (and corresponds to the pursuer) and the *jumper* who jumps between the left and right side (and corresponds to the evader). In the next section we define this game precisely and analyze it. In the succeeding section, we then formally show how to use the analysis of this game to establish the decomposition theorem.

4. Walker-jumper games. The walker-jumper game $WJ[D, \alpha_R, \alpha_L]$ has parameters D, a positive integer, and nonnegative real numbers α_R and α_L. The players are referred to as the walker (Wendy) and the jumper (Jack). The game "board" is the set $I_D = \{-D, -D+1, \ldots, D-1, D\}$. At each integer time $t \geq 0$, the position of the game is the ordered pair (w_t, j_t), where $w_t \in I_D$ is Wendy's position and $j_t \in \{-D, D\}$ is Jack's location. The initial position for Wendy is $w_0 = 0$, and Jack can choose either $j_0 = -D$ or $j_0 = D$.

At each time t, a legal move for Wendy consists of one step either to the left ($w_t = w_{t-1} - 1$) or to the right ($w_t = w_{t-1} + 1$). If $|w_t| = D$, Wendy has only one legal move. Thus the sequence $\mathbf{\Delta w}$ defined by $(\Delta w)_i = w_i - w_{i-1}$ has entries in the set $\{-1, 1\}$. We refer to Δw_i as the *direction* of move i. A move of -1 (resp., $+1$) will be referred to as a left move (resp., right move). Also, for convenience of notation, we sometimes use α_{-1}, α_1 in place of α_L, α_R.

Jack's answer to request w_t is either to stay where he is ($j_t = j_{t-1}$) or to *jump* to his other allowed position ($j_t = -j_{t-1}$). Jack's moves are subject to the constraint that $j_t \neq w_t$, i.e., if Wendy arrives at Jack's location ($w_t = j_{t-1}$), then Jack must jump ($j_t = -w_t$).

If $j_t = (\Delta w)_t D$, i.e., Wendy's move at time t brought her closer to the location that Jack occupies *after* his move, then we say that Wendy *hit* Jack; it is a *left hit* or a *right hit* depending on the direction of Wendy's move.

Formally, a (pure) strategy for Wendy (a request sequence) is given by a sequence $\mathbf{w} = (w_0 = 0, w_1, w_2, \ldots)$ having entries in I_D and satisfying $|w_t - w_{t-1}| = 1$ for each $t > 0$. A pure strategy for Jack is a function (algorithm) A that maps each finite request sequence $\mathbf{w} = (0, w_1, \ldots, w_s)$ to a sequence $(j_0, j_1, j_2, \ldots j_s)$ in $\{-D, D\}^{s+1}$. The map satisfies the constraint $A(\mathbf{w})_i \neq w_i$ for each i, and it also satisfies the consistency constraint that if $\mathbf{w}$ is an extension of $\mathbf{v}$, then $A(\mathbf{w})$ is an extension of $A(\mathbf{v})$; this constraint means that A is an on-line algorithm. As usual, we also consider randomized strategies for both players. A randomized strategy for Wendy is a probability distribution $\tilde{\mathbf{w}}$ on request sequences and a randomized strategy for Jack is a probability distribution $\tilde{A}$ on algorithms.

The cost function for algorithm A on request sequence $\mathbf{w}$, $C_A(\mathbf{w})$ (representing the cost to Jack), is given as follows.

1. Each jump by Jack costs him D.
2. Each right hit by Wendy costs Jack α_R.

3. Each left hit by Wendy costs Jack α_L.

Thus the cost of step t to Jack, $(\Delta C_A(\mathbf{w}))_t = C_A(\mathbf{w^t}) - C_A(\mathbf{w^{t-1}})$, can be written as

$$(\Delta C_A(\mathbf{w}))_t = D\chi(j_t = -j_{t-1}) + \alpha_{((\Delta w)_t)}\chi(j_t = (\Delta w)_t D),$$

where for the predicate P, $\chi(P) = 1$ if P is true and 0 otherwise.

As usual, if $\tilde{A}$ and $\tilde{\mathbf{w}}$ are randomized strategies, then $C_{\tilde{A}}(\tilde{\mathbf{w}})$ is defined to be the expectation of $C_A(\mathbf{w})$ with respect to the distributions.

We are interested in the competitive ratio $\lambda = \lambda(WJ[D, \alpha_R, \alpha_L])$ of this game with respect to a nonstandard base-cost function: $C_{\mathrm{BASE}}(\mathbf{w})$ is the total number of steps to the right. Note that $C_{\mathrm{BASE}}(\mathbf{w})$ is within $\pm D$ of $|\mathbf{w}|/2$. Applying Proposition 2.2, we obtain the following criterion for upper bounding λ.

COROLLARY 4.1. *Let $\tilde{A}$ be an algorithm for Jack. Suppose that b is a positive real such that for all $j \in \mathbb{N}$ and sequences $\mathbf{w}$ of term length j*

$$C_{\tilde{A}}(\mathbf{w}) \le j(b + g(j)),$$

where $g(j)$ is a function that tends to 0 as j tends to ∞. Then $\lambda \le 2b$.

Proof. Let $g'(x) = \max_{\{d:-D\le d\le D\}} g(x+d)$. The hypothesis implies that for any $\mathbf{w}$, $C_{\tilde{A}}(\mathbf{w}) \le (C_{\mathrm{BASE}}(\mathbf{w}) + D/2)(b + g'(C_{\mathrm{BASE}}(\mathbf{w})))$. So the corollary follows if we take $f(x) = (x + D/2)(b + g'(x))$, and $c = 2b$ in Proposition 2.2. □

Similarly, we get a criterion for lower bounding λ from Corollary 2.3. Recall that $\mathbf{w^j}$ denotes the prefix of $\mathbf{w}$ up to w_j.

COROLLARY 4.2. *Let $g(j) \to 0$ as $j \to \infty$. Let $\tilde{\mathbf{w}}$ be a distribution over infinite sequences for Wendy. Suppose that b is a positive real such that for any algorithm A for Jack and $j \in \mathbb{N}$,*

$$C_A(\tilde{\mathbf{w}}^{\mathbf{j}}) \ge j(b - g(j)).$$

Then $\lambda \ge 2b$.

An algorithm A for Jack is called *lazy* if Jack never jumps when Wendy moves away from him, i.e., if $(\Delta w)_t \ne j_{t-1}/D$, then $j_t = j_{t-1}$. It is easy to show that any nonlazy strategy is dominated by some lazy one in the sense that the lazy strategy performs at least as well on any request sequence by Wendy. Thus we may assume that Jack is restricted to lazy strategies.

We consider only the case that D is at least $\alpha_{\max} = \max\{\alpha_R, \alpha_L\}$. To develop some intuition for this game, let us first show that $\alpha_{\max} \le \lambda \le \alpha_L + \alpha_R + 1$.

To see the lower bound assume, without loss of generality, that $\alpha_L \ge \alpha_R$ and consider the following pure strategy for Wendy: move right D times to position D, and then, alternately, move between $D-1$ and D. Each time Wendy moves from D to $D-1$, Jack must start at $-D$ and pays $\alpha_{\max}$ (if he doesn't move) or D (if he does). After j steps by Wendy, Jack's cost is at least $\alpha\frac{j-D}{2} = j(\frac{\alpha_{\max}}{2} - \frac{D\alpha_{\max}}{2j})$. Applying Corollary 4.2 yields a lower bound of $\alpha_{\max}$ on the competitive ratio.

To see the upper bound of $\alpha_L + \alpha_R + 1$, consider the following pure strategy for Jack: never jump unless a jump is required (because Wendy arrives at the same location). On any j step sequence, Wendy takes at most $j/2 + D$ steps in each direction (since the number of left steps differs from the number of right steps by at most D) and Jack jumps at most $j/(2D)$ times (since he jumps at most once every $2D$ steps). Thus Jack's cost can be bounded above by $(\alpha_R + \alpha_L)(j/2 + D) + j/2$.

Applying Corollary 4.1 implies an upper bound of $\alpha_R + \alpha_L + 1$ on the competitive ratio.

As we shall see, the trivial lower bound above is much closer to the truth than the trivial upper bound. The main result of this section is an exact expression for the competitive ratio $\lambda(WJ[D, \alpha_R, \alpha_L])$. Define $\beta_R = 1 - \alpha_R/2D$ and $\beta_L = 1 - \alpha_L/2D$.

THEOREM 4.2.1. *For nonnegative real numbers α_R and α_L and positive integer $D \geq max\{\alpha_L, \alpha_R\}$, the competitive ratio λ of the game $WJ[D, \alpha_R, \alpha_L]$ is given by*

$$\lambda = \begin{cases} \alpha_R + \beta_R & \text{if } \alpha_R = \alpha_L, \\ \frac{\alpha_R \beta_L^{2D} - \alpha_L \beta_R^{2D}}{\beta_L^{2D} - \beta_R^{2D}} & \text{if } \alpha_R \neq \alpha_L. \end{cases}$$

Using elementary estimates and recalling that we defined $Z(x) = x/(e^x - 1)$ for $x > 0$ and $Z(0) = 1$, we obtain the following corollary.

COROLLARY 4.3. *For nonnegative real numbers α_R and α_L, define $\alpha_{\max} = \max\{\alpha_R, \alpha_L\}$ and $\alpha_{\text{diff}} = |\alpha_R - \alpha_L|$. Suppose that $\alpha_{\max} \geq 1$. Let $D \geq \alpha_{\max}^2$ be a positive integer. Then the competitive ratio λ of $WJ[D, \alpha_R, \alpha_L]$ satisfies*

$$\alpha_{\max} + Z(\alpha_{\text{diff}})(1 - \epsilon) \leq \lambda \leq \alpha_{\max} + Z(\alpha_{\text{diff}})(1 + \epsilon),$$

where $\epsilon = \frac{4\alpha_{\max}^2}{D}$.

We first deduce the corollary from the theorem.

Proof of Corollary 4.3. The case $\alpha_R = \alpha_L$ is trivial since $Z(0) = 1$. So assume $\alpha_R \neq \alpha_L$. From the theorem and simple algebraic manipulation, we get

$$\lambda = \frac{\alpha_R \beta_L^{2D} - \alpha_L \beta_R^{2D}}{\beta_L^{2D} - \beta_R^{2D}} = \alpha_{\max} + \frac{\alpha_{\text{diff}}}{(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D} - 1}.$$

To prove the corollary, we need to show that for $\epsilon = 4\alpha_{\max}^2/D$

$$\left| \frac{1}{(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D} - 1} - \frac{1}{e^{\alpha_{\text{diff}}} - 1} \right| \leq \frac{\epsilon}{e^{\alpha_{\text{diff}}} - 1},$$

or, equivalently,

$$\left| \frac{(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D} - e^{\alpha_{\text{diff}}}}{(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D} - 1} \right| \leq \epsilon.$$

Note that $(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D} \geq e^{\alpha_{\text{diff}}}$ by taking $x = (2D - \alpha_{\text{diff}})/\alpha_{\text{diff}}$ in the inequality $(1 + 1/x)^{x+1} \geq e$, which holds for all positive x. Thus we may remove the absolute value from the inequality to be proved. Replacing the denominator by a smaller quantity, it now suffices to show that

$$\frac{(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D} - e^{\alpha_{\text{diff}}}}{e^{\alpha_{\text{diff}}} - 1} \leq \epsilon.$$

Next we upper bound $(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})^{2D}$:

$$\begin{aligned} \left(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}}\right)^{2D} &= e^{2D \ln(1 + \frac{\alpha_{\text{diff}}}{2D - \alpha_{\max}})} \leq e^{\frac{2D\alpha_{\text{diff}}}{2D - \alpha_{\max}}} \leq e^{\alpha_{\text{diff}}} e^{\frac{\alpha_{\text{diff}}\alpha_{\max}}{2D - \alpha_{\max}}} \\ &\leq e^{\alpha_{\text{diff}}} e^{\frac{\alpha_{\text{diff}}\alpha_{\max}}{D}} \leq e^{\alpha_{\text{diff}}} \left(1 + \frac{2\alpha_{\text{diff}}\alpha_{\max}}{D}\right). \end{aligned}$$

(The last inequality uses the assumption $\alpha_{\max}^2 \le D$.) Thus it suffices to show

$$\frac{e^{\alpha_{\text{diff}}} 2\alpha_{\text{diff}}\alpha_{\max}/D}{e^{\alpha_{\text{diff}}}-1} \le \epsilon.$$

If $e^{\alpha_{\text{diff}}} \le 2$, then since the denominator is at least α_{diff}, the expression on the left is at most $4\alpha_{\max}/D$. If $e^{\alpha_{\text{diff}}} > 2$, then $e^{\alpha_{\text{diff}}}/(e^{\alpha_{\text{diff}}}-1) < 2$, and the expression is at most $4\alpha_{\text{diff}}\alpha_{\max}/D \le 4\alpha_{\max}^2/D$. □

We now return to the proof of Theorem 4.2.1. This is proved in two parts.

1. We analyze a specific randomized algorithm for Jack and use Corollary 4.1 to obtain an upper bound on λ.
2. We analyze a specific randomized strategy for Wendy (a probability distribution on request sequences) and use Corollary 4.2 to obtain a lower bound on λ.

4.1. The upper bound: A strategy for the jumper. We begin with an explicit description of a randomized strategy for Jack. The strategy is simple but not particularly intuitive; we will motivate it as we analyze it.

Jumper strategy.

1. The strategy is defined in terms of $2D+1$ parameters $0 = p_{-D} < p_{-D+1} < p_{-D+2} < \cdots < p_{D-1} < p_D = 1$, which are specified below. Initially, Jack chooses his initial position to be $-D$ with probability p_0. At round $t+1$, if Wendy moves in the direction away from j_t, Jack does not move. If $j_t = D$ and Wendy moves rightward from $w_t = j-1$ to $w_{t+1} = j$, then Jack moves to $-D$ with probability $\frac{p_j - p_{j-1}}{1-p_{j-1}}$. The probability of jumping from D to $-D$ is chosen so that if Jack occupied $-D$ with probability p_{j-1}, then Jack is now at $-D$ with probability $p_{j-1} + (1-p_{j-1})\frac{p_j - p_{j-1}}{1-p_{j-1}} = p_j$. If $j_t = -D$ and Wendy moves leftward from $w_t = j+1$ to $w_{t+1} = j$, then Jack moves to D with probability $\frac{p_{j+1}-p_j}{p_{j+1}}$. Here, the probability of jumping from $-D$ to D is chosen so that if Jack occupied $-D$ with probability p_{j+1}, then Jack is now at $-D$ with probability $p_{j+1}(1 - \frac{p_{j+1}-p_j}{p_{j+1}}) = p_j$.
   Notice that this strategy ensures that at all times t, Jack is at $-D$ with probability p_{w_t}: at the start, Wendy is at location 0 and Jack is at $-D$ with probability p_0; and, as noted above, this property is maintained inductively.
2. The parameters p_i that are used are given by

$$p_i = \begin{cases} \frac{1}{2} + \frac{i}{2D} & \text{if } \alpha_R = \alpha_L, \\ \frac{\beta_L^i \beta_R^{2D-i} - \beta_R^{2D}}{\beta_L^{2D} - \beta_R^{2D}} & \text{if } \alpha_R \ne \alpha_L. \end{cases}$$

For a given sequence $\mathbf{w}$ for Wendy, let $N_{j,j'} = N_{j,j'}(\mathbf{w})$ be the total number of steps that Wendy takes from j to j' (which can be nonzero only if $j' \in \{j-1, j+1\}$). Recall that each right hit costs Jack α_R, each left hit costs Jack α_L, and each jump costs Jack D. So each time Wendy takes a step from j to $j+1$, Jack pays α_R with probability $1-p_{j+1}$ (the probability Jack is at D after the step is taken) and pays D with probability $p_{j+1} - p_j$. Similarly, each time Wendy moves from $j+1$ to j, Jack's expected cost is $p_j\alpha_L + (p_{j+1} - p_j)D$. Therefore, we can express the expected cost to Jack of the sequence $\mathbf{w}$ as

$$C_{\tilde{A}}(\mathbf{w}) = \sum_{j=-D}^{D-1} N_{j,j+1}((1-p_{j+1})\alpha_R + (p_{j+1}-p_j)D) + N_{j+1,j}(p_j\alpha_L + (p_{j+1}-p_j)D)$$
$$\leq \sum_{j=-D}^{D-1} (N_{j,j+1}+1)((1-p_{j+1})\alpha_R + p_j\alpha_L + 2(p_{j+1}-p_j)D).$$

The inequality follows from the fact that $N_{j,j+1}+1 \geq N_{j+1,j}$.

The expression $r_j = (1-p_{j+1})\alpha_R + p_j\alpha_L + 2(p_{j+1}-p_j)D$ that multiplies $N_{j,j+1}$ can be interpreted as the expected cost to Jack if Wendy makes a "round trip" from j to $j+1$. The specific definition of p_j given in Jack's strategy was chosen so that r_j is the same for all j. These values can be determined by introducing a parameter K, setting the $r_j = K$ for each j, solving the resulting linear recurrence relation for p_j, and then using $\sum p_j = 1$ to determine K. As is easily verified, this yields

$$K = \begin{cases} \alpha_R + \beta_R & \text{if } \alpha_R = \alpha_L, \\ \frac{\alpha_R\beta_L^{2D} - \alpha_L\beta_R^{2D}}{\beta_L^{2D} - \beta_R^{2D}} & \text{if } \alpha_R \neq \alpha_L. \end{cases}$$

As noted, $\sum_{j=-D}^{D-1} N_{j,j+1} \leq \frac{1}{2}|\mathbf{w}| + D$. This leads to

$$C_{\tilde{A}}(\mathbf{w}) \leq \sum_{j=-D}^{D-1} (N_{j,j+1}+1)r_j$$
$$\leq K \sum_{j=-D}^{D-1} (N_{j,j+1}+1)$$
$$\leq \frac{1}{2}|\mathbf{w}|K + 3DK.$$

Applying Corollary 4.1 yields $\lambda \leq K$, as desired.

4.2. The lower bound: A strategy for the walker. We now prove a matching lower bound by describing and analyzing a randomized strategy for Wendy. As for the upper bound, the strategy is simple; we describe it first and motivate it as we analyze it. Essentially, Wendy's strategy is to follow a biased random walk on the line, where the bias to the left or right depends on the direction taken in the previous step.

Walker strategy.

1. The strategy is defined in terms of two parameters σ_R and σ_L, which are real numbers between 0 and 1. If $w_t = D$ or $w_t = -D$, then w_{t+1} is forced. Otherwise, $|w_t| < D$ and Wendy moves as follows. If move t was to the right ($w_t = w_{t-1}+1$), then Wendy goes left at step $t+1$ with probability σ_L and to the right with probability $1-\sigma_L$. Similarly, if move t was to the left ($w_t = w_{t-1}-1$), then Wendy goes right at step $t+1$ with probability σ_R and to the left with probability $1-\sigma_R$.
2. The parameters σ_R and σ_L are defined by $\sigma_R = \alpha_L/2D = 1-\beta_L$ and $\sigma_L = \alpha_R/2D = 1-\beta_R$.

The intuition behind this strategy is the following. We want to choose a strategy for Wendy that guarantees that the ratio of Jack's cost to the base-cost is at least a certain value, regardless of what Jack's algorithm is. This suggests that we seek a strategy that has the property that the expected cost to Jack is essentially independent of what Jack does. This was the guiding principle in designing this strategy.

Fix a lazy deterministic strategy A for Jack and define $C(\mathbf{w}) = C_A(\mathbf{w})$. We are interested in lower bounding the expectation of $C(\mathbf{w})$ with respect to the above distribution for Wendy. It will be convenient to introduce a modified cost function, $C^*(\mathbf{w}) = C(\mathbf{w}) + \psi(\mathbf{w})$, where the correction ψ depends only on the final step: if the last step by Wendy scored a hit on Jack (precisely, $j_s = D(w_s - w_{s-1})$), then $\psi(\mathbf{w}) = D - \alpha_R$ if the hit was to the right and $D - \alpha_L$ if the hit was to the left. Otherwise, $\psi(\mathbf{w}) = 0$. Thus $C(\mathbf{w}) \geq C^*(\mathbf{w}) - D$.

The purpose of introducing this modified cost is that with respect to this cost measure, the cost to Jack of any given step does not depend on what Jack does at that step. More precisely, define Jack's *modified cost at step* t to be $(\Delta C^*)_t(\mathbf{w}) = C^*(\mathbf{w^t}) - C^*(\mathbf{w^{t-1}})$. Since we are assuming that Jack is following a lazy strategy, Jack has an option to move only if $j_{t-1} = (\Delta w)_t D$. In this case, if he does not jump, then his true cost increases by $\alpha_{(\Delta w)_t}$, while his modified cost increases by $\alpha_{(\Delta w)_t} + \psi(\mathbf{w^t}) - \psi(\mathbf{w^{t-1}}) = D - \psi(\mathbf{w^{t-1}})$. If he jumps at time t, then his true cost goes up by D, but his modified cost goes up by $D + \psi(\mathbf{w^t}) - \psi(\mathbf{w^{t-1}}) = D - \psi(\mathbf{w^{t-1}})$, which is the same as if he did not jump. The reader can now check the following lemma.

LEMMA 4.4. *For any fixed lazy strategy A for Jack, the change $(\Delta C^*)_t(\mathbf{w})$ in the modified cost at time t is given by the following table.*

$(\Delta w)_t$	$(\Delta w)_{t-1}$	j_{t-1}	$(\Delta C^*)_t(\mathbf{w})$
$+1$	$+1$	$+D$	α_R
$+1$	$+1$	$-D$	0
-1	$+1$	$+D$	$\alpha_R - D$
-1	$+1$	$-D$	D
-1	-1	$-D$	α_L
-1	-1	$+D$	0
$+1$	-1	$-D$	$\alpha_L - D$
$+1$	-1	$+D$	D

Recall that if $|w_{t-1}| = D$, then by the rules of the game, it must be the case that $j_{t-1} = -w_{t-1}$, $(\Delta w)_{t-1} = w_{t-1}/D$, and $(\Delta w)_t = j_{t-1}/D$. Thus in this case, $(\Delta C^*)_t(\mathbf{w}) = D$.

If $|w_{t-1}| < D$, then when Wendy moves at time t, her move can depend on $(\Delta w)_{t-1}$, the direction of her last move, but not on j_{t-1}, which she does not know. So we try a strategy for Wendy in which her probability of moving in each direction depends on the direction of her last move. This motivates condition 1 in the definition of the strategy. So consider a strategy satisfying this condition, with σ_R and σ_L as yet unspecified. We now can write down an expression in terms of w_{t-1} and j_{t-1} for the expectation (with respect to this randomized strategy) in the change of the modified cost at time t for the case $|w_{t-1}| < D$.

$$(\Delta C^*)_t(\tilde{\mathbf{w}}) = \begin{cases} \alpha_R - D\sigma_L & \text{if } ((\Delta w)_{t-1}, j_{t-1}) = (+1, +D), \\ D\sigma_L & = (+1, -D), \\ \alpha_L - D\sigma_R & = (-1, -D), \\ D\sigma_R & = (-1, +D). \end{cases}$$

By selecting σ_L and σ_R so that this expectation is independent of j_{t-1}, we obtain condition 2 of the strategy.

Having motivated Wendy's strategy, we now continue with its analysis. The change in the expected modified cost at time t can now be written as

$$(\Delta C^*)_t(\tilde{\mathbf{w}}) = \begin{cases} D & \text{if } |w_{t-1}| = D, \\ \alpha_R/2 & \text{if } |w_{t-1}| < D \text{ and } (\Delta w)_{t-1} = +1, \\ \alpha_L/2 & \text{if } |w_{t-1}| < D \text{ and } (\Delta w)_{t-1} = -1. \end{cases}$$

To apply Corollary 4.2, we need to lower bound Jack's expected cost against the sequence $\mathbf{w}^j$ generated by the first j steps of Wendy's strategy.

Wendy's strategy can be described by a Markov chain with state space $\{L_i : -D \le i < D\} \bigcup \{R_i : -D < i \le D\}$, where Wendy is in state L_i at time $t-1$ if $w_{t-1} = i$ and $(\Delta_w)_{t-1} = -1$ (she is at point i and her last move was to the left) and Wendy is in state R_i at time $t-1$ if $w_{t-1} = i$ and $(\Delta_w)_{t-1} = +1$ (she is at point i and her last move was to the right). For state U, let $N_j(U)$ denote the expected number of visits to state U during the first j steps by Wendy. Also, let $p(U)$ denote the steady state probability for state U. For large j, $N_j(U) = p(U)j(1 + o(1))$. Thus for a sequence $\tilde{\mathbf{w}}$ of j steps chosen according to Wendy's strategy,

$$\begin{aligned} C^*(\tilde{\mathbf{w}}) &= D[N_j(L_{-D}) + N_j(R_D)] + \frac{1}{2} \sum_{i=1-D}^{D-1} [N_j(L_i)\alpha_L + N_j(R_i)\alpha_R] \\ &= j(1 + o(1)) \left[D(p(L_{-D}) + p(R_D)) + \frac{1}{2} \sum_{i=1-D}^{D-1} (p(L_i)\alpha_L + p(R_i)\alpha_R) \right]. \end{aligned}$$

Applying Corollary 4.2, we obtain

$$\lambda \ge 2 \left[D(p(L_{-D}) + p(R_D)) + \frac{1}{2} \sum_{i=1-D}^{D-1} (p(L_i)\alpha_L + p(R_i)\alpha_R) \right].$$

We proceed to determine the steady state probabilities of the Markov chain. The transition matrix of the chain yields the following equations for the steady state probabilities:

$$\begin{aligned} p(L_i) &= (1 - \sigma_R)p(L_{i+1}) + \sigma_L p(R_{i+1}) \quad \text{if } -D \le i < D-1, \\ p(L_{D-1}) &= p(R_D), \\ p(R_i) &= (1 - \sigma_L)p(R_{i-1}) + \sigma_R p(L_{i-1}) \quad \text{if } 1-D < i \le D, \\ p(R_{1-D}) &= p(L_{-D}). \end{aligned}$$

Solving this system and recalling that $\beta_R = 1 - \alpha_R/2D = 1 - \sigma_L$ and $\beta_L = 1 - \alpha_R/2D = 1 - \sigma_R$, we get the following solution (which can be easily checked against the defining equations):

$$p(L_i) = \frac{J}{4D} \left(\frac{\beta_R}{\beta_L} \right)^i, \tag{4.1}$$

$$p(R_i) = \frac{J}{4D} \left(\frac{\beta_R}{\beta_L} \right)^{i-1}, \tag{4.2}$$

where

$$J = \begin{cases} 1 & \text{if } \alpha_R = \alpha_L, \\ \frac{(\alpha_R - \alpha_L)(\beta_L \beta_R)^D}{\beta_L(\beta_L^{2D} - \beta_R^{2D})} & \text{if } \alpha_R \ne \alpha_L \end{cases}$$

is chosen so that the sum of the probabilities is 1. Notice that $p(L_i) = p(R_{i+1})$ for all i and therefore $\sum_i p(L_i) = \sum_i p(R_{i+1}) = 1/2$. Thus we can rewrite the lower bound on λ as

$$\begin{aligned}\lambda &= 2\left[D(p(L_{-D}) + p(R_D)) + \left(\frac{1}{2} - p(L_{-D})\right)\frac{\alpha_L}{2} + \left(\frac{1}{2} - p(R_D)\right)\frac{\alpha_R}{2}\right] \\ &= 2\left[D(\beta_L p(L_{-D}) + \beta_R p(R_D)) + \frac{\alpha_L + \alpha_R}{4}\right] \\ &= \frac{1}{2}\left[\left(\frac{J(\beta_L^{2D} + \beta_R^{2D})}{(\beta_L^{D-1}\beta_R^D)}\right) + \alpha_L + \alpha_R\right].\end{aligned}$$

A routine calculation shows that this simplifies to the expression in the theorem. ∎

5. Proof of the decomposition theorem. This section is devoted to the proof of Theorem 1.7. Recall the notation of the theorem: $\mathcal{M}$ is a metric space partitioned into subspaces $\mathcal{M}_L$ and $\mathcal{M}_R$. Their respective diameters and competitive ratios are denoted $\delta, \delta_L, \delta_R$ and $\lambda, \lambda_L, \lambda_R$. Also $\delta_{\max} = \max\{\delta_L, \delta_R\}$, $\lambda_{\max} = \max\{\lambda_L, \lambda_R\}$, and $\lambda_{\text{diff}} = |\lambda_L - \lambda_R|$.

The overview of the proof in section 3 developed the walker-jumper game as a rough model for the PE game on a partitioned space. We now make this connection precise.

LEMMA 5.1. *Let $\mathcal{M}_L, \mathcal{M}_R$ be a partition of $\mathcal{M}$ such that $\frac{\delta}{\delta_{\max}}$ is at least 32. Let α_R and α_L be nonnegative numbers and $\alpha_{\max}$ be their maximum. Let D be an integer satisfying* $\max\{2\lambda_{\max} + 2, \sqrt{\frac{\delta}{\delta_{\max}}}\} \le D \le \frac{\delta}{4\delta_{\max}}$.

1. *If $\alpha_L \ge \lambda_L$ and $\alpha_R \ge \lambda_R$, then*

$$\lambda \le \lambda(WJ[D, \alpha_R, \alpha_L])(1 + \eta).$$

2. *If $\alpha_L \le \lambda_L$ and $\alpha_R \le \lambda_R$, then*

$$\lambda \ge \lambda(WJ[D, \alpha_R, \alpha_L])(1 - \eta),$$

where

$$\eta \le 6\frac{\delta_{\max} D}{\delta}.$$

Given Lemma 5.1 and the analysis of the walker-jumper game, the decomposition Theorem 1.7 is easily proved.

Proof of Theorem 1.7. For the first part, we need to show

$$\frac{\lambda - \alpha_{\max}}{Z(\alpha_{\text{diff}})} - 1 \le 23e^{\alpha_{\text{diff}}}\sqrt{\frac{\delta_{\max}}{\delta}\alpha_{\max}^3}.$$

Leaving D unspecified for now, let us abbreviate $\lambda(WJ[D, \alpha_R, \alpha_L])$ by $\lambda(WJ)$. Then the left-hand side may be written and upper bounded as follows:

$$\frac{\lambda - \lambda(WJ)}{Z(\alpha_{\text{diff}})} + \frac{\lambda(WJ) - \alpha_{\max}}{Z(\alpha_{\text{diff}})} - 1 \le \left|\frac{\lambda - \lambda(WJ)}{Z(\alpha_{\text{diff}})}\right| + \left|\frac{\lambda(WJ) - \alpha_{\max}}{Z(\alpha_{\text{diff}})} - 1\right|.$$

We bound the two terms in the sum separately. Corollary 4.3 implies that, as long as D is chosen larger than $\alpha_{\max}^2$, the second term is at most $4\alpha_{\max}^2/D$. The first term

is bounded using Lemma 5.1, the trivial upper bound $\lambda(WJ) \le 2\alpha_{\max} + 1 \le 3\alpha_{\max}$ observed early in section 4, and the fact that $Z(x) \ge e^{-x}$ for all nonnegative x:

$$\begin{aligned}\left|\frac{\lambda - \lambda(WJ)}{Z(\alpha_{\text{diff}})}\right| &\le \frac{3\alpha_{\max}\eta}{Z(\alpha_{\text{diff}})} \\ &\le \frac{3\alpha_{\max}\cdot 6\cdot \delta_{\max} D}{Z(\alpha_{\text{diff}})\delta} \\ &\le \frac{18\alpha_{\max}\delta_{\max} D e^{\alpha_{\text{diff}}}}{\delta}.\end{aligned}$$

Now set D to be an integer satisfying $\sqrt{\delta\alpha_{\max}/\delta_{\max}} \le D \le \frac{19}{18}\sqrt{\delta\alpha_{\max}/\delta_{\max}}$. This is possible because the hypothesis of the theorem implies that $\sqrt{\delta\alpha_{\max}/\delta_{\max}} \ge 18$. It is easily verified that, under the hypothesis of Theorem 1.7, the resulting D satisfies both the hypothesis $D \ge \alpha_{\max}^2$ of Corollary 4.3 and the conditions in Lemma 5.1. Summing the upper bounds on the two terms using this value of D yields an upper bound of

$$19e^{\alpha_{\text{diff}}}\sqrt{\frac{\alpha_{\max}^3\delta_{\max}}{\delta}} + 4\sqrt{\frac{\alpha_{\max}^3\delta_{\max}}{\delta}} \le 23e^{\alpha_{\text{diff}}}\sqrt{\frac{\alpha_{\max}^3\delta_{\max}}{\delta}},$$

as required for the first part of the theorem.

For the second part of the theorem, either $\lambda_{\max} \ge \alpha_{\max} + Z(\alpha_{\text{diff}})(1-\zeta)$ or else we must prove

$$\frac{\alpha_{\max} - \lambda}{Z(\alpha_{\text{diff}})} + 1 \le \ 23e^{\alpha_{\text{diff}}}\sqrt{\frac{\delta_{\max}}{\delta}\alpha_{\max}^3}.$$

The proof is similar to that of the first part. □

Thus it remains to prove Lemma 5.1. The proof of this follows the outline in section 3 but needs a lot of technical work which is divided into three subsections. First, we state two technical results that bound the constants occurring in the definition of the competitive ratio; the proofs of these lemmas are deferred until the end of the paper. Then we prove the upper and lower bounds of the lemma.

5.1. Two technical lemmas. In the proof overview, we related a step to the left (resp., right) by the walker in the walker-jumper game, to the addition of an s-block for some appropriately chosen s by the pursuer in the PE game. In sketching how this works, we approximated the cost of such a left block to the evader by $s\lambda_R$. When we formalize this argument one of the things we will have to do is to bound the error in this approximation. For this, we will need two lemmas concerning the existence of "good" strategies for the pursuer and the evader in the PE game.

The competitive ratio for a two-person game was defined, in general, as an infimum over κ for which there is a κ-competitive algorithm. In general, this infimum need not be attained, i.e., if the competitive ratio is λ, there need not be a λ competitive strategy for MIN. However, as we will see in the next lemma, for the PE game there is always a randomized evader strategy that attains the competitive ratio. Furthermore, we can also upper bound the constant K that occurs in the definition of the competitive ratio.

LEMMA 5.2. *Let $\mathcal{M}$ be a metric space of diameter δ, and let λ denote the competitive ratio of its PE problem. Then there exists an algorithm $\tilde{A}$ such that for any probe sequence ρ*

$$C_{\tilde{A}}(\rho) \le \lambda C_{\text{OPT}}(\rho) + \delta\lambda.$$

Corollary 2.3 provides a criterion for lower bounding the competitive ratio of any game. The sequence of strategies $\tilde{r}_i$ in the hypothesis of the corollary can be viewed as a *witness* to the fact that the competitive ratio is at least κ. To prove tight lower bounds on the competitive ratio, we would like to be able to find such a witness sequence in the case $c = \lambda$.

LEMMA 5.3. *Let $\mathcal{M}$ be a metric space and λ the competitive ratio of its PE game. For any $s > 0$, there is a distribution $\tilde{\rho}_{\mathbf{s}}$ on s-blocks such that for any response algorithm A,*

$$C_A(\tilde{\rho}_{\mathbf{s}}) \geq \lambda(s - \delta) - \delta.$$

The proofs require a somewhat tedious technical formulation, after which the results follow from elementary analysis. So as not to distract from the flow of the main argument, we defer the proof to the last section of the paper.

5.2. The upper bound (proof of Lemma 5.1, part 1). We now proceed with the proof of Lemma 5.1. We assume for now that both the left and right space have at least two points. In the case that one or both of them is degenerate (has only one point), the proof is similar but requires some technical modifications which we indicate at the end of the subsection.

The upper bound is proved by associating for each jumper strategy $\tilde{J}$ for $WJ[D, \alpha_R, \alpha_L]$ an evader algorithm $\tilde{A}(\tilde{J})$ for $PE(\mathcal{M})$ that satisfies that if $\tilde{J}$ is κ-competitive, then $\tilde{A}(\tilde{J})$ is $\kappa(1+\eta)$-competitive. The evader algorithm $\tilde{A}(\tilde{J})$ must specify an on-line rule for responding to a probe sequence. The idea will be to associate the incoming probe sequence to a walker sequence, apply $\tilde{J}$ to that walker sequence, and then translate the jumper's moves into moves for the evader.

We first describe a rule for associating a probe sequence ρ for $\mathcal{M}$ to a walker sequence $\mathbf{w} = \mathbf{w}(\rho)$. View ρ as the interleaving of two probe sequences, one for $\mathcal{M}_R$ (the *right* subsequence) and one for $\mathcal{M}_L$ (the *left* subsequence). Define the parameter $s_2 = (\delta - 2\delta_{\max})/D$. (The reader should think of this as approximating δ/D.) Parse each of these subsequences separately into its s_2-block partition, as defined in section 2.3. Now build $\mathbf{w}$ in the following on-line manner. Each time a right (resp., left) block of ρ is completed, a right (resp., left) walker step is added to $\mathbf{w}$ unless that step will take the walker outside the bounds $[-D, D]$. Let $\rho^{\mathbf{i}}$ denote the prefix corresponding to the steps up to w_i. Let $k = |\mathbf{w}|$ and let μ be the portion of ρ coming after the last s_2-block of either subsequence; thus $\rho = \rho^{\mathbf{k}}\mu$.

Given the jumper strategy $\tilde{J}$, the algorithm $\tilde{A}(\tilde{J})$ responds to the probe sequence ρ as follows. As the sequence ρ is received, the evader constructs $\mathbf{w}(\rho)$ and simulates the response of $\tilde{J}$ to this sequence. The evader uses the position (left or right) of the simulated jumper to determine which space to be in; the response by $\tilde{J}$ to a step of $\mathbf{w}$ determines whether the evader continues in the same space or moves to the other. The times at which the evader switches spaces delimits a sequence of *left* and *right* intervals. During a right (resp., left) interval, the evader ignores probes on the other side and responds to the subsequence of right (resp., left) probes by applying the algorithm $\tilde{A}_R$ (resp., $\tilde{A}_L$), where $\tilde{A}_R$ (resp., $\tilde{A}_L$) is the evader strategy for $PE(\mathcal{M}_R)$ (resp., $PE(\mathcal{M}_L)$) that satisfies the conclusion of Lemma 5.2.

Now, supposing that $\tilde{J}$ is κ-competitive, we need to verify that $\tilde{A}(\tilde{J})$ is $\kappa(1+\eta)$-competitive. We will deduce this from the following lemma.

LEMMA 5.4. *Let $\tilde{J}$ be a jumper strategy for $WJ[D, \alpha_R, \alpha_L]$. Then for any probe sequence ρ for $PE(\mathcal{M})$,*

1. $C_{\mathrm{OPT}}(\rho) \geq (s_2 - \delta_{\max})C_{\mathrm{BASE}}(\mathbf{w}(\rho)) - \delta$,
2. $C_{\tilde{A}(\tilde{J})}(\rho) \leq (s_2 + 2\delta_{\max})C_{\tilde{J}}(\mathbf{w}(\rho)) + K$ *for some constant* K *independent of* ρ.

Using this lemma and the assumption that $\tilde{J}$ is κ-competitive, we obtain that for some constants H and H' independent of ρ

$$\begin{aligned} C_{\tilde{A}(\tilde{J})}(\rho) &\leq (s_2 + 2\delta_{\max})C_{\tilde{J}}(\mathbf{w}(\rho)) + K \\ &\leq (s_2 + 2\delta_{\max})\kappa C_{\mathrm{BASE}}(\mathbf{w}(\rho)) + H \\ &\leq \frac{s_2 + 2\delta_{\max}}{s_2 - \delta_{\max}}\kappa C_{\mathrm{OPT}}(\rho) + H' \\ &\leq \left(1 + \frac{3D\delta_{\max}}{\delta - (D+2)\delta_{\max}}\right)\kappa C_{\mathrm{OPT}}(\rho) + H' \\ &\leq \left(1 + \frac{6D\delta_{\max}}{\delta}\right)\kappa C_{\mathrm{OPT}}(\rho) + H', \end{aligned}$$

where the last inequality follows from $\delta \geq 2(D+2)\delta_{\max}$ which is an easy consequence of the hypotheses of the lemma. This implies that $\tilde{A}(\tilde{J})$ is $(1+\eta)\kappa$-competitive, as required to prove the upper bound.

So it remains to prove Lemma 5.4.

Proof of Lemma 5.4. Define $C_L(i)$ (resp., $C_R(i)$), for $1 \leq i \leq k$, to be the minimum cost of a response sequence to $\rho^{\mathbf{i}}$ whose last point is in $\mathcal{M}_L$ (resp., $\mathcal{M}_R$). Note that $C_{\mathrm{OPT}}(\rho) \geq C_{\mathrm{OPT}}(\rho^{\mathbf{k}}) = \min\{C_R(k), C_L(k)\}$. Since $|C_L(i) - C_R(i)| \leq \delta$, we have $C_{\mathrm{OPT}}(\rho) \geq C_R(k) - \delta$.

LEMMA 5.5. $C_R(i) \geq (s_2 - \delta_{\max})(i + w_i)/2$ *and* $C_L(i) \geq (s_2 - \delta_{\max})(i - w_i)/2$.

Since $C_{\mathrm{BASE}}(\mathbf{w}) = (k + w_k)/2$, the first part of Lemma 5.4 follows.

Proof. We prove this by induction on i; the basis $i = 0$ is trivial. Suppose $i > 0$. Assume that $(\Delta w)_i = +1$; the case $(\Delta w)_i = -1$ is proved analogously. Thus $\rho^{\mathbf{i}}$ marks the end of a right block.

The induction step for $C_L(i)$ follows from $C_L(i) \geq C_L(i-1)$. For $C_R(i)$, let $j < i$ be the least index such that $(\Delta w)_h = -1$ for $j < h < i$. Thus either $j = 0$ and w_i is the first step to the right, or w_j is the last step to the right prior to w_i. It is easy to see that this implies that $i + w_i = j + w_j + 2$, and hence $j \geq i + w_i - D - 2$.

We need to show that any response sequence for $\rho^{\mathbf{i}}$ that ends on the right costs at least $(s_2 - \delta_{\max})(w_i + i)/2$. Consider such a response sequence, written as $\sigma\tau$, where σ is the portion that is a response sequence to $\rho^{\mathbf{j}}$.

Note that $d(\sigma)$ must be at least the minimum of $C_L(j)$ and $C_R(j)$, which, by induction and the above expression for j, is at least $(s_2 - \delta_{\max})(j - D)/2 \geq (s_2 - \delta_{\max})(i + w_i - 2D - 2)/2$. Any move between the two spaces costs at least $\delta - 2\delta_{\max} = Ds_2$, which can be shown to be larger than $(s_2 - \delta_{\max})(D+1)$ using the hypotheses $D \geq \sqrt{\delta/\delta_{\max}}$ of the theorem. Thus it follows that if there is any move between spaces subsequent to σ, then the total cost of the response sequence is at least $(s_2 - \delta_{\max})(w_i + i)/2$, as required.

So assume that after σ there is no move between spaces. Then σ ends on the right and all steps of τ are on the right. Since the portion of ρ between $\rho^{\mathbf{j}}$ and $\rho^{\mathbf{i}}$ contains a right s_2-block, it follows that τ has optimal cost at least $s_2 - \delta_{\max}$. Thus

$$\begin{aligned} d(\sigma\tau) &\geq (s_2 - \delta_{\max})(w_j + j)/2 + s_2 - \delta_{\max} \\ &= (s_2 - \delta_{\max})(w_j + j + 2)/2 \\ &= (s_2 - \delta_{\max})(w_i + i)/2, \end{aligned}$$

as required. □

We now turn to the proof of the second part of Lemma 5.4. For this we want to upper bound the expected cost incurred by $\tilde{A}(\tilde{J})$ on ρ, in terms of the expected cost of $\tilde{J}$ on $\mathbf{w}(\rho)$. Consider the cost incurred by the evader during each interval that begins after a move into one space and ends with the move out of that space, or, in the case of the last such interval, with the end of the sequence ρ. We will compare the evader's cost to the cost of the jumper during the corresponding interval. Without loss of generality, assume that the evader is on the right during this interval and thus the jumper is at $+D$. This implies that the simulated walker cannot be at $+D$ at any step (except possibly the last one) during this interval.

First, consider the case that the interval is not the last such interval. The evader responds only to the probes that occur on the right during that interval. In the definition of the algorithm, the subsequence of right probes was partitioned into right s_2-blocks. Let m be the number of s_2-blocks that end during this interval. By the definition of $\mathbf{w}$ and the fact that the walker is not at $+D$ at any time during the interval, it follows that the walker took m right steps during the interval. Thus the cost to the simulated jumper is $m\alpha_R$ for those steps, plus D for the final jump.

Now consider the cost to the evader corresponding to the interval. Since the evader is on the right, he responds only to the right probes that occur during the interval. The subsequence of right probes has optimal cost (with respect to the right space) at most $m(s_2 + \delta_R)$ since it is the concatenation of m sequences, each with optimal cost at most s_2. Since the evader uses algorithm $\tilde{A}_R$ to respond, the expected cost of responding is at most $\alpha_R m(s_2 + \delta_R) + \lambda_R \delta_R \le \alpha_R m(s_2 + 2\delta_R)$. Adding the cost of the final move to the left space, which is at most δ, we obtain an upper bound on the evader's cost for the interval of

$$\begin{aligned}\alpha_R m(s_2 + 2\delta_R) + \delta &= \alpha_R m(s_2 + 2\delta_R) + Ds_2 + 2\delta_{\max} \\ &\le \alpha_R m(s_2 + 2\delta_R) + D(s_2 + 2\delta_{\max}) \\ &\le (\alpha_R m + D)(s_2 + 2\delta_{\max}).\end{aligned}$$

Thus the evader's cost on every interval except the last is at most $(s_2 + 2\delta_{\max})$ times the simulated jumper's cost on the interval.

For the last interval, the cost to the jumper is $\alpha_R m$. To bound the expected cost to the evader we must include the portion μ of ρ which occurs after the last step by the simulated walker. This could increase the optimal cost (with respect to the right space) of the subsequence of right probes in this interval to $(m+1)(s_2+\delta_R)$. Thus the expected cost to the evader within the interval is bounded above by $\alpha_R(m+1)(s_2 + \delta_R) + \alpha_R\delta_R$, which is in turn bounded above $(s_2 + 2\delta_R)$ times the simulated jumper's cost of $m\alpha_R$ plus a constant that does not depend on ρ. □

We now indicate how to modify the proof in the case that one or both of the spaces has exactly one point. First, we have to modify the definition of the walker sequence $\mathbf{w}(\rho)$ associated to a probe sequence. As before, we view ρ as the interleaving of a right and a left subsequence, and we parse each of these sequences into their s_2-block partitions. Recall that for one-point spaces, we defined an s_2-block partition to be the partition into singleton blocks. As before, at the completion of a left or right block, the next step of the walker is generated. For a block corresponding to a nondegenerate subspace, the walker step is generated as before. A block corresponding to a degenerate subsequence will correspond to not one but a sequence of walker steps that take the walker all the way to the corresponding endpoint ($-D$ for a left block and D for a right block). If this is the ith completed block, then we abuse notation

by setting $w_i = D$ for a right block and $w_i = -D$ for a left block; thus we compress all of the walker steps corresponding to a degenerate block into one step.

Lemma 5.4 can be extended to hold in this case. The definitions of $C_L(i)$ and $C_R(i)$ need to be modified in Lemma 5.7: if the left (resp., right) space is degenerate and $\rho^{(\mathbf{i})}$ ends with a left (resp., right) block, then the definition of $C_L(i)$ (resp., $C_R(i)$) does not make sense and we modify it by defining $C_L(i) = C_R(i) + \delta$ (resp., $C_R(i) = C_L(i) + \delta$). The proof of Lemma 5.7 is then routine.

In the proof of the second part of Lemma 5.4 we analyze intervals defined by moves of the evader from one space to the other. The case of an interval in which the evader occupies a degenerate space is different than those analyzed but is actually easier since during the interval there are no requests inside the degenerate space and the cost to the evader is the cost of the move to the other space at the end of the interval which is bounded above by δ. The cost to the jumper during the same interval is just D.

5.3. The lower bound (proof of Lemma 5.1, part 2). As with the upper bound, we prove the lower bound in the case that $\mathcal{M}_L$ and $\mathcal{M}_R$ each have size at least two. Afterwards, we indicate how to modify the proof to handle the degenerate case that one or both spaces has only one point.

The strategy of the lower bound is the "mirror image" of the upper bound proof. That is, we will define a function which maps each evader algorithm $\tilde{A}$ for $PE(\mathcal{M})$ to a jumper algorithm $\tilde{J}(\tilde{A})$ for $WJ[D, \alpha_R, \alpha_L]$ and has the following property: if $\tilde{A}$ is a κ-competitive algorithm for the $PE(\mathcal{M})$, then $\tilde{J}(\tilde{A})$ is a $\frac{\kappa}{1-\eta}$-competitive algorithm for $WJ[D, \alpha_R, \alpha_L]$, where $\eta \le 6\frac{\delta_{\max} D}{\delta}$. This immediately implies the lower bound of the lemma.

There are two main steps. Define the parameter $s_1 = \delta/D$.

1. For each walker strategy $\mathbf{w}$ for the game $WJ[D, \alpha_R, \alpha_L]$, we define a distribution $\tilde{\mu}(\mathbf{w})$ on probe sequences for the pursuer in the PE game on $\mathcal{M}$ such that

$$C_{\mathrm{BASE}}(\mathbf{w})(s_1 + \delta_{\max}) \ge C_{\mathrm{OPT}}(\tilde{\mu}(\mathbf{w})). \tag{5.1}$$

2. For each (randomized) evader algorithm $\tilde{A}$ we define a jumper algorithm $\tilde{J}(\tilde{A})$ for $WJ[D, \alpha_R, \alpha_L]$ and show that for all evader algorithms $\tilde{A}$ and walker strategies $\mathbf{w}$

$$C_{\tilde{A}}(\tilde{\mu}(\mathbf{w})) \ge C_{\tilde{J}(\tilde{A})}(\mathbf{w})(s_1 - 2\delta_{\max}) \tag{5.2}$$

(where the first cost function is with respect to $PE(\mathcal{M})$ and the second is with respect to $WJ[D, \alpha_R, \alpha_L]$).

It follows immediately from these two steps that if $\tilde{A}$ is κ-competitive, then there is a real number K such that for any walker sequence $\mathbf{w}$

$$(1 - 2\delta_{\max} D/\delta)(1 - \delta_{\max} D/\delta) C_{\tilde{J}(\tilde{A})} \le \kappa C_{\mathrm{BASE}}(\mathbf{w}) + K,$$

and thus $\tilde{J}(\tilde{A})$ is $\frac{\kappa}{(1-3\delta_{\max} D/\delta)}$-competitive, which will complete the proof.

For this proof, we define a *left block* (resp., *right block*) to be an s_1-block of the space $\mathcal{M}_L$ (resp., $\mathcal{M}_R$). For a walker sequence $\mathbf{w} = (w_0 = 0, w_1, \ldots, w_k)$ in the game $WJ[D, \alpha_R, \alpha_L]$ we say that probe sequence ν for $\mathcal{M}$ is *compatible* with $\mathbf{w}$ if ν is the concatenation of k probe sequences $\nu_{\mathbf{1}}\nu_{\mathbf{2}} \ldots \nu_{\mathbf{k}}$ satisfying the following.

1. For each i such that $|w_i| \neq D$, $\nu_{\mathbf{i}}$ is a single right block if $(\Delta_w)_i = 1$ and is a single left block if $(\Delta_w)_i = -1$.
2. Let N be a large integer parameter to be specified later. If $|w_i| = D$, then $\nu_{\mathbf{i}}$ is the concatenation of N right blocks if $(\Delta_w)_i = 1$ and is the concatenation of N left blocks if $(\Delta_w)_i = -1$.

The sequences $\nu_{\mathbf{i}}$ are referred to as *segments* and are designated as left or right segments depending on whether they are from $\mathcal{M}_R$ or $\mathcal{M}_L$. A segment consisting of a single block is called a *small* segment, and one consisting of N blocks is called a *large* segment.

The distribution $\tilde{\mu}(\mathbf{w})$ will be a distribution over probe sequences compatible with $\mathbf{w}$. To define this distribution, we first need the following special case (actually, a slight weakening) of Lemma 5.3.

COROLLARY 5.6. *There exists a distribution $\tilde{\rho}_L$ on the set of left s_1-blocks and a distribution $\tilde{\rho}_R$ on the set of right s_1-blocks such that the following hold.*

1. *For any response algorithm A for $\mathcal{M}_L$, $C_A(\tilde{\rho}_L) \geq \alpha_L(s_1 - 2\delta_L)$.*
2. *For any response algorithm A for $\mathcal{M}_R$, $C_A(\tilde{\rho}_R) \geq \alpha_R(s_1 - 2\delta_R)$.*

The distribution $\tilde{\mu}(\mathbf{w})$ is now defined to be the distribution over probe sequences compatible with $\mathbf{w}$ in which each left block is chosen independently from the distribution $\tilde{\rho}_L$ and each right block is chosen independently from the distribution $\tilde{\rho}_R$.

Next, we relate the base-cost of $\mathbf{w}$ to the expected optimal cost $\tilde{\mu}(\mathbf{w})$. The base-cost of $\mathbf{w}$ equals the number of right steps, which is equal to $(k+w_k)/2$, where $k = |\mathbf{w}|$. We will show that for any probe sequence ν compatible with $\mathbf{w}$, its optimal cost is at most $(s_1 + \delta_{max})(k + w_k)/2$. Since $\tilde{\mu}(\mathbf{w})$ is a distribution over such sequences, this will imply inequality (5.1).

So fix ν compatible with $\mathbf{w}$. Denote by $C_L(i)$ (resp., $C_R(i)$) the minimum cost of a response sequence for the first i segments of ν whose last point is in $\mathcal{M}_L$ (resp., $\mathcal{M}_R$). Note that $|C_L(i) - C_R(i)| \leq \delta$. Then $C_{\mathrm{OPT}}(\nu)$ is equal to the minimum of $C_L(k)$ and $C_R(k)$, and thus the desired upper bound on $C_{\mathrm{OPT}}(\nu)$ is an immediate consequence of the following lemma.

LEMMA 5.7. *$C_R(i) \leq (s_1 + \delta_{\max})(i + w_i)/2$ and $C_L(i) \leq (s_1 + \delta_{\max})(i - w_i)/2$.*

Proof. We prove this by induction on i; the basis case $i = 0$ is trivial. Now suppose that $i > 0$ and that the result holds for $i - 1$.

We assume $(\Delta w)_i = +1$; the case $(\Delta w)_i = -1$ is proved analogously. Thus ν_i consists of one or N s_1-blocks from $\mathcal{M}_R$.

To prove the induction step for $C_L(i)$ it suffices to observe that $C_L(i) = C_L(i-1)$, which is obvious since the definition of $C_L(i-1)$ implies that there is a response sequence for $\nu_{\mathbf{1}} \ldots \nu_{\mathbf{i-1}}$ that ends at some point $y \in \mathcal{M}_L$ that costs $C_L(i-1)$. This sequence can be extended by remaining at y through $\nu_{\mathbf{i}}$ and the cost does not increase.

Next we prove the induction step for $C_R(i)$. In the case that segment ν_i consists of a single right block, then by the definition of $C_R(i-1)$ there is a response sequence σ for $\nu_{\mathbf{1}} \ldots \nu_{\mathbf{i-1}}$ ending at some point $y \in \mathcal{M}_R$ that costs at most $C_R(i-1)$. Since ν_i is an s_1-block for $\mathcal{M}_R$, there is a response sequence τ for $\nu_{\mathbf{i}}$ that costs at most s_1 and ends in $\mathcal{M}_R$. Then the sequence $\sigma\tau$ costs at most $C_R(i-1)+s_1+d(y,\tau_1) \leq C_R(i-1)+s_1+\delta_{\max}$ which is bounded by $(s_1 + \delta_{\max})(i + w_i)/2$ by the induction hypothesis.

In the case that the segment ν_i consists of N right blocks, we must have $w_i = D$. Then we use the fact that $C_R(i) \leq C_L(i)+\delta$ to obtain $C_R(i) \leq (s_1+\delta_{\max})(i-D)/2+ s_1 D \leq (s_1 + \delta_{\max})(i + D)/2$ as required. ∎

Now we turn to the second and final step: the definition of $\tilde{J}(\tilde{A})$ and the verification of inequality (5.2). Given an evader strategy $\tilde{A}$, the jumper strategy $\tilde{J}(\tilde{A})$ will

be defined as follows. On being presented walker sequence $\mathbf{w}$, J generates a probe sequence according to the distribution $\tilde{\mu}(\mathbf{w})$. Note that this generation can be done on-line with segment i being generated given w_i. At the same time, he simulates the algorithm A on the sequence $\tilde{\mu}(\mathbf{w})$. At step i, the jumper responds to w_i as follows.

1. If $|w_i| = D$, then $j_i = -w_i$ (which is forced by the rules of the game).
2. If $|w_i| < D$ and $\mathbf{w}_i$ is in the direction opposite to j_{i-1} (i.e., they have opposite signs), then the jumper does not move, i.e., $j_i = j_{i-1}$.
3. If $|w_i| < D$ and $\mathbf{w}_i$ is in the direction towards j_{i-1}, then $j_i = D$ if $\tilde{A}$ is in $\mathcal{M}_R$ at the end of segment i and $j_i = -D$ otherwise.

It remains to verify inequality (5.2) and it suffices to verify this inequality for deterministic algorithms A; the result for randomized algorithms will follow by taking expectation with respect to the distribution over algorithms. So fix a deterministic algorithm A.

We will say that the simulated evader and the jumper are *synchronized at step i* if after the ith step of the jumper the jumper is at $+D$ and the evader is either in the left space or the jumper is at $-D$ and the evader is in the right space.

For technical reasons, it will be useful to modify the cost to the jumper by adding D to his cost if at the last step he is not synchronized with the evader. Obviously, this modified cost is an upper bound on the true cost, so it suffices to work with this modified cost. What we will show is that the expected change in the cost incurred by the evader during segment i is at least $(s_1 - 2\delta_{\max})$ times the expected change in the modified cost incurred by the jumper at step i. We assume that the jumper is on the right, $j_{i-1} = D$; the other case is handled similarly.

If step i of the walker is to the left, then the change in the jumper's actual cost will be 0. His modified cost will go up by D if and only if the evader was in $\mathcal{M}_R$ and moved to $\mathcal{M}_L$ during segment i. In this case, the evader's cost increased by at least $\delta - 2\delta_{\max}$ which is at least $(s_1 - 2\delta_{\max})D$.

Now consider the case that the ith step of the walker is to the right. If the jumper and evader were not synchronized at step $i-1$, then at the end of step i, there are three possibilities: they both end on the left, they both end on the right, or they switch places. (This can happen only if $w_i = D$ so that the jumper is forced to jump.) In the first two cases, the expected change in the modified cost of the jumper is less than or equal to 0, while the evader always incurs a nonnegative cost, and the desired inequality is trivial. In the third case, the jumper's modified cost increases by D, while the evader's cost increases by at least $\delta - 2\delta_{\max}$ which is at least $(s_1 - 2\delta_{\max})D$.

So we may assume that the jumper and evader are both on the right after step $i-1$ and the next step is to the right. As the evader is following a lazy algorithm, we may assume that if the evader moves left during the block, he does not move again during the block.

We consider separately the cases that $w_i < D$ and $w_i = D$. In the case that $w_i < D$, then by the definition of the jumper's strategy, they will still be synchronized at step i. So either they both stayed on the right, or they both moved to the left. If they both end on the left, then the jumper's modified cost increases by D and the evader's cost increases by at least $\delta - 2\delta_{\max}$ which we have already noted is good for us. If they both stay on the right, then the jumper incurs a cost of α_R and we'd like to say that the evader incurs an expected cost of at least $(s_1 - 2\delta_R)\alpha_R$. This would seem to be true: the pursuer chooses her s_1-block from the distribution $\tilde{\rho}_{\mathbf{R}}$, and we know from Corollary 5.6 that any algorithm B incurs expected cost at least $(s_1 - 2\delta_R)\alpha_R$ against this distribution.

However, there is a subtle flaw in this reasoning. The algorithm A does not have to decide whether to move to the left at the beginning of the s_1-block; it can start on the right and at some point decide to move left. For example, suppose A stays on the right as long as the cost he has incurred during that block is less than some value V. The conditional expectation of the cost incurred given that the algorithm finishes the block on the right is at most V. Since V can be chosen less than $(s_1 - 2\delta_R)\alpha_R$, this demonstrates that the above argument is fallacious.

To argue correctly, we must consider that, conditioned on the probe sequence up to the beginning of the current block, there is a probability p that A stays on the right for the entire block. Thus the expected increase in the jumper's modified cost is $p\alpha_R + (1-p)D$. We need to show that the expected increase in the evader's cost is at least $(s_1 - 2\delta_R)$ times this. The expected increase in the evader's cost is $(1-p)(\delta - 2\delta_{\max})$ plus the expected cost incurred in responding to probes while on the right. Now the key point is that when computing this expected cost, not only must we consider the cost incurred when the evader finishes on the right, but also, in the case that the evader finishes on the left, we must consider the cost incurred by the evader before moving to the left.

So let us consider the behavior of the algorithm A from the beginning of this s_1-block until the point that it jumps to the left. We can think of this algorithm as one for the game $PE(\mathcal{M}_R)$ which has the additional option of *stopping* the game in the middle. Let us call such an algorithm a *stopping algorithm*.

LEMMA 5.8. *Let B be a stopping algorithm for $PE(\mathcal{M}_R)$. Let p be the probability that B does not stop on input distribution $\tilde{\rho}_R$. Let q denote the expected cost incurred by B before it stops. Then $q \geq p\alpha_R(s_1 - 2\delta_R) - (1-p)\delta_R(3\lambda_R + 1)$.*

Proof. Define the (nonstopping) algorithm $\tilde{B}'$ for $PE(\mathcal{M}_R)$ as follows: respond using B until B stops. Then switch to using the algorithm $\tilde{A}_R$, the algorithm that satisfies the conclusion of Lemma 5.2.

Now we upper bound $C_{\tilde{B}'}(\tilde{\rho}_{\mathbf{R}})$. This cost is at most q plus the cost incurred after switching to $\tilde{A}_R$. The probability of ever switching to $\tilde{A}_R$ is $1-p$, and conditioned on switching, $\tilde{A}_R$ incurs a cost of at most δ_R for its first move and an expected cost of at most $\lambda_R(s_1 + \delta_R)$ for the rest of its moves since it is responding to an s_1 block (or a subsequence of it). Thus $C_{\tilde{B}'}(\tilde{\rho}_R) \leq q + (1-p)(\lambda_R(s_1 + \delta_R) + \delta_R)$. On the other hand, by Corollary 5.6, $C_{\tilde{B}'}(\tilde{\rho}_R) \geq \alpha_R(s_1 - 2\delta_R)$. We conclude, therefore, that $q \geq p\alpha_R(s_1 - 2\delta_R) - (1-p)(3\lambda_R + 1)(\delta_R)$. ∎

With Lemma 5.8 in hand, we now can lower bound the expected cost to the evader in responding to the s_1-block by

$$\begin{aligned} &p\alpha_R(s_1 - 2\delta_R) - (1-p)\delta_R(3\lambda_R + 1) + (1-p)(\delta - 2\delta_{\max}) \\ &\quad \geq p\alpha_R(s_1 - 2\delta_R) + (1-p)(\delta - (3\lambda_R + 3)\delta_{\max}) \\ &\quad \geq (s_1 - 2\delta_{\max})(p\alpha_R) + (1-p)D(s_1 - 2\delta_{\max}), \end{aligned}$$

where the last inequality is obtained by applying the hypotheses about D and s_1. The final term is $(s_1 - 2\delta_{\max})$ times the change in the jumper's modified cost, as needed.

Finally, we consider the case that both the jumper and evader begin on the right, $w_{i-1} = D - 1$, and the step by the walker is to the right. As before, since the evader is following a lazy algorithm, either he stays on the right or he moves to the left at some time and stays there. Let p be the probability that the evader stays on the right. The idea is that since the segment corresponding to the last step of the walker consists of N right blocks, where N is a large integer, we want to conclude that either p is extremely small, or the evader incurs a very large cost. For $1 \leq j \leq N$, define p_j to

be the probability that the evader is still on the right after j of the s_1-blocks; thus for all j, $p_j \geq p_N = p$. Note that the conditional probability that he is on the right after block j given that he is on the right after block $j-1$ is p_{j+1}/p_j. Then by Lemma 5.8, the expected cost incurred due to responding to requests on the right during block j is at least $p_j(\alpha_R(s_1 - 2\delta_R)) - (p_{j-1} - p_j)\delta_R(3\lambda_R + 1)$.

Summing this over j, we find that the expected cost incurred due to responses on the right is at least $pN\alpha_R(s_1 - 2\delta_R) - (1-p)\delta_R(3\lambda_R + 1)$. Adding in the expected cost of the final move to the left, and simplifying we get a lower bound on the evader's cost of the form $(1-p)D(s_1 - 2\delta_{\max}) + pNT$, where T is a positive real number. We need this to be at least $(s_1 - 2\delta_{\max})$ times the expected change in the jumper's modified cost, which is $D + pD$ since he must move to the left. For this we need pNT to be at least $2pD(s_1 - 2\delta_{\max})$ and this will hold as long as N was chosen large enough. □

Finally, let us indicate how the above proof changes if one of the spaces, say $\mathcal{M}_R$, has exactly one point p. The only change in the definitions of $\mathbf{w}(\rho)$ and $\tilde{J}(\tilde{A})$ comes from a change in the definition of compatibility. The modification is that for a step to the right $(\Delta w)_i = 1$, the segment $\nu_{\mathbf{i}}$ is defined to be empty if $w_i < D$ and is equal to the single point p if $w_i = D$. The analysis of this strategy involves similar considerations to the given proof and is left to the reader.

6. Applications of the decomposition theorem. In this section, we use Theorem 1.7 to prove Theorems 1.3 and 1.5. (Theorem 1.4 was already proved in the introduction.)

Recall that a subspace $\mathcal{N}$ of space $\mathcal{M}$ is ϵ-small relative to $\mathcal{M}$ if $\delta(\mathcal{N}) \leq \epsilon\delta(\mathcal{M})$. We also say that $\mathcal{M}$ is *ϵ-uniform* if the distance between any two points in $\mathcal{M}$ is at least $\epsilon\delta(\mathcal{M})$. The following easy consequence of Theorem 1.1 is left to the reader (see also [KRR]).

LEMMA 6.1. *The competitive ratio of an* ϵ-uniform *space on n points is between $\epsilon \ln n$ and $(2/\epsilon)\ln n$.*

It will be convenient to state three special cases of Theorem 1.7.

COROLLARY 6.2. *Let $\mathcal{M}$ be a metric space of at least three points and let $\mathcal{M}_R, \mathcal{M}_L$ be a partition. Let α_R and α_L be nonnegative numbers with $\alpha_{\max}$ their maximum and α_{diff} their absolute difference. Suppose that $\alpha_{\max} \geq 1$ and that both spaces are $(e^{2\alpha_{\text{diff}}}\alpha_{\max}^3 2200)^{-1}$-small in $\mathcal{M}$.*

1. *If $\alpha_L \geq \lambda_L$ and $\alpha_R \geq \lambda_R$, then $\lambda \leq \alpha_{\max} + 3Z(\alpha_{\text{diff}})/2$.*
2. *If $\alpha_L \leq \lambda_L$ and $\alpha_R \leq \lambda_R$, then $\lambda \geq \alpha_{\max} + Z(\alpha_{\text{diff}})/2$.*

Proof. Apply the bounds of Theorem 1.7, noting that the hypothesis of the Corollary guarantees $\zeta \leq 1/2$. □

The special case where $\alpha_R = \alpha_L$ is worth noting.

COROLLARY 6.3. *Let $\mathcal{M}$ be a metric space and let $\mathcal{M}_R, \mathcal{M}_L$ be a partition into two subspaces of size at least 2. Let $\beta \geq 1$ be a lower bound on both λ_R and λ_L and suppose that both spaces $\mathcal{M}_R$ and $\mathcal{M}_L$ are $(2200\beta^3)^{-1}$-small. Then $\lambda \geq \beta + 1/2$.*

We also have the following corollary.

COROLLARY 6.4. *Let $\mathcal{M}$ be a metric space of at least three points and let $\mathcal{N}$ be a subspace. Let $\beta \geq 1$ be a lower bound on $\lambda(\mathcal{N})$ and suppose that $\mathcal{N}$ is $(27000e^{3\beta})^{-1}$-small. Then $\lambda(\mathcal{M}) \geq \beta + e^{-2\beta}$.*

Proof. Let x, y be points of $\mathcal{M}$ of distance δ. Then by the triangle inequality, one of the spaces obtained by adding exactly one of x and y to $\mathcal{N}$ has diameter at least $\delta/2$. Assume x is the point and let $\mathcal{K}$ be the union of $\mathcal{K}_R = \mathcal{N}$ and $\mathcal{K}_L = \{x\}$. The hypothesis of the corollary guarantees that the ratio of $\delta(\mathcal{N})$ to $\delta(\mathcal{K})$ is at most $\frac{2e^{-3\beta}}{27000}$ which is less than or equal to $\frac{\beta^{-3}e^{-2\beta}}{2200}$ for any $\beta \geq 1$. Thus we may apply Theorem 1.7

to the space $\mathcal{K}$ with $\alpha_R = \beta$ and $\alpha_L = 0$. We obtain a lower bound on the competitive ratio of $\mathcal{K}$ of $\beta + Z(\beta)(1-\zeta)$. Finally, note that $Z(\beta) \geq 2e^{-2\beta}$ for $\beta \geq 1$ and that the hypothesis of the corollary guarantees that $\zeta \leq 1/2$. □

6.1. Tight bounds for highly unbalanced spaces.

Proof of Theorem 1.5. We want to show that for some polynomial $p(n)$, any $p(n)$-unbalanced metric space has competitive ratio between $\ln n$ and $3 \ln n$. We prove only the upper bound here; the lower bound proof is very similar. For $n = 1, 2$ the result is trivial. Let $n > 2$ and let $\mathcal{M}$ be a $p(n)$-unbalanced metric space on n points. Let x and y be two points of distance δ, the diameter of the space. Then by the imbalance property, every other point z is close to exactly one of the points x or y, i.e., its distance to one of them is at most $\frac{\delta}{p(n)}$. This yields a $\frac{2}{p(n)}$-bipartition $(\mathcal{M}_L, \mathcal{M}_R)$. Let $n_L = |\mathcal{M}_L|$ and $n_R = |\mathcal{M}_R|$ and assume $n_L \geq n_R$. By induction, $\lambda(\mathcal{M}_L) \leq 3\ln(n_L)$ and $\lambda(\mathcal{M}_R) \leq 3\ln(n_R)$. Calling these upper bounds α_L and α_R, we have $\alpha_{\max} = 3\ln(n_L)$ and $\alpha_{\text{diff}} = 3\ln(n_L/n_R)$. By choosing $p(n)$ to be a sufficiently large polynomial, the conditions of Corollary 6.2 apply, and $\lambda \leq 3\ln(n_L) + (3/2)Z(3\ln(n_L/n_R))$. Thus it suffices to show

$$3\ln n \geq 3\ln(n_L) + \frac{3}{2} Z(3ln(n_L/n_R)).$$

For $n_L = n_R$, this reduces to $\ln 2 \geq 1/2$. For $n_L > n_R$, let $\rho = n_L/n_R - 1$, so that $\rho > 0$. Rewriting the desired inequality in terms of ρ, we need

$$\ln\left(1 + \frac{1}{1+\rho}\right) \geq \frac{3\ln(1+\rho)}{6\rho + 6\rho^2 + 2\rho^3}.$$

Using $x \geq \ln(1+x) \geq x - x^2/2$ to lower bound the left-hand side and upper bound the right-hand side, it is enough to show

$$\frac{1+2\rho}{2(1+\rho)^2} \geq \frac{3}{6 + 6\rho + 2\rho^2},$$

which is easily checked by cross-multiplying. □

6.2. A lower bound for all metric spaces. In order to prove Theorem 1.3 we will need a structure lemma for finite metric spaces, which says roughly that every finite metric space contains at least one of the following: (1) a small set of points whose removal reduces the diameter by a large fraction, (2) a roughly uniform subspace of large size, or (3) a bipartite subspace in which both parts are large and have small diameter relative to their union.

LEMMA 6.5. *Let $k \geq 0$ be an integer and $s \geq 1$. Every finite metric space $\mathcal{M}$ has a subspace satisfying at least one of the following conditions:*

1. *a 2^{1-s}-small subspace of size at least $(1 - \frac{1}{s})|\mathcal{M}|$,*
2. *a $\frac{1}{4}$-uniform subspace of size at least s,*
3. *a 2^{1-k}-bipartite subspace, each of whose parts has size at least $\frac{|\mathcal{M}|}{2s^{k+2}}$.*

Proof. The first step in the proof is given by the following proposition.

PROPOSITION 6.1. *Let $k \geq 0$ be an integer and $s \geq 1$ be real. Any finite metric space $\mathcal{M}$ that has no $\frac{1}{4}$-uniform subspace of size at least s has a 2^{-k}-small subspace of size at least $\frac{|\mathcal{M}|}{s^k}$.*

We proceed by induction on k; the basis $k = 0$ is trivial. Suppose $k > 0$ and that $\mathcal{M}$ has no $\frac{1}{4}$-uniform subspace of size at least s. By the induction hypothesis there

is a subspace $\mathcal{N}$ of size at least $\frac{|\mathcal{M}|}{s^{k-1}}$ and diameter at most $\delta 2^{1-k}$. Let $x_1, x_2, \ldots, x_t$ be a maximal sequence of points in $\mathcal{N}$ such that $d(x_i, x_j) \geq \delta(\mathcal{N})/4$ for $i \neq j$. For each $i \in \{1, \ldots, t\}$ let $X_i = \{y \in \mathcal{N} : d(y, x_i) < \delta(\mathcal{N})/4\}$. By the maximality of t, $\cup_i X_i = \mathcal{N}$. The largest X_i has size at least $|\mathcal{N}|/t \geq |\mathcal{M}|/s^k$ and has diameter at most $\delta(\mathcal{N})/2 \leq \delta(\mathcal{M})/2^k$. □

PROPOSITION 6.2. *Let $k \geq 0$ be an integer, $s \geq 1$, and $\gamma \in (0, 1)$. Any finite metric space $\mathcal{M}$ has at least one of the following:*

1. *a 1/2-small subspace of size at least $(1-\gamma)|\mathcal{M}|$,*
2. *a 1/4-uniform subspace of size at least s,*
3. *a 2^{1-k}-bipartite subspace, each of whose parts has size at least $\frac{\gamma}{s^k}|\mathcal{M}|$.*

Proof. Assume that $\mathcal{M}$ has no 1/4-uniform subspace of size at least s. Define sequences $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_u$ and $\mathcal{N}_1, \ldots, \mathcal{N}_u$ as follows. $\mathcal{M}_0 = \mathcal{M}$. Having constructed $\mathcal{M}_i$, if $|\mathcal{M}_i| < \gamma|\mathcal{M}|$, then stop and set $u = i$. Otherwise, the previous proposition implies that $\mathcal{M}_i$ has a 2^{-k}-small subspace of size at least $\frac{|\mathcal{M}_i|}{s^k} \geq \frac{\gamma}{s^k}|\mathcal{M}|$. Let $\mathcal{N}_{i+1}$ be such a subspace and let $\mathcal{M}_{i+1} = \mathcal{M}_i - \mathcal{N}_{i+1}$.

The union $\mathcal{N}$ of the $\mathcal{N}_i$ has size at least $(1-\gamma)|\mathcal{M}|$. If it is 1/2-small, then we have a space satisfying the first condition. Otherwise, there exist $x, y \in \mathcal{N}$ with $d(x, y) \geq \delta/2$. If $\mathcal{N}_i$ and $\mathcal{N}_j$ are the parts containing x and y, respectively, then their union is a space satisfying the third condition. □

To complete the proof of Lemma 6.5, fix k, s as hypothesized. Let $\mathcal{M}$ be given and assume that $\mathcal{M}$ has no $\frac{1}{4}$-uniform subspace of size at least s and no 2^{1-k}-bipartite subspace in which each part has size at least $\frac{1}{2s^{k+2}}|\mathcal{M}|$. We show that $\mathcal{M}$ has a 2^{1-s}-small subset of size at least $(1-\frac{1}{s})|\mathcal{M}|$. This is trivial if $s = 1$, so assume $s > 1$ and set $\gamma = 1/s^2$. We define a sequence of metric spaces $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_{\lfloor s \rfloor}$, where $\mathcal{M}_i$ has size at least $(1 - i\gamma)|\mathcal{M}|$ and has diameter at most $\delta 2^{-i}$. Then $\mathcal{M}_{\lfloor s \rfloor}$ has the desired properties.

To define the sequence, let $\mathcal{M}_0 = \mathcal{M}$. For $0 < i < \lfloor s \rfloor$, having defined $\mathcal{M}_i$, apply Proposition 6.2 to it. Our assumption about $\mathcal{M}$ implies that neither conclusion (2) nor conclusion (3) of Proposition 6.2 hold for $\mathcal{M}_i$. (For this we need to observe that since $i \leq \lfloor s \rfloor$, $|\mathcal{M}_i| \geq |\mathcal{M}|/2$.) $\mathcal{M}_i$ has a $\frac{1}{2}$-small subspace of size at least $(1-\gamma)|\mathcal{M}_i| \geq (1-(i+1)\gamma)|\mathcal{M}|$, which we take to be $\mathcal{M}_{i+1}$. □

Proof of Theorem 1.3. Let $g(n)$ denote the minimum competitive ratio over all n point spaces. Because of the uniform space, $g(n) \leq 2 \ln n$. We derive a recurrence inequality for $g(n)$. Let $\mathcal{M}$ be an n point space. Fix $s = s(n)$ and an integer $k = k(n)$ to be specified later and apply Lemma 6.5. If the second conclusion holds, then Lemma 6.1 implies that $\lambda(\mathcal{M}) \geq \frac{1}{4} + \ln s$. If the third conclusion of Lemma 6.5 holds, we want to apply Corollary 6.3 with $\beta = g(\lceil \frac{n}{2s^{k+2}} \rceil)$ to conclude that $\lambda(\mathcal{M}) \geq g(\lceil \frac{n}{2s^{k+2}} \rceil) + 1/2$. To apply Corollary 6.3 it suffices that $2^{-k} \leq 1/(2200(2 \ln n)^3)$ (using the fact that $g(n) \leq 2 \ln n$)). So we choose $k = \lceil 20 + 3 \log \ln n \rceil \leq 20 + 6 \log \log n$. Finally, if the first conclusion of Lemma 6.5 holds, then we want to apply Corollary 6.4 with $\beta = g(\lceil n(1-\frac{1}{s}) \rceil)$ to conclude $\lambda(\mathcal{M}) \geq g(\lceil n(1-\frac{1}{s}) \rceil) + e^{-2g(\lceil n(1-\frac{1}{s}) \rceil)}$. To apply Corollary 6.4 it suffices that $2^{1-s} \leq 1/(27000 e^{6 \ln n})$ which holds if $s \geq 6 \log n + 20$. Thus under this assumption on s,

$$g(n) \geq \min \left\{ \frac{\log s}{4}, g\left(\left\lceil n\left(1 - \frac{1}{s}\right) \right\rceil \right) + e^{-2g(\lceil n(1-\frac{1}{s}) \rceil)}, g\left(\left\lceil \frac{n}{2s^{6 \log \log n + 22}} \right\rceil \right) + \frac{1}{2} \right\}. \tag{6.1}$$

It now suffices to choose $s(n)$ satisfying the above condition, guess a function that lower bounds $g(n)$, and use the recurrence to verify the lower bound.

We choose $s(n) = 2^{C_0\sqrt{\log n/\log\log n}}$, where $C_0 > 0$ is a small constant to be chosen later. We choose $n_0 = n_0(C_0)$ so that for $n \geq n_0$, $s(n) \geq 6\log n + 20$. Finally set $h(n) = C_1 \log s(n)$, where $C_1 > 0$ is chosen to be at most $1/4$ and also small enough so that for $3 \leq n \leq n_0$, $h(n) \leq 1$.

We claim that $g(n) \geq h(n)$ for all $n \geq 3$, which suffices to prove the theorem. We proceed by induction on n.

For $n \leq n_0$, the result is trivial since $g(n) \geq 1$. So assume $n > n_0$. Applying the recurrence (6.1) and the induction hypothesis, we get

$$g(n) \geq \min\left\{\frac{\log s}{4}, h\left(\left\lceil n\left(1-\frac{1}{s}\right)\right\rceil\right) + e^{-2h(\lceil n(1-\frac{1}{s})\rceil)}, h\left(\left\lceil \frac{n}{2s^{6\log\log n+22}}\right\rceil\right) + \frac{1}{2}\right\}. \tag{6.2}$$

In substituting h for g in the second term, we observe that $x + e^{-2x}$ increases with x for $x \geq \frac{\ln 2}{2}$ and is at most 1 for $x \in [0, \frac{\ln 2}{2}]$. Next, since h increases with n, we can drop the $\lceil\cdot\rceil$:

$$g(n) \geq \min\left\{\frac{\log s}{4}, h\left(n\left(1-\frac{1}{s}\right)\right) + e^{-2h(n(1-\frac{1}{s}))}, h\left(\frac{n}{2s^{6\log\log n+22}}\right) + \frac{1}{2}\right\}. \tag{6.3}$$

Now it suffices to show that each of the terms in the minimum is at least $h(n)$. This is true for the first term since $C_1 \leq 1/4$. For the second and third terms, we bound $h(n) - h(n/B)$ for $B > 1$ by the following chain of inequalities:

$$\begin{aligned} h(n) - h(n/B) &= C_1C_0\sqrt{\frac{\log n}{\log\log n}} - C_1C_0\sqrt{\frac{\log(n/B)}{\log\log(n/B)}} \\ &\leq C_1C_0\sqrt{\frac{\log n}{\log\log n}}\left(1 - \sqrt{1-\frac{\log B}{\log n}}\right) \\ &\leq C_1C_0\sqrt{\frac{\log n}{\log\log n}}\frac{\log B}{\log n} \\ &= C_1C_0\frac{\log B}{\sqrt{\log n\log\log n}}. \end{aligned}$$

To show that the second term in (6.3) is at least $h(n)$ we take $B = s/(s-1)$ in the above inequality. The final expression is then at most $C_1C_0\frac{1}{s\sqrt{\log n\log\log n}}$. We need to show that this is at most $e^{-2h(n(1-1/s))}$. This is true, as $C_1C_0\frac{1}{s\sqrt{\log n\log\log n}} \leq 1/s \leq 1/s^{2C_1/\ln 2} = e^{-2h(n)} \leq e^{-2h(n(1-1/s))}$, where the second inequality uses $C_1 \leq 1/4$.

To show that the third term in (6.3) is at least $h(n)$ take $B = 2s^{6\log\log n+22}$ in the above inequality. The final expression in the inequality is then at most $C_1C_0^2(6\log\log n + 22)/\log\log n$, which is at most $1/2$ for sufficiently small C_0. ∎

7. Proof of technical lemmas. In this section, we present the proofs of Lemmas 5.2 and 5.3. Throughout this section, $\mathcal{M}$ is a fixed finite metric space with distance function d and diameter δ. The minimum distance between (distinct) points in $\mathcal{M}$ is denoted $\delta_{\min}$.

We begin with some facts about the topological structure of the set of randomized

evader algorithms.[1] Recall from section 2.3 that an algorithm can be defined as a function that labels each E-node of $T_{\mathcal{M}}$ by a probability distribution on $\mathcal{M}$. The set $P(\mathcal{M})$ of probability distributions on $\mathcal{M}$ can be viewed as a topological subspace of Euclidean space $\mathbf{R}^{\mathcal{M}}$, and so the set of randomized algorithms can be viewed as a product of copies of this space (where the product is indexed by the E-nodes of $T_{\mathcal{M}}$).

The topological facts we need are summarized in the following proposition.

PROPOSITION 7.1.

1. *Any sequence of randomized evader algorithms has a subsequence that converges.*
2. *Suppose that the sequence $\tilde{A}_1, \tilde{A}_2, \ldots$ of algorithms converges to algorithm $\tilde{A}$ in the product topology. If ρ is any probe sequence, then the sequence of real numbers $\tilde{A}_1(\rho), \tilde{A}_2(\rho), \ldots$ converges to $\tilde{A}(\rho)$.*

Proof. The topological space of algorithms is equivalent to the product of a countable number of spaces isomorphic to $P(\mathcal{M})$. Notice that $P(\mathcal{M})$ is metrizable and compact. By Tychonoff's theorem (see [Kel, pp. 143–144]), the product of compact topological spaces is compact with respect to the product topology. Moreover, the product of a countable number of metrizable topological spaces is metrizable (see [Kel, p. 122]). The first part follows from the fact that every sequence in a compact metrizable topological space has a subsequence that converges to a point in the space (see [Kel, pp. 138–139]). For the second part, we note that for a probe sequence ρ, the mapping from the set of randomized algorithms to the reals, defined by $\tilde{A} \longrightarrow \tilde{A}(\rho)$ is continuous. □

We need a modification of the PE game. Fix the metric space $\mathcal{M}$ and let λ denote the competitive ratio of the PE game. We define the *modified game* for $\mathcal{M}$ to be a game whose strategy sets are the same as for the PE game but whose cost function is $f_A(\rho) = C_A(\rho) - \lambda C_{\mathrm{OPT}}(\rho)$. For $s > 0$, let G_s (resp., H_s) denote the game obtained from the modified game by restricting the strategies of the pursuer to be probe sequences ρ satisfying $C_{\mathrm{OPT}}(\rho) \leq s$ (resp., satisfying $s - \delta \leq C_{\mathrm{OPT}}(\rho) \leq s$). We will prove the following lemma.

LEMMA 7.1. *For every $s > 0$,*

1. *the games G_s and H_s have the min-max property, and*
2. *the values of G_s and H_s satisfy*

$$-\delta \leq V(H_s) \leq V(G_s) \leq \lambda\delta.$$

Assuming this lemma, it is easy to prove the two main technical lemmas.

Proof of Lemma 5.2. For $s > 0$, let $\tilde{A}_s$ denote an optimal strategy for G_s. By the first part of Proposition 7.1, there is a an infinite sequence $i_1 < i_2 < \cdots$ of positive integers such that the sequence $\tilde{A}_{i_j}$ converges to a limit algorithm $\tilde{A}$. If ρ is any probe sequence and j is chosen such that $i_j \geq C_{\mathrm{OPT}}(\rho)$, the second part of

[1] In the following discussion, we use the following concepts from point set topology (see, e.g., [Kel]): A set system $\mathcal{F}$ is a *topology* if and only if the union of any number of elements of $\mathcal{F}$ is in $\mathcal{F}$ and the intersection of a finite number of elements of $\mathcal{F}$ is in $\mathcal{F}$. The pair $(X, \mathcal{F})$, where $X = \cup F \in \mathcal{F}$, is called a *topological space*. The elements of $\mathcal{F}$ are called *open sets*. If $Y \subset X$, then $(Y, \mathcal{G})$, where $\mathcal{G} = \{F \cap Y;\ F \in \mathcal{F}\}$, is a *topological subspace* of $(X, \mathcal{F})$. (Notice that a topological subspace is a topological space.) For a metric space $\mathcal{M} = (X, d)$, the associated *metric topology* $\mathcal{F}$ is derived by putting $\mathcal{F}$ to be the set of all the unions of open balls in $\mathcal{M}$. A topological space that can be derived this way is *metrizable*. Any subset of $\mathcal{F}$ whose union equals X is an *open cover* of the topological space $(X, \mathcal{F})$. A topological space is *compact* if and only if every open cover has a finite subset that is also an open cover. If $(X, \mathcal{F})$ and $(Y, \mathcal{G})$ are topological spaces, then a function $f : X \to Y$ is *continuous* if and only if for every $G \in \mathcal{G}$, $f^{-1}(G) \in \mathcal{F}$.

Lemma 7.1 implies $C_{\tilde{A}_{i_j}}(\rho) - \lambda C_{\mathrm{OPT}}(\rho) \le V(G_{i_j}) \le \lambda\delta$. Letting i_j tend to ∞ yields $C_{\tilde{A}}(\rho) - \lambda C_{\mathrm{OPT}}(\rho) \le \lambda\delta$, as required. □

Proof of Lemma 5.3. Fix $s > 0$ and let $\tilde{\rho}_s$ denote an optimal strategy for the pursuer for the game H_s. Then for any evader algorithm $\tilde{A}$, we have $C_{\tilde{A}}(\rho_s) - \lambda C_{\mathrm{OPT}}(\rho) \ge -\delta$, or $C_{\tilde{A}}(\rho_s) \ge \lambda(s-\delta) - \delta$, as required. □

The remainder of the section proves Lemma 7.1. To prove the first part of the lemma, it would be enough, by Theorem 2.2.1, to show that the set of pure strategies is finite. Unfortunately, it is not, in general, true that the set of pure strategies is finite. However, we will reduce these games to an analysis of games with finite strategy sets. More precisely, we will define the notion of a *standard* probe sequence and a *standard* evader algorithm and show that the restrictions of the games G_s and H_s to these sets are essentially finite games and that the the analysis of G_s and H_s can be reduced to that of the restricted game.

As a preliminary restriction, we say that an evader algorithm A is *lazy* if it does not move unless it has to; i.e., the evader moves from its current location only if the pursuer probes the location he occupies. It is easy to show and well known (see [BE]) that the set of lazy algorithms dominates the set of all algorithms in the sense of section 2.2. (Actually, the versions of this result that appear in the literature prove the dominance with respect to the cost function $C_A(\rho)$ rather than $f_A(\rho)$, but it is easy to show that the one result implies the other). Thus, applying Lemma 2.1 (2.1), it suffices to restrict the evader to lazy algorithms, which we do from now on.

By the definition of a lazy algorithm, if the evader is at point a after probe sequence ρ, the evader will move only if the next probe is to a. Thus for a lazy deterministic algorithm A, each probe sequence ρ induces a *transition function* $A[\rho]$ mapping the metric space to itself where $A[\rho](a)$ is the point the evader moves to if it is at a after ρ and the next probe is to a.

Before we can define the notion of standard algorithm and standard probe sequence, we need to review some basic facts about the PE game and the function C_{OPT}. For the PE game, C_{OPT} can be expressed as follows. For probe sequence ρ and point $p \in M$, let $C_A(\rho; p)$ denote the cost incurred by A in responding to ρ and then moving to point p; this is $C_A(\rho) + d(a, p)$, where a is the final point of $A(\rho)$. Let $C_{OPT}(\rho; p)$ denote the minimum of $C_A(\rho; p)$ over all algorithms A. Then $C_{\mathrm{OPT}}(\rho) = \min\{C_{\mathrm{OPT}}(\rho; p) : p \in \mathcal{M}\}$. For ρ equal to the empty string, we have $C_{\mathrm{OPT}}(\rho; p) = 0$. For any ρ and any points a, p, we have

$$C_{\mathrm{OPT}}(\rho a; p) = \begin{cases} C_{\mathrm{OPT}}(\rho; p) & \text{if } a \neq p, \\ \min_{q \neq p} C_{\mathrm{OPT}}(\rho; q) + d(q, a) & \text{if } a = p. \end{cases}$$

For each ρ, $C_{\mathrm{OPT}}(\rho; p)$ is a function mapping $p \in \mathcal{M}$ to the nonnegative reals. This is the well known *work function* associated with this game (see, e.g., [BE]) and will be denoted $WF[\rho]$.

It is easy to check that if τ is an extension of ρ, then $WF[\tau] \ge WF[\rho]$, where the inequality of work functions is defined pointwise. τ is a *null extension* of ρ if they have the same work function. A point a is said to be *null* with respect to a sequence ρ if ρa is a null extension of ρ. Let $N(\rho)$ denote the set of points that are null with respect to ρ.

We state a few easy facts without proof.

PROPOSITION 7.2. *Let $\rho = (\rho_1, \ldots, \rho_k)$ be an arbitrary probe sequence.*

1. *$\rho_k \in N(\rho)$.*
2. *If τ has the same work function as ρ, then $N(\rho) = N(\tau)$. In particular, if $a \in N(\rho)$, then $N(\rho a) = N(\rho)$.*
3. *If a is any point that minimizes $WF[\rho](\cdot)$, then $a \notin N(\rho)$. In particular, there is at least one point that is not null with respect to ρ.*

A sequence $\rho = (\rho_1, \ldots, \rho_k)$ is said to be *standard* if the work functions associated to its prefixes are all different; equivalently, for each i between 2 and k $\rho_i \notin N(\rho_1, \ldots, \rho_{i-1})$.

A *standard* algorithm A is a lazy algorithm that ignores null points. More precisely, a standard algorithm never moves to a point that is null with respect to the current sequence, and ignores requests to null points in the sense that its present and future behavior is unaffected by such requests.

Let G'_s (resp., H'_s) denote the game obtained from G_s (resp., H_s) by restricting to standard probe sequences and standard evader algorithms.

PROPOSITION 7.3. *For each $s > 0$ there is an integer $t(s)$ with the property that any standard probe sequence of optimal cost at most s has term length at most $t(s)$.*

Proof. Let $\rho = (\rho_1, \ldots, \rho_t)$ be a standard probe sequence. Let w^i denote the work function associated to the prefix ρ^i. For each i, $w^i(p) = w^{i-1}(p)$ if $p \neq \rho_i$ and $w^i(\rho_i) > w^{i-1}(\rho_{i-1})$. Let $\Delta_i = w^i(\rho_i) - w^{i-1}(\rho_{i-1})$. Induction on s yields $\sum_p w^s(p) = \sum_{i=1}^s \Delta_i$. Since the maximum and minimum work function values associated to any probe sequence differ by at most δ, $w^i(p) \geq \frac{1}{n}(\sum_{i=1}^s \Delta_i) - \delta$. Recall that $\delta_{\min}$ is the minimum distance between two distinct points in the metric space. We will show the following.

CLAIM. *In any subsequence of n^n consecutive indices there is an i such that $\Delta_i \geq \delta_{\min}$.*

This implies that for any ρ of length t, $w^i(\rho) \geq \lfloor \frac{t}{n^n} \rfloor \delta_{\min}$ and so for any $s > 0$ we can choose $t(s)$ so that for $|\rho| \geq t(s)$, the minimum work function value is at least s.

So we prove the claim. Fix an arbitrary ordering $<$ on $\mathcal{M}$. For each i, let $p_1^i, \ldots, p_n^i$ denote the sequence of points ordered so that $w^i(p_r^i) \leq w^i(p_{r+1}^i)$ with ties broken according to the ordering $<$. Let Γ_i denote the directed graph on vertex set $\mathcal{M}$, with arcs $p \longrightarrow_i q$ if $w^i(q) = w^i(p) + d(p, q)$. (Note that a point p is null with respect to ρ^i precisely if its in-degree in Γ_i is positive.) Define $\mathbf{d}^i = (d_1^i, \ldots, d_n^i)$ with d_r^i equal to the outdegree of p_r^i in Γ_i. We will show that for each i, if $\Delta_i < \delta_{\min}$, then the sequence $\mathbf{d}^i$ is lexicographically greater than $\mathbf{d}^{i-1}$. Since there are at most n^n distinct out-degree sequences, this will prove the claim and the proposition.

Γ_i differs from Γ_{i-1} only on the arcs incident on ρ_i. The in-degree of ρ_i in Γ_{i-1} is 0 since ρ_i is nonnull with respect to ρ^{i-1} and is positive in Γ_i since ρ_i is null with respect to ρ^i. Let h be an index such that $p_h^{i-1} \longrightarrow_i \rho_i$ and let j be the index such that $\rho_i = p_j^{i-1}$. By definition of Γ_i, $w^i(\rho_i) = w^i(p_h^{i-1}) + d(p_h^{i-1}, \rho_i) \geq w^{i-1}(p_h^{i-1}) + \delta_{\min}$.

If $h > j$, then $w^{i-1}(\rho_i) \leq w^{i-1}(p_h^{i-1})$ and we conclude $w^i(\rho_i) - w^{i-1}(\rho_i) \geq \delta_{\min}$.

So suppose that $h < j$. For $r < j$, $p_r^i = p_r^{i-1}$ and all edges out of p_r^{i-1} in Γ_{i-1} are present in Γ_i. Thus $d_r^i \geq d_r^{i-1}$. For $r = h$, we have strict inequality since $d_h^i \longrightarrow_i \rho_i$. This implies that $\mathbf{d}^i$ is lexicographically greater than $\mathbf{d}^{i-1}$. □

An immediate consequence of Proposition 7.3 is that the number of standard probe sequences of cost at most s is finite. Also, although the number of deterministic evader algorithms is infinite, if we call two algorithms equivalent if they respond identically to any probe sequence of term length at most $t(s)$, then the number of equivalence classes is finite. Hence G'_s (resp., H'_s) is essentially a finite game, and

thus by Theorem 2.2.1 it has the min-max property. We will show that the game G_s (resp., H_s) is "essentially" the same as G'_s (resp., H'_s), to prove the first part of Lemma 7.1. For ease of notation, we refer only to G_s in this argument. The argument holds as well if we replace G_s by H_s and G'_s by H'_s.

For an evader algorithm $\tilde{A}$, we write $V_{\mathrm{MIN}}(\tilde{A})$ for the value of $\tilde{A}$ in the game G_s, i.e., the supremum over all probe sequences ρ of optimal cost at most s of $f_{\bar{A}}(\rho)$. We write $V'_{\mathrm{MIN}}(\tilde{A})$ for the value of $\tilde{A}$ in the game G'_s. Similarly, for a randomized probe sequence $\tilde{\rho}$, we write $V_{\mathrm{MAX}}(\tilde{\rho})$ and $V'_{\mathrm{MAX}}(\tilde{\rho})$, respectively, for the value of $\tilde{\rho}$ with respect to the games G_s and G'_s.

PROPOSITION 7.4. *For any randomized standard evader algorithm $\tilde{A}$, $V_{\mathrm{MIN}}(\tilde{A}) = V'_{\mathrm{MIN}}(\tilde{A})$.*

Proof. Let $\rho = (\rho_1, \ldots, \rho_k)$ be an arbitrary probe sequence. Write w^i for the work function associated to the prefix ρ^i. We observed earlier that $w^1 \leq w^2 \leq \cdots \leq w^k$ (where the inequality is pointwise). Define the sequence of indices $i_1 < i_2 < \cdots < i_j$, where $i_1 = 1$, i_2 is the least i such that $w^{i_2} \neq w^{i_1}$ and, in general, i_h is the least i such that $w^{i_h} \neq w^{i_{h-1}}$. It is easy to see that the sequence $\hat{\rho} = \rho_{i_1}\rho_{i_2}\ldots\rho_{i_j}$ is standard and has the same work function as ρ.

Now let $\tilde{A}$ be a randomized standard algorithm. Since $\tilde{A}$ ignores all probes to null points, it behaves the same on ρ as it does on $\hat{\rho}$. More precisely, the probability that $\tilde{A}$ responds to $\hat{\rho}$ with the sequence $\sigma_1\sigma_2\ldots\sigma_j$ is equal to the probability that it responds to ρ with $\sigma_1^{i_2-1}\sigma_2^{i_3-i_2}\ldots\sigma_{j-1}^{i_j-i_{j-1}}\sigma_j^{n+1-i_j}$, which has the same cost as $\sigma_1\ldots\sigma_j$. □

Given an analogous result for pursuer strategies, the first part of Lemma 7.1 would follow immediately. However, such a result is not true: if $\tilde{\rho}$ is a randomized standard pursuer strategy, then since the pursuer never probes null points, a nonstandard algorithm might be able to avoid the pursuer by moving to null points.

So we need to prove something a little subtler about pursuer strategies. The key property of a null point is that the pursuer can probe a null point without increasing the work function, and thus probes to a null point are "free" to the pursuer. Thus we can modify any standard pursuer strategy so that it probes null points often enough to ensure that it does not benefit the evader to ever visit a null point. If we modify the optimal standard pursuer strategy in this way, we will get a strategy whose value in the game G_s is the same as that of the optimal standard strategy in the game G'_s.

We now make this argument precise. Let $\rho = (\rho_1, \ldots, \rho_k)$ be a probe sequence. For i between 1 and k, Let $\eta_{\mathbf{i}}$ be some fixed ordering on $N(\rho^i)$, the set of points that are null with respect to ρ^i. For a nonnegative integer j, we define the *j-fold inflation of ρ* to be the sequence $\rho_1{\eta_{\mathbf{1}}}^j\rho_2{\eta_{\mathbf{2}}}^j\ldots\rho_k{\eta_{\mathbf{k}}}^j$. It is trivial that the work functions associated with a sequence and its j-fold inflation are identical.

If $\tilde{\rho}$ is a randomized pursuer strategy, i.e., a probability distribution over probe sequences, then the j-fold inflation of $\tilde{\rho}$ is the distribution obtained by selecting a sequence according to $\tilde{\rho}$ and applying j-fold inflation to it.

PROPOSITION 7.5. *For j a sufficiently large integer (depending on $\mathcal{M}$ and s) the following holds. For any deterministic lazy evader algorithm A there is a deterministic standard algorithm $\bar{A}$ such that for any randomized pursuer strategy $\tilde{\rho}$ in the game G_s, if $\tilde{\sigma}$ is the j-fold inflation of ρ, then $C_{\bar{A}}(\tilde{\rho}) \leq C_A(\tilde{\sigma})$. Consequently, $V'_{\mathrm{MAX}}(\tilde{\rho}) \leq V_{\mathrm{MAX}}(\tilde{\sigma})$.*

Before proving the proposition, let us see that it implies the first part of Lemma 7.1. Let $\tilde{A}^*$ and $\tilde{\rho}^*$ be the optimal strategies of evader and pursuer in the game G'_s, and let $\tilde{\sigma}^*$ be the j-fold inflation of $\tilde{\rho}^*$ (where j is large enough for Proposition 7.5). From Proposition 7.4, $V_{\mathrm{MIN}}(\tilde{A}^*) = V'_{\mathrm{MIN}}(\tilde{A}^*)$. From Proposition 7.5, $V'_{\mathrm{MAX}}(\tilde{\rho}^*) \leq$

$V_{\mathrm{MAX}}(\tilde{\sigma}^*)$. Since G'_s has the min-max property, we have that $V'_{\mathrm{MIN}}(\tilde{A}^*) = V'_{\mathrm{MAX}}(\tilde{\rho}^*)$. Hence $V_{\mathrm{MIN}}(\tilde{A}^*) \leq V_{\mathrm{MAX}}(\tilde{\sigma}^*)$, and combining this with Lemma 2.2, we conclude that $V_{\mathrm{MIN}}(\tilde{A}^*) = V_{\mathrm{MAX}}(\tilde{\sigma}^*)$, i.e., G_s has the min-max property.

Proof of Proposition 7.5. Fix j sufficiently large. Given A, we need to define $\bar{A}$. The behavior of A on a probe sequence $\rho = (\rho_1, \ldots, \rho_k)$ is defined using an on-line simulation of A applied to the j-fold inflation $\sigma = \rho_1 \eta_{\mathbf{1}}{}^j \ldots \rho_k \eta_{\mathbf{k}}{}^j$ of ρ. Prior to the ith request, $\bar{A}$ will have processed $\rho_1, \ldots, \rho_{i-1}$ and responded with $\sigma_1 \sigma_2 \ldots \sigma_{i-1}$. It will also have simulated A on the j-fold inflation of $\rho_1, \ldots, \rho_{i-1}$. Upon receipt of ρ_i, it continues the simulation by giving A the sequence $\rho_i \eta_{\mathbf{i}}{}^j$. Let τ_i denote the final point A occupies after processing that sequence.

$\bar{A}$ then chooses its response σ_i according to the following rule. If $\sigma_{i-1} \neq \rho_i$, then $\sigma_i = \sigma_{i-1}$ (following laziness). Otherwise, if τ_i is not null with respect to $\rho_1 \ldots \rho_i$, then $\sigma_i = \tau_i$, else $\bar{A}$ moves to any nonnull point (say, the least one under some predetermined ordering).

It is immediate that $\bar{A}$ is a standard algorithm. We claim that $C_{\bar{A}}(\rho) \leq C_A(\sigma)$. This then extends by linearity to the case of randomized pursuer strategies.

We say that A is *well behaved* on $\rho_1, \rho_2, \ldots, \rho_k$ provided that for each i, the point τ_i is not null with respect to $\rho_1 \ldots \rho_i$. By the definition of $\bar{A}$ (and the fact that A itself is lazy), if A is well behaved, then the response sequence $\sigma_1, \ldots, \sigma_k$ of $\bar{A}$ is just $\tau_1, \tau_2, \ldots, \tau_k$ and is thus a subsequence of A's response sequence. Hence $C_{\bar{A}}(\rho) \leq C_A(\sigma)$.

If A is not well behaved, then for some i, τ_i is null with respect to $\rho_1 \ldots \rho_i$. Consider the responses of A to $\rho_i \eta_i^j$. If A ever responded with a point that is not null with respect to $\rho_1 \ldots \rho_i$, then since A is lazy and all requests appearing in η_i are null, A would have ended in a nonnull point. So it must be that each of A's responses to this subsequence was a null point. Now since η_i contains each null point once, A was forced to move at least j times. Thus $c_A(\sigma) \geq j\delta_{\min}$. Taking j large enough (depending on $\mathcal{M}$ and s) ensures that this is at least $C_{\bar{A}}(\rho)$. □

This completes the proof of the first part of Lemma 7.1. We proceed to the proof of the second part. We state without proof a routine fact concerning the function C_{OPT}.

LEMMA 7.2. *Let $\mathcal{M}$ be a metric space of diameter δ, and let $\rho_{\mathbf{1}}, \rho_{\mathbf{2}}, \ldots, \rho_{\mathbf{w}}$ be probe sequences. Then*

$$\sum_{i=1}^{w} C_{\mathrm{OPT}}(\rho_{\mathbf{i}}) \leq C_{\mathrm{OPT}}(\rho_{\mathbf{1}} \rho_{\mathbf{2}} \ldots \rho_{\mathbf{w}}) \leq \sum_{i=1}^{w} C_{\mathrm{OPT}}(\rho_{\mathbf{i}}) + (w-1)\delta.$$

Furthermore, the inequalities extend to the case that the ρ_i are replaced by randomized probe sequences $\tilde{\rho}_{\mathbf{i}}$.

(Recall that the definition of C_{OPT} allows the evader to choose its own starting point. So if each $\rho_{\mathbf{i}}$ misses some point in $\mathcal{M}$, then $\sum_{i=1}^{w} C_{\mathrm{OPT}}(\rho_{\mathbf{i}}) = 0$, and if all $\rho_{\mathbf{i}}$ miss the *same* point in $\mathcal{M}$, then $C_{\mathrm{OPT}}(\rho_{\mathbf{1}} \rho_{\mathbf{2}} \ldots \rho_{\mathbf{w}}) = 0$.)

We now prove the second part of Lemma 7.1. The middle inequality is trivial since the only difference between G_s and H_s is that the pursuer strategy set in H_s is a subset of that in G_s.

Consider the first inequality and suppose for contradiction that there is an $s > 0$ and an $\epsilon > 0$ such that $V(H_s) = -\delta - \epsilon$. Let $\tilde{B}_s$ denote an optimal evader algorithm for H_s. We define an evader algorithm $\tilde{B}$ for the original PE game as follows: for probe sequence τ determine its s-block partition, $\tau_{\mathbf{1}} \tau_{\mathbf{2}} \ldots \tau_{\mathbf{r}}$, which can be parsed on-line. The algorithm $\tilde{B}$ is performed by applying $\tilde{B}_s$ to each $\tau_{\mathbf{i}}$. We obtain a contradiction

by showing that $\tilde{B}$ is λ'-competitive for some $\lambda' < \lambda$. There is some absolute upper bound K on the cost incurred by $\tilde{B}_s$ on a sequence of optimal cost at most s. We have

$$\begin{aligned} C_{\tilde{B}}(\tau) &\le \sum_{i=1}^{r-1} (C_{\tilde{B}_s}(\tau_{\mathbf{i}}) + \delta) + C_{\tilde{B}_s}(\tau_{\mathbf{r}}) \\ &\le \sum_{i=1}^{r-1} (\lambda C_{\mathrm{OPT}}(\tau_{\mathbf{i}}) - \epsilon) + K \\ &\le \sum_{i=1}^{r-1} \left(\lambda - \frac{\epsilon}{s}\right) C_{\mathrm{OPT}}(\tau_{\mathbf{i}}) + K \\ &\le \left(\lambda - \frac{\epsilon}{s}\right) C_{\mathrm{OPT}}(\tau) + K, \end{aligned}$$

where the first and last inequalities follow from Lemma 7.2, the second inequality follows from the optimality of $\tilde{B}_s$ for H_s, and the third inequality follows from the fact that each τ_i has optimal cost at most s. We conclude that $\tilde{B}$ is $(\lambda - \frac{\epsilon}{s})$-competitive, a contradiction that completes the proof of the first inequality.

Turning to the third inequality, we suppose for contradiction that there is an $s > 0$ and an $\epsilon > 0$ such that $V(G_s) \ge \lambda\delta + \epsilon$. Let $\tilde{\rho}_s$ be the optimal pursuer strategy for G_s. Then for any algorithm $\tilde{B}$, $C_{\tilde{B}}(\tilde{\rho}_s) \ge \lambda(C_{\mathrm{OPT}}(\tilde{\rho}_s) + \delta) + \epsilon$. For each $j \in \mathbb{N}$, define the distribution $\tilde{\tau}_{\mathbf{j}}$ on probe sequences obtained by concatenating j sequences generated independently from $\tilde{\rho}_s$. Then by Lemma 7.2, $C_{\mathrm{OPT}}(\tilde{\tau}_{\mathbf{j}})$ is bounded above by $u_j = j(C_{\mathrm{OPT}}(\tilde{\rho}_{\mathbf{s}}) + \delta)$. We will obtain a contradiction by showing that there exists a real number $\gamma > 0$ such that for any deterministic algorithm A, $C_A(\tilde{\tau}_{\mathbf{j}}) \ge u_j(\lambda + \gamma)$, which by Proposition 2.3 would imply that the competitive ratio is greater than $\lambda + \gamma$.

Let A be a deterministic algorithm and consider the cost of A on $\tilde{\tau}_{\mathbf{j}}$. By Lemma 7.2, we can lower bound this cost by the sum of costs that are incurred in responding to each of the j blocks. Note that the way that A responds to the ith block depends on the previous $i-1$ blocks, and we can view A's behavior on the ith block as a randomized algorithm $\tilde{B}_i$, where the randomization comes from the previous $i-1$ blocks. Thus

$$\begin{aligned} C_A(\tilde{\tau}_{\mathbf{j}}) &\ge \sum_{i=1}^{j} C_{\tilde{B}_i}(\tilde{\rho}_{\mathbf{s}}) \\ &\ge j[\lambda(C_{\mathrm{OPT}}(\tilde{\rho}_{\mathbf{s}}) + \delta) + \epsilon] \\ &= u_j \left(\lambda + \frac{\epsilon}{C_{\mathrm{OPT}}(\tilde{\rho}_{\mathbf{s}}) + \delta}\right), \end{aligned}$$

where the second inequality follows from the optimality of $\tilde{\rho}_s$. Thus we have the desired contradiction, which establishes the claim and the lemma.

Acknowledgments. We would like to thank the anonymous referees for the many useful comments in their reports, which helped improve the presentation of our results.

REFERENCES

[BBKTW] S. Ben-David, A. Borodin, R.M. Karp, G. Tardos, and A. Wigderson, *On the power of randomization in online algorithms*, Algorithmica, 11 (1994), pp. 2–14.

[BBBT] Y. Bartal, A. Blum, C. Burch, and A. Tomkins, *A polylog(n)-competitive algorithm for metrical task systems*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, ACM, New York, 1997, pp. 711–719.

[BLS] A. Borodin, N. Linial, and M. Saks, *An optimal on-line algorithm for metrical task systems*, J. ACM, 39 (1992), pp. 745–763.

[BE] A. Borodin and R. El Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge, UK, 1998.

[CDRS] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir, *Random walks on weighted graphs and applications to on-line algorithms*, J. ACM, 40 (1993), pp. 421–453.

[FKLMSY] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young, *Competitive paging algorithms*, J. Algorithms, 12 (1991), pp. 685–699.

[FRR] A. Fiat, Y. Rabani, and Y. Ravid, *Competitive k-server algorithms*, J. Comput. System Sci., 48 (1994), pp. 410–428.

[Gro] E. Grove, *The harmonic k-server algorithm is competitive*, in Proceedings of the 23rd Annual ACM Symposium on Theory of Computing, ACM, New York, 1991, pp. 260–266.

[KMRS] A.R. Karlin, M.S. Manasse, L. Rudolph, and D.D. Sleator, *Competitive snoopy caching*, Algorithmica, 3 (1988), pp. 79–119.

[KRR] H.J. Karloff, Y. Rabani, and Y. Ravid, *Lower bounds for randomized k-server and motion-planning algorithms*, SIAM J. Comput., 23 (1994), pp. 293–312.

[Kel] J.L. Kelley, *General Topology*, American Book Company, New York, 1955.

[KP] E. Koutsoupias and C. Papadimitriou, *On the k-server conjecture*, J. ACM, 42 (1995), pp. 971–983.

[LR] C. Lund and N. Reingold, *Linear programs for randomized on-line algorithms*, in Proceedings of the 5th Annual ACM–SIAM Symposium on Discrete Algorithms, Arlington, VA, ACM, New York, SIAM, Philadelphia, 1994, pp. 382–391.

[MMS] M.S. Manasse, L.A. McGeoch, and D.D. Sleator, *Competitive algorithms for on-line problems*, J. Algorithms, 11 (1990), pp. 208–230.

[Mat] J. Matoušek, *Ramsey-like properties for bi-Lipschitz mappings of finite metric spaces*, Comment. Math. Univ. Carolin., 33 (1992), pp. 451–463.

[MS] L.A. McGeoch and D.D. Sleator, *A strongly competitive randomized paging algorithm*, Algorithmica, 6 (1991), pp. 816–825.

[RS] P. Raghavan and M. Snir, *Memory versus randomization in on-line algorithms*, in Automata, Languages, and Programming (Stresa, 1989), Lecture Notes in Comput. Sci. 372, Springer-Verlag, Berlin, 1989, pp. 687–703.

[ST] D.D. Sleator and R.E. Tarjan, *Amortized efficiency of list update and paging rules*, Comm. ACM, 28 (1985), pp. 202–208.

[vNM] J. von Neumann and O. Morgenstern, *Theory of Games and Economic Behavior*, 2nd ed., Princeton University Press, Princeton, NJ, 1947.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1662–1677

LINEAR TIME ALGORITHMS FOR HAMILTONIAN PROBLEMS ON (CLAW,NET)-FREE GRAPHS*

ANDREAS BRANDSTÄDT†, FEODOR F. DRAGAN‡, AND EKKEHARD KÖHLER§

Abstract. We prove that claw-free graphs, containing an induced dominating path, have a Hamiltonian path, and that 2-connected claw-free graphs, containing an induced doubly dominating cycle or a pair of vertices such that there exist two internally disjoint induced dominating paths connecting them, have a Hamiltonian cycle. As a consequence, we obtain linear time algorithms for both problems if the input is restricted to (claw,net)-free graphs. These graphs enjoy those interesting structural properties.

Key words. claw-free graphs, (claw,net)-free graphs, Hamiltonian path, Hamiltonian cycle, dominating pair, dominating path, linear time algorithms

AMS subject classifications. 05C38, 05C45, 05C85, 05C75, 68R10

PII. S0097539799357775

1. Introduction.

Hamiltonian properties of claw-free graphs have been studied extensively in the last couple of years. Different approaches have been made, and a couple of interesting properties of claw-free graphs have been established (see [1, 2, 3, 5, 6, 13, 14, 15, 16, 19, 22, 23, 25, 26]). The purpose of this work is to consider the algorithmic problem of finding a Hamiltonian path or a Hamiltonian cycle efficiently. It is not hard to show that both the Hamiltonian path problem and the Hamiltonian cycle problem are NP-complete, even when restricted to line graphs [28]. Hence, it is quite reasonable to ask whether one can find interesting subclasses of claw-free graphs for which efficient algorithms for the above problems exist.

Already in the eighties, Duffus, Jacobson, and Gould [12] defined the class of (claw,net)-free (CN-free) graphs, i.e., graphs that contain neither an induced claw nor an induced net (see Figure 1.1). Although this definition seems to be rather restrictive, the family of CN-free graphs contains a couple of graph families that are of interest in their own right. Examples of those families are unit interval graphs, claw-free asteroidal triple-free (AT-free) graphs, and proper circular arc graphs. In their paper [12], Duffus, Jacobson, and Gould showed that this class of graphs has the nice property that every connected CN-free graph contains a Hamiltonian path and every 2-connected CN-free graph contains a Hamiltonian cycle. Later, Shepherd [27] proved that there is an $O(n^6)$ algorithm for finding such a Hamiltonian path/cycle in CN-free graphs. Note also that CN-free graphs are exactly the Hamiltonian-hereditary

*Received by the editors June 23, 1999; accepted for publication (in revised form) January 13, 2000; published electronically November 28, 2000. An extended abstract of these results has been presented at the 25th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 1665, Springer-Verlag, New York, 1999, pp. 364–376.

http://www.siam.org/journals/sicomp/30-5/35777.html

†Fachbereich Informatik, Universität Rostock, A. Einstein Str. 21, D-18051 Rostock, Germany (ab@informatik.uni-rostock.de).

‡Department of Mathematics and Computer Science, Kent State University, Kent, OH 44242 (dragan@mcs.kent.edu). The research of this author was supported by the German National Science Foundation (DFG).

§Fachbereich Mathematik, Technische Universität Berlin, Straße des 17. Juni 136, D-10623 Berlin, Germany (ekoehler@math.TU-Berlin.DE). The research of this author was supported by the graduate program "Algorithmic Discrete Mathematics," grant GRK 219/2-97 of the German National Science Foundation (DFG).

graphs [10], i.e., the graphs for which every connected induced subgraph contains a Hamiltonian path.

In this paper we give a constructive existence proof and present linear time algorithms for the Hamiltonian path and Hamiltonian cycle problems on CN-free graphs. The important structural property that we exploit for this is the existence of an induced *dominating path* in every connected CN-free graph (Theorem 2.3). The concept of a dominating path was first used by Corneil, Olariu, and Stewart [8] in the context of AT-free graphs. They also developed a simple linear time algorithm for finding such a path in every AT-free graph [7]. As we show in Theorem 2.3, for the class of CN-free graphs, a linear time algorithm for finding an induced dominating path exists as well. This property is of interest for our considerations since we prove that all claw-free graphs that contain an induced dominating path have a Hamiltonian path (Theorem 3.1). The proof implies that, given a dominating path, one can construct a Hamiltonian path for a claw-free graph in linear time.

For 2-connected claw-free graphs, we show that the existence of a dominating pair is sufficient for the existence of a Hamiltonian cycle. (A *dominating pair* is a pair of vertices such that every induced path connecting them is a dominating path.) Again, given a dominating pair, one can construct a Hamiltonian cycle in linear time (Theorem 5.6). This already implies, for example, a linear time algorithm for finding a Hamiltonian cycle in claw-free AT-free graphs, since every AT-free graph contains a dominating pair and it can be found in linear time [9]. Unfortunately, CN-free graphs do not always have a dominating pair. For example, an induced cycle with more than six vertices is CN-free but does not have such a pair of vertices. Nevertheless, 2-connected CN-free graphs have another nice property: they have a good pair or an induced doubly dominating cycle. An *induced doubly dominating cycle* is an induced cycle such that every vertex of the graph is adjacent to at least two vertices of the cycle. A *good pair* is a pair of vertices, such that there exist two internally disjoint induced dominating paths connecting these vertices. We prove that the existence of an induced doubly dominating cycle or a good pair in a claw-free graph is sufficient for the existence of a Hamiltonian cycle (Theorems 5.1 and 5.5). Moreover, given an induced doubly dominating cycle or a good pair of a claw-free graph, a Hamiltonian cycle can be constructed in linear time. In section 4 we present an $O(m+n)$ time algorithm which, for a given 2-connected CN-free graph, finds either a good pair or an induced doubly dominating cycle.

For terms not defined here, we refer to [11, 17]. In this paper we consider finite connected undirected graphs $G = (V, E)$ without loops and multiple edges. The cardinality of the vertex set is denoted by n, whereas the cardinality of the edge set is denoted by m.

A *path* is a sequence of vertices $(v_0, \ldots, v_l)$ such that all v_i are distinct and $v_i v_{i+1} \in E$ for $i = 0, \ldots, l-1$; its *length* is l. An *induced path* is a path where $v_i v_j \in E$ if and only if $i = j-1$ and $j = 1, \ldots, l$. A *cycle* (*k-cycle*) is a path $(v_0, \ldots, v_k)$ $(k \geq 3)$ such that $v_0 = v_k$; its *length* is k. An *induced cycle* is a cycle where $v_i v_j \in E$ if and only if $|i-j| = 1$ (modulo k). A *hole* H_k is an induced cycle of length $k \geq 5$.

The *distance* $dist(v, u)$ between vertices v and u is the smallest number of edges in a path joining v and u. The *eccentricity* $ecc(v)$ of a vertex v is the maximum distance from v to any vertex in G. The *diameter* $diam(G)$ of G is the maximum eccentricity of a vertex in G. A pair v, u of vertices of G with $dist(v, u) = diam(G)$ is called a *diametral pair*.

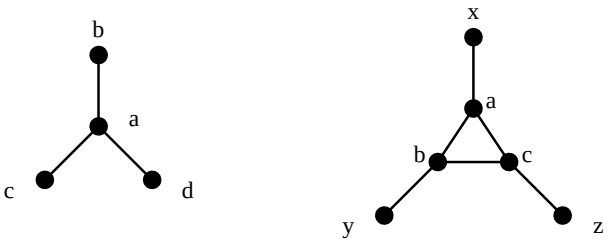


FIG. 1.1. *The claw* $K(\mathrm{a};\mathrm{b},\mathrm{c},\mathrm{d})$ *and the net* $N(\mathrm{a},\mathrm{b},\mathrm{c};\mathrm{x},\mathrm{y},\mathrm{z})$.

For every vertex we denote by $\mathrm{N}(v)$ the set of all neighbors of v, $\mathrm{N}(v) = \{u \in V : dist(u,v) = 1\}$. The *closed neighborhood* of v is defined by $\mathrm{N}[v] = \mathrm{N}(v) \cup \{v\}$. For a vertex v and a set of vertices $S \subseteq V$, the minimum distance between v and vertices of S is denoted by $dist(v,S)$. The *closed neighborhood* $\mathrm{N}[S]$ of a set $S \subseteq V$ is defined by $\mathrm{N}[S] = \{v \in V : dist(v,S) \leq 1\}$.

We say that a set $S \subseteq V$ *dominates* G if $\mathrm{N}[S] = V$, and S *doubly dominates* G if every vertex of G has at least two neighbors in S. An induced path of G which dominates G is called an *induced dominating path*. A shortest path of G which dominates G is called a *dominating shortest path*. Analogously one can define an *induced dominating cycle* of G. A *dominating pair* of G is a pair of vertices $v, u \in V$, such that every induced path between v and u dominates G. A *good pair* of G is a pair of vertices $v, u \in V$, such that there exist two internally disjoint induced dominating paths connecting v and u.

The *claw* is the induced complete bipartite graph $K_{1,3}$, and for simplicity, we refer to it by $K(a;b,c,d)$ (see Figure 1.1). The *net* is the induced six-vertex graph $N(a,b,c;x,y,z)$ shown in Figure 1.1. A graph is called *CN-free* or, equivalently, *(claw, net)-free* if it contains neither an induced claw nor an induced net. An *asteroidal triple* of G is a triple of pairwise nonadjacent vertices, such that for each pair of them there exists a path in G that does not contain any vertex in the neighborhood of the third one. A graph is called *AT-free* if it does not contain an asteroidal triple. Finally, a *Hamiltonian path* or *Hamiltonian cycle* of G is a path or cycle, respectively, containing all vertices of G.

2. Induced dominating path. In this section we give a constructive proof for the property that every connected CN-free graph contains an induced dominating path. In fact, we show that there is an algorithm that finds such a path in linear time. To prove the main theorem of this section we will need the following two lemmas.

LEMMA 2.1 (see [12]). *Let $P = (x_1, \ldots, x_k)$ be an induced path of a CN-free graph G, and let v be a vertex of G such that $dist(v,P) = 2$. Then any neighbor y of v with $dist(y,P) = 1$ is adjacent to x_1 or to x_k.*

LEMMA 2.2. *Let P be an induced path connecting vertices v and u of a CN-free graph G. Let also s be a vertex of G such that $s \notin \mathrm{N}[P]$ and $dist(v,s) \leq dist(v,u)$. Then*

1. *for every shortest path P' connecting v and s, $P' \bigcap P = \{v\}$ holds, and*
2. *if there is an edge xy of G such that $x \in P \setminus \{v\}$ and $y \in P' \setminus \{v\}$, then both x and y are neighbors of v.*

Proof. Let y be the vertex of $P' \setminus \{v\}$ which is closest to s and has a neighbor x on $P \setminus \{v\}$; clearly, $y \neq s$. Let s', v' be the neighbors of y on the subpaths of P' connecting y with s and y with v, respectively. Since $s' \notin \mathrm{N}[P]$, by Lemma 2.1, vertex

y must be adjacent to v or to u. If $yu \in E$, then $v'u \in E$, too (otherwise, we have a claw $K(y; s', v', u)$). But now $dist(v, u) \leq dist(v, v') + 1 = dist(v, y) < dist(v, s) \leq dist(v, u)$, and a contradiction arises. Therefore, y is adjacent to v, and since $y \notin P$, the paths P and P' have only the vertex v in common. Moreover, to avoid a claw $K(y; s', v, x)$, vertex x has to be adjacent to v. □

THEOREM 2.3. *Every connected CN-free graph G has an induced dominating path, and such a path can be found in $O(n + m)$ time.*

Proof. Let G be a connected CN-free graph. One can construct an induced dominating path in G as follows. Take an arbitrary vertex v of G. Using breadth first search (BFS), find a vertex u with the largest distance from v and a shortest path P connecting u with v. Check whether this path P dominates G. If so, we are done. Now, assume that the set $S = V \setminus \mathrm{N}[P]$ is not empty. Again, using BFS, find a vertex s in S with largest distance from v and a shortest path P' connecting v with s. Create a new path P^* by joining P and P' in the following way: $P^* = (P \setminus \{v\}, P' \setminus \{v\})$ if there is a chord xy between the paths P and P' (see Lemma 2.2), and $P^* = (P \setminus \{v\}, P')$, otherwise. By Lemma 2.2, the path P^* is induced. It remains to show that this path dominates G.

Assume there exists a vertex $t \in V \setminus \mathrm{N}[P^*]$. First, we claim that t is dominated neither by P nor by P'. Indeed, if $t \in (\mathrm{N}[P] \bigcup \mathrm{N}[P']) \setminus \mathrm{N}[P^*]$, then necessarily $tv \in E$ and $v \notin P^*$, i.e., neighbors $x \in P$ and $y \in P'$ of v are adjacent. Therefore, we get a net $N(v, y, x; t, s', u')$, where s' and u' are the vertices at distance two from v on paths P' and P, respectively. Note that vertices s', u' exist because $dist(v, s) \geq 2$.

Thus, t is dominated neither by P nor by P'. Moreover, from the choice of u and s we have $2 \leq dist(v, t) \leq dist(v, s) \leq dist(v, u)$. Now let P'' be a shortest path, connecting t with v, and let z be a neighbor of v on this path. Applying Lemma 2.2 twice (to P, P'' and to P', P''), we obtain a subgraph of G depicted in Figure 2.1. We have three shortest paths P, P', P'', each of length at least 2 and with only one common vertex v. These paths can have only chords of type zx, zy, xy. Any combination of them leads to a forbidden claw or net. This contradiction completes the proof of the theorem. Evidently, the method described above can be implemented to run in linear time. □

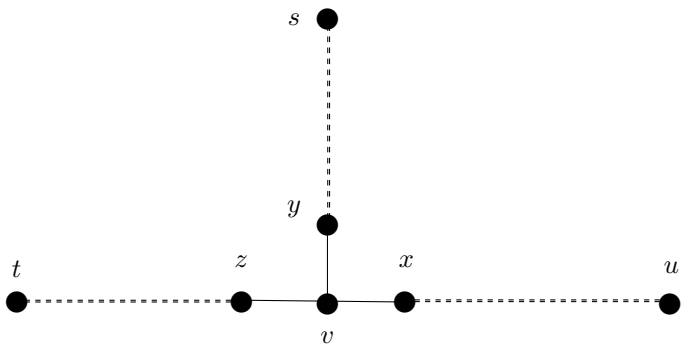


FIG. 2.1.

It is not clear whether CN-free graphs can be recognized efficiently. But, to apply our method for finding an induced dominating path in these graphs, we do not need to know in advance that a given graph G is CN-free. Actually, our method can be

applied to any graph G. It either finds an induced dominating path or returns either a claw or a net of G, showing that G is not CN-free.

COROLLARY 2.4. *There is a linear time algorithm that for a given (arbitrary) connected graph G either finds an induced dominating path or outputs an induced claw or an induced net of G.*

Proof. Let G be a graph. For an arbitrary vertex v of G, we find a vertex u with the largest distance from v and a shortest path P connecting u with v. If P dominates G, then we are done. Else, we find a vertex $s \in V \setminus \mathrm{N}[P]$ with the largest distance from v and a shortest path P' connecting v with s. If there are vertices in $P' \setminus \{v\}$ which have a neighbor on $P \setminus \{v\}$, we take the vertex y that is closest to s and check whether y is adjacent to v and u. If it is adjacent neither to u nor to v, then G has a net or a claw (see the proof of Lemma 2.1). If $yu \in E$ or $yv \in E$ and a neighbor x of y on $P \setminus \{v\}$ is not adjacent to v, then G has a claw (see Lemma 2.2). Now, if we did not yet find a forbidden subgraph, then the only possible chord between the paths P and P' is xy with $xv, yv \in E$, and we can create an induced path P^* as described in the proof of Theorem 2.3. Hence, it remains to check whether P^* dominates G. If there exists a vertex $t \in V \setminus \mathrm{N}[P^*]$, then again we will find a net or a claw in G (see Theorem 2.3). It is easy to see that the total time bound of all these operations is linear. □

3. Hamiltonian path. In what follows we show that for claw-free graphs the existence of an induced dominating path is a sufficient condition for the existence of a Hamiltonian path. The proof for this result is constructive, implying that, given an induced dominating path, one can find a Hamiltonian path efficiently.

THEOREM 3.1. *Every connected claw-free graph G containing an induced dominating path has a Hamiltonian path. Moreover, given an induced dominating path, a Hamiltonian path of G can be constructed in linear time.*

Proof. Let $G = (V, E)$ be a connected claw-free graph and let $P = (x_1, \ldots, x_k)$ ($k \geq 1$) be an induced dominating path of G. If $k = 1$, vertex x_1 dominates G and, since G is claw-free, there are no three independent vertices in $G - \{x_1\}$. (By $G - \{x_1\}$ we denote a subgraph of G induced by vertices $V \setminus \{x_1\}$.) If $G - \{x_1\}$ is not connected, then, again because G is claw-free, it consists of two cliques C_0, C_1 and a Hamiltonian path of G can easily be constructed. If $G - \{x_1\}$ is connected, we can construct a Hamiltonian path as follows. First, we construct a maximal path $P_1 = (y_1, \ldots, y_l)$, i.e., all vertices that are not in P_1 are neither connected to y_1 nor to y_l. Let R be the set of all remaining vertices. If $R = \emptyset$, we are done. If there is any vertex in R, it follows that $y_1 y_l \in E$ since otherwise there are three independent vertices in $G - \{x_1\}$. Furthermore, any two vertices of R are joined by an edge, since otherwise they would form an independent triple with y_1 (and with y_l as well). Hence, R induces a clique. Since $G - \{x_1\}$ is connected, there has to be an edge from a vertex $v_R \in R$ to some vertex $y_i \in P_1$ ($1 < i < l$). Now we can construct a Hamiltonian path P of G: $P = (x_1, y_{i+1}, y_{i+2}, \ldots, y_l, y_1, y_2, \ldots, y_i, v_R, \tilde{R})$, where $\tilde{R}$ stands for an arbitrary permutation of the vertices of $R \setminus \{v_R\}$.

For $k \geq 2$ we first construct a Hamiltonian path P_2 for $G' = G(\mathrm{N}[x_1] \setminus \{x_2\})$ as described above, using x_1 as the dominating vertex. At least one endpoint of P_2 is adjacent to x_2 since if $G' - \{x_1\}$ is not connected, x_2 has to be adjacent to all vertices of either C_0 or C_1 (otherwise, there is a claw in G), and if $G' - \{x_1\}$ is connected, the construction gives a path ending in x_1 which is, of course, adjacent to x_2. To construct a Hamiltonian path for the rest of the graph we define for each vertex x_i ($i \geq 2$) of P a set of vertices $C_i = \mathrm{N}(x_i) \setminus \bigcup_{j=1}^{i-1} \mathrm{N}[x_j]$. Each set C_i forms a clique of G since if two

vertices $u, v \in C_i$ are not adjacent, then the set u, v, x_i, x_{i-1} induces a claw. Hence we can construct a path $P^* = (P_2, x_2, P_2^C, x_3, P_3^C, x_4, \ldots, x_{k-1}, P_{k-1}^C, x_k, P_k^C)$, where P_i^C stands for an arbitrary permutation of the vertices of $C_i \setminus \{x_{i+1}\}$. This path P^* is a Hamiltonian path of G because it obviously is a path, and, since P is a dominating path, each vertex of G has to be either on P, P_2, or in one of the sets C_i.

For the case $k = 1$ both finding the connected components of $G - \{x_1\}$ and constructing the path P_1 can easily be done in linear time. For $k \geq 2$ we just have to make sure that the construction of the sets C_i can be done in $O(n+m)$, and this can be realized easily within the required time bound. □

THEOREM 3.2. *Every connected CN-free graph G has a Hamiltonian path, and such a path can be found in $O(n+m)$ time.*

Proof. By Theorem 2.3, every connected CN-free graph has an induced dominating path P, and it can be found in linear time. Using the path P, by Theorem 3.1, one can construct a Hamiltonian path of G in linear time. □

Analogously to Corollary 2.4, we can state the following.

COROLLARY 3.3. *There is a linear time algorithm that for a given (arbitrary) connected graph G either finds a Hamiltonian path or outputs an induced claw or an induced net of G.*

Proof. The proof follows from Corollary 2.4 and the proof of Theorem 3.1. □

4. Induced dominating cycle, dominating shortest path, or good pair. In this section we show that every 2-connected CN-free graph G has an induced doubly dominating cycle or a good pair. Moreover, we present an efficient algorithm that, for a given 2-connected CN-free graph G, finds either a good pair or an induced doubly dominating cycle.

LEMMA 4.1. *Every hole of a connected CN-free graph G dominates G.*

COROLLARY 4.2. *Let H be a hole of a connected CN-free graph G. Every vertex of $V \setminus H$ is adjacent to at least two vertices of H.*

A *subgraph G' of G (doubly) dominates* G if the vertex set of G' (doubly) dominates G.

LEMMA 4.3. *Every induced subgraph of a connected CN-free graph G which is isomorphic to S_3 or S_3^- (see Figure 4.1) dominates G.*

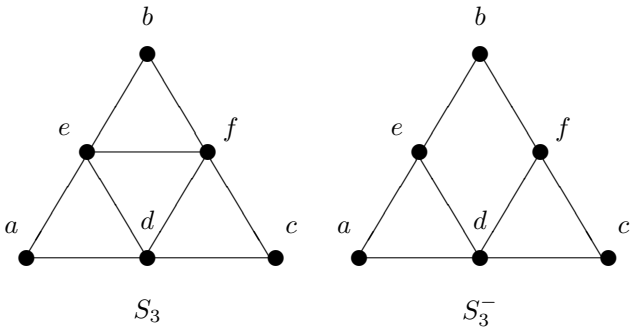


FIG. 4.1.

Proof. Let G contain an induced subgraph isomorphic to S_3^-, and assume that it does not dominate G. Then, there must be a vertex s such that $dist(s, S_3^-) = 2$. Let x be a neighbor of s from $N[S_3^-]$. If x is adjacent neither to a, nor to b, nor to c (see Figure 4.1), then G contains a claw (e.g., if $xf \in E$, then a claw $K(f; b, c, x)$ arises). Thus, without loss of generality, x has to be adjacent to a or b.

If $xa \in E$, then x is adjacent neither to b nor to c, since otherwise we will get a claw ($K(x; a, b, s)$ or $K(x; a, c, s)$). To avoid a net $N(a, e, d; x, b, c)$ vertex x must be adjacent to e or d. But, if $ex \in E$, then $xd \in E$ too. (Otherwise, we will have a claw $K(e; b, d, x)$.) Analogously, if $xd \in E$, then also $xe \in E$. Hence, x is adjacent to both e and d, and a net $N(x, e, d; s, b, c)$ arises.

Now, we may assume that x is adjacent to b and not to a, c. To avoid a claw $K(b; x, e, f)$, x must be adjacent to e or f. But again, $xe \in E$ if and only if $xf \in E$. (Otherwise, we get a net $N(x, b, e; s, f, a)$ or $N(x, b, f; s, e, c)$.) Hence x is adjacent to both e and f and a claw $K(x; s, e, f)$ arises.

Consequently, S_3^- dominates G. Similarly, every induced S_3 (if it exists) dominates G. □

LEMMA 4.4. *Let P be an induced path connecting vertices v and u of a connected CN-free graph G. Let s be a vertex of G such that $s \notin \mathrm{N}[P]$ and $dist(v, s) \leq dist(v, u)$, $dist(u, s) \leq dist(v, u)$. Then G has an induced doubly dominating cycle, and such a cycle can be found in linear time.*

Proof. Let P_v and P_u be shortest paths connecting vertex s with v and u, respectively. Both these paths as well as the path P have lengths at least 2. Since $dist(v, s) \leq dist(v, u)$ and $dist(u, s) \leq dist(u, v)$, by Lemma 2.2, we have $P \bigcap P_v = \{v\}$ and $P \bigcap P_u = \{u\}$. Moreover, if there is a chord between P and P_v, then it is unique and both its endvertices are adjacent to v. The same holds for P and P_u; both endvertices of the chord (if it exists) are adjacent to u.

Now, without loss of generality, we suppose that $dist(s, u) \leq dist(s, v)$. Then, from $u \notin \mathrm{N}[P_v]$ and Lemma 2.2 we deduce that $P_u \bigcap P_v = \{s\}$ and between paths P_v and P_u at most one chord is possible, namely, the one with both endvertices adjacent to s. Consequently, we have constructed an induced subgraph of G shown in Figure 4.2 (only chords $s's'', v'v''$ and $u'u''$ are possible).

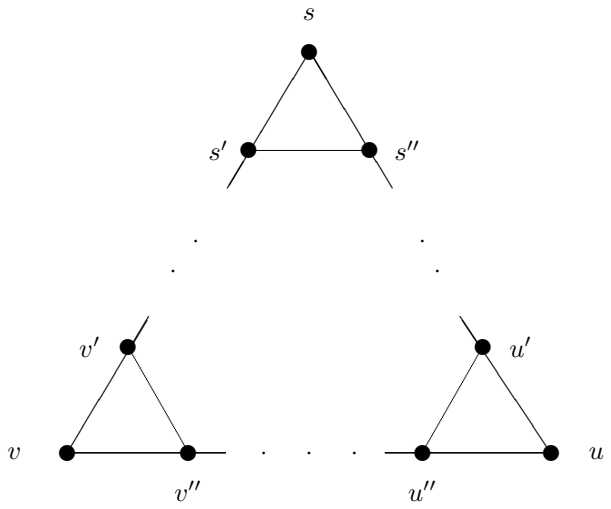


FIG. 4.2.

If the lengths of all three paths P, P_v, P_u are at least 3, then it is easy to see that G has a hole H_k ($k \geq 6$). Furthermore, if at least one of these paths has length greater than or equal to 4, or two of them have lengths 3, then G must contain a hole H_k ($k \geq 5$). It remains to consider two cases: lengths of both P_v and P_u are 2 and

the length of P is 3 or 2. Clearly, in both of these cases the graph G contains either a hole H_k ($k \in \{5, 6, 7\}$) or an induced subgraph isomorphic to S_3^- or S_3. By Corollary 4.2, every hole of G doubly dominates G.

Let G contain an S_3^- with vertex labeling shown in Figure 4.1. We claim that the induced cycle (e, b, f, d, e) dominates G or G contains a hole H_6. Indeed, if a vertex s of G does not belong to S_3^-, then, by Lemma 4.3, it is adjacent to a vertex of S_3^-. Suppose that s is adjacent to none of e, b, f, d. Then, without loss of generality, $sa \in E$, and we obtain an induced subgraph of G isomorphic either to a net $N(e, a, d; b, s, c)$ or to $H_6 = (s, a, e, b, f, c, s)$, depending on whether vertices s and c are adjacent. Hence, we may assume that (e, b, f, d, e) dominates G, and since G is claw-free, this cycle is doubly dominating.

Now let G contain an S_3 with vertex labeling shown in Figure 4.1. We will show that every vertex of G is adjacent to at least two vertices of the cycle (e, f, d, e) or G contains a hole H_5. Suppose vertex s of G is adjacent to none of e, d. Then, by Lemma 4.3, s is adjacent to at least one of a, b, c, f. Let $sf \in E$. To avoid a claw, vertex s is adjacent to both b and c. But then a hole $H_5 = (s, b, e, d, c, s)$ arises. Assume that $sf \notin E$ and, without loss of generality, $sa \in E$. To avoid a net $N(a, e, d; s, b, c)$, s must be adjacent to b or c. In both cases a hole H_5 occurs.

Clearly, the construction of an induced doubly dominating cycle of G given above takes linear time. □

THEOREM 4.5. *There is a linear time algorithm that, for a given connected CN-free graph G, either finds an induced doubly dominating cycle or gives a dominating shortest path of G.*

Proof. Let G be a connected CN-free graph. One can construct an induced doubly dominating cycle or a dominating shortest path of G as follows (compare with the proof of Theorem 2.3). Take an arbitrary vertex v of G. Find a vertex u with the largest distance from v and a shortest path P connecting u with v. Check whether P dominates G. If so, we are done; P is a dominating shortest path of G. Assume now that the set $S = V \setminus \mathrm{N}[P]$ is not empty. Find a vertex s in S with the largest distance from v and a shortest path P_v connecting v with s. Create again a new path P^* by "joining" shortest paths P and P_v as in the proof of Theorem 2.3. We have proven there that P^* dominates G. Now let P_u be a shortest path between s and u. If $dist(s, u) \leq dist(v, u)$ or both $dist(s, u) > dist(v, u)$ and $v \notin \mathrm{N}[P_u]$, then Lemma 4.4 can be applied to get an induced doubly dominating cycle of G in linear time. Therefore, we may assume that $dist(s, u) > dist(v, u) \geq dist(v, s)$ and $v \in \mathrm{N}[P_u]$. Now we show that the shortest path P_u dominates G. If v lies on the path P_u, then $P^* = P_u$ and we are done. Otherwise, let x be a neighbor of v in P_u. Note that $dist(v, s) > 1$ and so $x \neq s, u$. Since G is claw-free, v is adjacent to a neighbor $y \in P_u$ of x. Assume, without loss of generality, that x is closer to s than y. If we show that $dist(v, s) = 1 + dist(x, s)$ and $dist(v, u) = 1 + dist(y, u)$, then again, by the proof of Theorem 2.3, the path P_u will dominate G (as a path obtained by "joining" two shortest paths that connect v with u and v with s, respectively). By the triangle condition, we have $dist(u, s) < dist(v, u) + dist(v, s)$ (strict inequality because $v \notin P_u$) and $dist(v, s) \leq 1 + dist(x, s)$, $dist(u, v) \leq 1 + dist(y, u)$. Consequently, $dist(v, u) + dist(v, s) > dist(u, s) = dist(u, y) + 1 + dist(x, s) \geq dist(v, u) - 1 + 1 + dist(v, s) - 1 = dist(v, u) + dist(v, s) - 1$. That is, $dist(u, s) = dist(v, u) + dist(v, s) - 1$ and $dist(v, s) = 1 + dist(x, s)$, $dist(u, v) = 1 + dist(y, u)$. □

Since all our proofs were constructive, we can conclude the following.

COROLLARY 4.6. *There is a linear time algorithm that, for a given (arbitrary) connected graph G, either finds an induced doubly dominating cycle, or gives a dominating shortest path, or outputs an induced claw or an induced net of G.*

LEMMA 4.7. *Let $P = (v, x_2, \ldots, x_{k-1}, u)$ be a dominating shortest path of a graph G. Then* $\max\{ecc(v), ecc(u)\} \geq diam(G) - 1$.

Proof. Let x, y be a diametral pair of vertices of G; that is, $diam(G) = dist(x, y)$. If both x and y are on P, then necessarily $\{x, y\} = \{v, u\}$ and therefore $dist(v, u) = diam(G) = ecc(v) = ecc(u)$. If $x \in P$ and $y \in N[P] \setminus P$, then either $x \neq v, u$ and $diam(G) = dist(x, y) = dist(v, u)$ holds or, without loss of generality, $v = x$ and $ecc(v) = diam(G)$. Finally, if both x and y are in $N[P] \setminus P$ and $dist(v, u) < dist(x, y)$, then we may assume that at least one of x, y belongs to $N(v)$, say, x. Hence, $dist(x, y) \leq 1 + dist(v, y) \leq 1 + ecc(v)$; that is, $ecc(v) \geq diam(G) - 1$. □

A pair of vertices u, v of G with $dist(u, v) = ecc(u) = ecc(v)$ is called a *pair of mutually furthest vertices.*

COROLLARY 4.8. *For a graph G with a given dominating shortest path, a pair of mutually furthest vertices can be found in linear time.*

Proof. Let $P = (v, x_2, \ldots, x_{k-1}, u)$ be a dominating shortest path of G with $ecc(v) \geq ecc(u)$. Then, by Lemma 4.7, $ecc(v) \geq diam(G) - 1$ holds. Denote by x a vertex of G such that $dist(v, x) = ecc(v)$. Note that both the eccentricity of v and a vertex furthest from v can be found in linear time by BFS. Now, if $ecc(x) = ecc(v)$, then v, x are mutually furthest vertices of G. Else, $ecc(x) > ecc(v) \geq diam(G) - 1$ must hold and vertices x and y, where y is a vertex with $dist(x, y) = ecc(x)$, form a diametral pair of G; $dist(x, y) = ecc(x) = ecc(y) = diam(G)$. □

In what follows we will use the fact that in a 2-connected graph every pair of vertices is joined by two internally disjoint paths. In order to actually find such a pair of paths, one can use Tarjan's linear time depth first search- (DFS)-algorithm for finding the blocks of a given graph. For the proof of Lemma 4.9, we refer to [21].

LEMMA 4.9. *Let G be a 2-connected graph, and let x, y be two different nonadjacent vertices of G. Then one can construct in linear time two induced, internally disjoint paths, both joining x and y.*

THEOREM 4.10. *There is a linear time algorithm that, for a given 2-connected CN-free graph G, either finds an induced doubly dominating cycle or gives a good pair of G.*

Proof. By Theorem 4.5, we get either an induced doubly dominating cycle or a dominating shortest path of G in linear time. We show that, having a dominating shortest path of a 2-connected graph G, one can find in linear time a good pair or an induced doubly dominating cycle. By Corollary 4.8, we may assume that a pair x, y of mutually furthest vertices of G is given. Let also P_1, P_2 be two induced internally disjoint paths connecting x and y in G. They exist and can be found in linear time by Lemma 4.9 (clearly, we may assume that $xy \notin E$, because otherwise $N[x] = V = N[y]$ and x, y together with a vertex $z \in V \setminus \{x, y\}$ will form a doubly dominating triangle). If one of these paths, say, P_1, is not dominating, then there must be a vertex $s \in V \setminus N[P_1]$. Since x, y are mutually furthest vertices of G, we have $dist(s, x) \leq dist(x, y)$, $dist(s, y) \leq dist(x, y)$. Hence, we are in the conditions of Lemma 4.4 and can find an induced doubly dominating cycle of G in linear time. □

COROLLARY 4.11. *There is a linear time algorithm that, for a given (arbitrary) 2-connected graph G, either finds an induced doubly dominating cycle, or gives a good pair, or outputs an induced claw or an induced net of G.*

5. Hamiltonian cycle. In this section we prove that, for claw-free graphs, the existence of an induced doubly dominating cycle or a good pair is sufficient for the existence of a Hamiltonian cycle. The proofs are also constructive and imply linear time algorithms for finding a Hamiltonian cycle.

THEOREM 5.1. *Every claw-free graph G that contains an induced doubly dominating cycle has a Hamiltonian cycle. Moreover, given an induced doubly dominating cycle, a Hamiltonian cycle of G can be constructed in linear time.*

Proof. Let $DC = (x_1, \ldots, x_k, x_1)$ $(k \geq 3)$ be an induced doubly dominating cycle of G. As before, we define $C_i = \mathrm{N}(x_i) \setminus \bigcup_{j=1}^{i-1} \mathrm{N}[x_j]$ $(2 \leq i \leq k)$. Each set C_i forms a clique of G; otherwise, we would have a claw. Furthermore, $C_k = \emptyset$ holds, and the sets $\mathrm{N}[x_1], C_2, \ldots, C_{k-1}$ form a partition of the vertex set of G. Note that any vertex adjacent to x_k and not to x_j $(1 < j < k)$ belongs to $\mathrm{N}[x_1]$, since the cycle DC is doubly dominating. Let $G' = G(\mathrm{N}[x_1] \setminus \{x_2, x_k\})$ be the subgraph of G induced by $\mathrm{N}[x_1] \setminus \{x_2, x_k\}$. If we show that there is a Hamiltonian path P in G' starting at a neighbor of x_k and ending at a neighbor of x_2, then we are done; the cycle $(x_k, P, x_2, P_2^C, x_3, P_3^C, x_4, \ldots, x_{k-1}, P_{k-1}^C, x_k)$, is a Hamiltonian cycle of G (recall that P_i^C stands for an arbitrary permutation of the vertices of $C_i \setminus \{x_{i+1}\}$).

Since G' is a connected graph, by Theorem 3.1, there exists a Hamiltonian path $P' = (s, y_1, \ldots, y_l, t)$ of G'. Assume that $x_k s, x_k t \notin E$. Then, to avoid a claw $K(x_1; x_k, s, t)$, vertices s and t have to be adjacent, giving a new Hamiltonian path P'' of G' starting at x_1 and ending at a vertex y. If y is adjacent neither to x_k nor to x_2, then a claw $K(x_1; x_k, x_2, y)$ occurs. (Note that in case $k = 3$, i.e., $x_k x_2 \in E$, y is adjacent to at least one of x_k, x_2 because the cycle $DC = (x_1, x_2, x_3, x_1)$ is doubly dominating.) Without loss of generality, $y x_2 \in E$ and the path P'' is a desired path of G'.

So, we may assume that x_k is adjacent to t or s. Analogously, x_2 is adjacent to one of t, s. If x_k, x_2 are adjacent to different vertices, then we are done; the path P' starts at a neighbor of x_k and ends at a neighbor of x_2. Otherwise, let both x_k and x_2 be adjacent to t and not to s. Then a claw $K(x_1; x_k, x_2, s)$ arises when $k > 3$, or we get a contradiction with the property of $DC = (x_1, x_2, x_3, x_1)$ to be a doubly dominating cycle. □

COROLLARY 5.2. *Every claw-free graph, containing an induced dominating cycle of length at least 4, has a Hamiltonian cycle, and, given that induced dominating cycle, one can construct a Hamiltonian cycle in linear time.*

Let $G = (V, E)$ be a graph, and let $P = (x_1, \ldots, x_k)$ be an induced dominating path of G. P is called an *enlargeable* path if there is some vertex v in $V \setminus P$ that is either adjacent to x_1 or to x_k but not to both of them and, additionally, to no other vertex in P. Consequently, an induced dominating path P is called *nonenlargeable* if such a vertex does not exist. Obviously, every graph G that has an induced dominating path has a nonenlargeable induced dominating path as well. Furthermore, given an induced dominating path P, one can find in linear time a nonenlargeable induced dominating path P' by simply scanning the neighborhood of both x_1 and x_k. For the next theorem we will need an auxiliary result.

LEMMA 5.3. *Let G be a claw-free graph, and let $P = (x_1, x_2, \ldots, x_k)$ $(k > 2)$ be an induced nonenlargeable dominating path of G such that there is no vertex y in G with $\mathrm{N}(y) \cap P = \{x_1, x_k\}$. Then there is a Hamiltonian path in G that starts in x_1 and ends in x_k and, given the path P, one can construct this Hamiltonian path in linear time.*

Proof. Let $C_i = \mathrm{N}(x_i) \setminus \bigcup_{j=1}^{i-1} \mathrm{N}[x_j]$ $(i \geq 2)$. Since P is nonenlargeable, C_k is empty. Using the method described in the proof of Theorem 3.1, we can easily construct a path, starting in x_1 and ending in x_k, that contains all vertices of $C_2, \ldots, C_{k-1}$. This implies that we have to worry only about how to insert the vertices of the neighborhood of x_1 into this path. We have to consider two cases.

Case 1. $H = G(\mathrm{N}(x_1) \setminus \{x_2\})$ consists of two connected components C_0, C_1.

Since G is claw-free, both C_0 and C_1 induce cliques in G. Furthermore, x_2 is adjacent to all vertices of at least one of C_0 and C_1, say, C_1, because otherwise we have a claw in G.

Let y be an arbitrary vertex of C_0. Since P is nonenlargeable, y has at least one neighbor on $P \setminus \{x_1\}$, and let x_j be the one with smallest index. By the preconditions of our lemma, $j \neq k$. If $j > 2$, then y has to be adjacent to x_{j+1} as well, since otherwise $K(x_j; y, x_{j-1}, x_{j+1})$ is a claw. Furthermore, y is adjacent to all vertices $c_j \in C_j$, since otherwise $K(x_j; y, x_{j-1}, c_j)$ is a claw. Hence, when constructing the Hamiltonian path, we can simply add y to C_j.

Now we consider the set Y of all vertices y of C_0 with $yx_2 \in E$. Suppose there is a vertex c_2 in C_2 with $c_2 \neq x_3$. If there is a vertex $c_1 \in C_1$ that is nonadjacent to vertex c_2, then there is an edge from every vertex $c_0 \in Y$ to c_2; otherwise, $K(x_2; c_0, c_1, c_2)$ is a claw of G. This implies that we can construct a Hamiltonian path with the required properties. If, on the other hand, all vertices of C_1 are adjacent to all vertices of C_2, we can construct such a path by starting in x_1, traversing through Y, x_2, C_1, C_2, and proceeding as before. Now suppose that there is no vertex $c_2 \in C_2$ with $c_2 \neq x_3$. In this case either all vertices $c_0 \in Y$ or all vertices $c_1 \in C_1$ have to be adjacent to x_3, because otherwise $K(x_2; c_0, c_1, x_3)$ is a claw. Suppose, without loss of generality, that all vertices of Y are adjacent to x_3. Then we construct the path by starting in x_1, traversing through C_1, x_2, Y, x_3, and proceeding as before.

Case 2. $H = G(\mathrm{N}(x_1) \setminus \{x_2\})$ induces a connected graph.

If x_2 is not adjacent to any of the vertices in H, then H has to be a clique and we can apply the method described in case 1.

Suppose now that x_2 is adjacent to some vertex in H. First, we construct a Hamiltonian path $P' = (y_1, \ldots, y_l)$ in H, which is done as in the proof of Theorem 3.1, since there is no independent triple in H. Now we claim that either x_2 is adjacent to one of y_1 or y_l, or P' does in fact induce a Hamiltonian cycle of H implying again the existence of a path with an end-vertex adjacent to x_2. Indeed, suppose x_2 is not adjacent to any of the endvertices of P'. Then, since G is claw-free, y_1 has to be adjacent to y_l, because otherwise $K(x_1; y_1, y_l, x_2)$ would induce a claw in G. Hence P' induces a Hamiltonian cycle in H.

Using P', we can easily construct a Hamiltonian path in $\mathrm{N}[x_1]$ starting in x_1 and ending in x_2. The rest of the Hamiltonian path of G can be constructed as before. □

In fact, we can prove a slightly stronger result. Let $A = \mathrm{N}(x_1) \setminus \bigcup_{j=2}^{k} \mathrm{N}[x_j]$, $B = \mathrm{N}(x_k) \setminus \bigcup_{j=1}^{k-1} \mathrm{N}[x_j]$, and, as usual, $C_i = \mathrm{N}(x_i) \setminus \bigcup_{j=1}^{i-1} \mathrm{N}[x_j]$ $(i \geq 2)$. Each of these sets forms a clique of G.

LEMMA 5.4. *Let G be a claw-free graph, and let $P = (x_1, x_2, \ldots, x_k)$ $(k > 2)$ be an induced dominating path of G such that there is no vertex y in G with $\mathrm{N}(y) \cap P = \{x_1, x_k\}$. Let also P be enlargeable but only to one end, e.g., $A = \emptyset$, $B \neq \emptyset$, and assume that there exists an edge zb with $z \in C_{k-1} \setminus \{x_k\}$ and $b \in B$. Then there is a Hamiltonian path in G that starts in x_1 and ends in x_k and, given the path P, one can construct this Hamiltonian path in linear time.*

Proof. First, we can easily construct a path, starting in x_1 and ending in x_{k-1}, that contains all vertices of $C_2, \ldots, C_{k-2}$. Then we attach to this path a path which starts at x_{k-1}, goes through C_{k-1}, B using all their vertices, and ends in x_k. Finally, we insert the vertices of the neighborhood of x_1 into the obtained path as we have done in the proof of Lemma 5.3. □

THEOREM 5.5. *Let G be a 2-connected claw-free graph with a good pair u, v. Then G has a Hamiltonian cycle and, given the corresponding induced dominating paths, one can construct a Hamiltonian cycle in linear time.*

Proof. Let $P_1 = (u = x_1, \ldots, v = x_k)$, $P_2 = (u = y_1, \ldots, v = y_l)$ be the induced dominating paths, corresponding to the good pair u, v. By the definition of a good pair both k and l are greater than 2. We may also assume that, for any induced dominating path $P = (a_1, \ldots, a_s)$ of G with $s > 2$, no vertex $y \in V \setminus P$ exists such that $\mathrm{N}(y) \bigcap P = \{a_1, a_s\}$. Otherwise, P together with y would form an induced dominating cycle of length at least 4, and we can apply Corollary 5.2 to construct a Hamiltonian cycle of G in linear time.

Let A_1 be the set of vertices a_1 that are adjacent to x_1 but to no other vertex of P_1; let B_1 be the set of vertices b_1 that are adjacent to x_k but to no other vertex of P_1. A_2 and B_2 are defined accordingly for P_2. Of course, each of the sets A_1, A_2, B_1, B_2 forms a clique of G.

First we assume that one of these paths, say, P_1, is nonenlargeable, i.e., $A_1 = \emptyset$, $B_1 = \emptyset$. In this case we do the following. We remove the inner vertices of P_2 from G and get the graph $G - (P_2)$, where (P_2) denotes the inner vertices of P_2. Then, using P_1, we create a Hamiltonian path in $G - (P_2)$ that starts at u and ends at v (Lemma 5.3), and we add (P_2) to this path to create a Hamiltonian cycle of G.

We can use this method for creating a Hamiltonian cycle of G whenever we have two internally disjoint paths P, P' of G both connecting u with v such that one of them is an induced dominating and nonenlargeable path of the graph obtained from G by removing the inner vertices of the other path.

Now we suppose that both paths P_1, P_2 are enlargeable. Because of symmetry we have to consider the following three cases.

Case 1. There exist a vertex $a_1 \in A_1 \setminus A_2$ and a vertex $b_1 \in B_1 \setminus B_2$.

In this case there must be edges from a_1, b_1 to inner vertices y_i, y_j of P_2. Consequently, we can form a new path P_2' by starting in u and traversing through A_1, $y_i, \ldots, y_j$, B_1, v, where $(y_i, \ldots, y_j)$ is the subpath of P_2 between y_i and y_j. Evidently, P_2' contains all vertices of B_1, A_1 and is internally disjoint from P_1, which is nonenlargeable in $G - (P_2')$.

Case 2. $B_1 = B_2$ and either $A_1 = A_2$ or there exists a vertex $a_1 \in A_1 \setminus A_2$.

In this case none of the vertices of $B := B_1 = B_2$ (if $B \neq \emptyset$) has a neighbor in $P_1 \bigcup P_2$ other than v. As G is 2-connected, for some vertex $b \in B$ there has to be a vertex $z \in V \setminus (P_1 \bigcup P_2 \bigcup B)$ with $zb \in E$. Since P_2 dominates G and $z \notin B$, vertex z must be adjacent to a vertex $y \in P_2 \setminus \{v\}$. If z is only adjacent to $y_1 = u$ but to no other vertex of P_2, then z necessarily belongs to A_2 and we can form a new path P_1' by starting in u, using all vertices of A_2, B and ending in v. Again, P_1' is internally disjoint from P_2 and P_2 is nonenlargeable in $G - (P_1')$. If $\mathrm{N}(z) \bigcap P_2 = \{u, v\}$, then we can apply Corollary 5.2.

Therefore, we may assume that z is adjacent to an inner vertex y of P_2. Now, if there exists a vertex $a_1 \in A_1 \setminus A_2$, then a_1 is adjacent to some vertex y' of (P_2) and we can construct a new path P_2' by using u, A_1, y', $\ldots$, y, z, B, v. (If B was empty, then P_2' ends at $\ldots, y', \ldots, y_{l-1}, v$.) This path is internally disjoint from P_1, which

is nonenlargeable in $G-(P_2')$. If $A_1 = A_2$, then from the discussion above we may assume that either $A := A_1 = A_2$ is empty or there is a vertex $z' \in V \setminus (P_1 \bigcup P_2 \bigcup A)$ which is adjacent to a vertex of A and has a neighbor y' in (P_2). Hence, we can construct a path P_2' by using u, A, z', y', $\ldots$, y, z, B, v, which is internally disjoint from P_1. (If $z' = z$, then P_2' is constructed by using u, A, z, B, v.)

Case 3. A_2 is strictly contained in A_1, and B_1 is strictly contained in B_2.

Consider vertices $b \in B_1$, $z \in B_2 \setminus B_1$, and $z' \in A_1 \setminus A_2$ and cliques $C_i = \mathrm{N}(x_i) \setminus \bigcup_{j=1}^{i-1} \mathrm{N}[x_j]$ $(i \geq 2)$. If $zz' \in E$, then we can construct a new path P_2' by using u, A_1, z, B_1, v. This path is internally disjoint from P_1, which is nonenlargeable in $G-(P_2')$.

Since $z' \notin A_2$, there must be a neighbor $y' \in (P_2)$ of z'. If vertex b is adjacent to some vertex in $C_{k-1} \setminus \{v\}$, then we construct a new path P_2' by using $u, A_1, y', \ldots, v$. It will be internally disjoint from P_1, which is enlargeable only to one end (at $x_k = v$) in $G-(P_2')$. We are now in the conditions of Lemma 5.4 and can construct a Hamiltonian path of $G-(P_2')$ that starts in u and ends in v. Adding (P_2') to this path, we obtain a Hamiltonian cycle of G.

So, we may assume that $zz' \notin E$ for any vertex $z' \in A_1 \setminus A_2$ and that vertex b is not adjacent to any vertex of $C_{k-1} \setminus \{v\}$. From this we conclude also that $z \notin C_{k-1}$. But since $z \notin B_1$, there must be a neighbor $x_j \in (P_1)$ of z. We choose vertex $x_j \in (P_1)$ with the smallest j. Clearly, $1 < j < k-1$ and $z \in C_j$.

First we define a new induced path $P_1' := (P_1 \setminus \{x_{j+1}, \ldots x_{k-1}\}) \cup \{z\}$ and cliques $A_1' := \mathrm{N}(u) \setminus \bigcup_{x \in P_1' \setminus \{u\}} \mathrm{N}[x]$, $B_1' := \mathrm{N}(v) \setminus \bigcup_{x \in P_1' \setminus \{v\}} \mathrm{N}[x]$. We have $z' \in A_1'$, since otherwise from the construction of P_1', z' would be adjacent to z, and that is impossible.

Note that vertex x_{j+1} is dominated by the path P_2. If it is adjacent to only vertex v from P_2, then $j+1 = k-1$ and a claw $K(v; x_{k-1}, y_{l-1}, b)$ arises. Therefore, x_{j+1} must be adjacent to an inner vertex y of P_2. Now we define a new path P_2' by using $u, A_1', y', \ldots, y, x_{j+1}, C_{j+1}, x_{j+2}, \ldots, C_{k-1}, v$. It is internally disjoint from P_1' and contains all vertices of A_1' and C_i $(j+1 \leq i \leq k-1)$. It is clear from the construction that the path P_1' dominates the graph $G-(P_2')$. (Every vertex which was not dominated by the path P_1' in G belongs to some set C_i $(j+1 \leq i \leq k-2)$.)

It remains to show that the path P_1' is nonenlargeable in $G-(P_2')$. Assume by way of contradiction that it is enlargeable. Since $A_1' \subset (P_2')$, this is possible only if $B_1' \neq \emptyset$. Let p be a vertex of B_1'. Then p does not belong to B_1, since otherwise it should be adjacent to z, which is contained in (P_1'). (Recall that B_1, B_2 are cliques, $B_1 \subset B_2$, and $z \in B_2 \setminus B_1$.) Now, from $p \in B_1' \setminus B_1$ we conclude that the neighbors of p in $P_1 \setminus \{v\}$ are only vertices from $\{x_{j+1}, \ldots, x_{k-1}\}$, i.e., p belongs to a set C_s for some $s \geq j+1$. Consequently, a contradiction to $C_s \subset (P_2')$ arises.

It is not hard to see that the above method can be implemented to run in linear time. □

THEOREM 5.6. *Every 2-connected claw-free graph G that contains a dominating pair has a Hamiltonian cycle, and, given a dominating pair, a Hamiltonian cycle can be constructed in linear time.*

Proof. Let v, u be a dominating pair of a 2-connected graph G. If $vu \notin E$, then by Lemma 4.9, there exist two internally disjoint induced paths connecting v and u. Both these paths dominate G, and, therefore, u, v is a good pair of G. Thus, the statement holds by Theorem 5.5.

Now let $vu \in E$. Define sets $A := \mathrm{N}(u) \setminus \mathrm{N}[v]$, $B := \mathrm{N}(v) \setminus \mathrm{N}[u]$, and $S := \mathrm{N}(v) \bigcap \mathrm{N}(u)$. Since G is claw-free, the sets A and B are cliques of G. Notice also that

sets A, B, S, and $\{v, u\}$ form a partition of the vertex set of G.

If there is an edge ab in G such that $a \in A$ and $b \in B$, then vertices a, u, v, b induce a 4-cycle which dominates G. Hence, we can apply Corollary 5.2 to get a Hamiltonian cycle of G. Therefore, assume that no such edge exists. But since G is 2-connected, there must be edges ax, by with $x, y \in S$, $a \in A$, and $b \in B$. We distinguish between two cases. Let G_S denote the subgraph of G induced by S.

Case 1. G_S is disconnected.

Then, it consists of two cliques S_1 and S_2. Now, if vertices x, y are in different components of G_S, say, $x \in S_1$ and $y \in S_2$, then $(u, P^{A\setminus\{a\}}, a, x, P^{S_1\setminus\{x\}}, v, P^{B\setminus\{b\}}, b, y, P^{S_2\setminus\{y\}}, u)$ is a Hamiltonian cycle of G. (P^M stands for an arbitrary permutation of the vertices of a set M.) If x, y are in one component, say, S_1, then $(u, P^{A\setminus\{a\}}, a, x, P^{S_1\setminus\{x,y\}}, y, b, P^{B\setminus\{b\}}, v, P^{S_2}, u)$ is a Hamiltonian cycle of G.

Case 2. G_S is connected.

Then, by Theorem 3.1, there exists a Hamiltonian path $P = (s, y_1, \ldots, y_l, t)$ of G_S. Assume that $as, at \notin E$. Then, to avoid a claw $K(u; a, s, t)$, vertices s and t have to be adjacent, giving a Hamiltonian cycle $HC := (s, y_1, \ldots, y_l, t, s)$ of G_S. Vertices x and y split this cycle into two paths $P_1 = (x, \ldots, y)$ and $P_2 = HC \setminus P_1$. Hence, a cycle $(u, P^{A\setminus\{a\}}, a, P_1, b, P^{B\setminus\{b\}}, v, P_2, u)$ is a Hamiltonian cycle of G.

Now, we may assume that a is adjacent to s or t. Analogously, b is adjacent to one of t, s. If a, b are adjacent to different vertices, say, $as, bt \in E$, then $(u, P^{A\setminus\{a\}}, a, P, b, P^{B\setminus\{b\}}, v, u)$ is a Hamiltonian cycle of G. Finally, if a, b are adjacent only to s (similarly, to t), then $(u, P \setminus \{s\}, v, P^{B\setminus\{b\}}, b, s, a, P^{A\setminus\{a\}}, u)$ is a Hamiltonian cycle of G. □

THEOREM 5.7. *Every* 2*-connected CN-free graph* G *has a Hamiltonian cycle, and such a cycle can be found in* $O(n + m)$ *time.*

Proof. The proof follows from Theorems 4.10, 5.1, and 5.5. □

COROLLARY 5.8. *There is a linear time algorithm that for a given (arbitrary)* 2*-connected graph* G *either finds a Hamiltonian cycle or outputs an induced claw or an induced net of* G.

COROLLARY 5.9. *A Hamiltonian cycle of a* 2*-connected (claw,AT)-free graph can be found in* $O(n + m)$ *time.*

Remark. Corollary 5.8 implies that every 2-connected unit interval graph has a Hamiltonian cycle, which is, of course, well known (see [24, 20]). The interesting difference of the above algorithm compared to the existing algorithms for this problem on unit interval graphs is that it does not require the creation of an interval model. It also follows from Corollaries 3.3 and 5.8 that both the Hamiltonian path problem and the Hamiltonian cycle problem are linear time solvable on proper circular arc graphs. Note that previously known algorithms for these problems had time bounds $O(m + nlogn)$ [18].

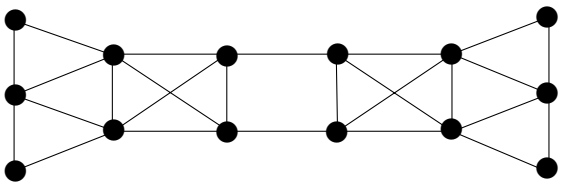


FIG. 5.1. *Claw-free graph, containing a dominating pair and a net.*

It should also be mentioned that Theorems 3.1 and 5.5 do cover a class of graphs that is not contained in the class of CN-free graphs. Figure 5.1 shows a graph that is claw-free, does contain a dominating/good pair and, consequently, a dominating path, but, obviously, it is neither AT-free nor net-free.

REFERENCES

[1] A.S. Asratian, *Every 3-connected, locally connected, claw-free graph is Hamilton-connected*, J. Graph Theory, 23 (1996), pp. 191–201.

[2] B. Bollobás, O. Riordan, Z. Ryjáček, A. Saito, and R.H. Schelp, *Closure and Hamiltonian-connectivity of claw-free graphs*, Discrete Math., 195 (1999), pp. 67–80.

[3] S. Brandt, O. Favaron, and Z. Ryjáček, *Closure and stable Hamiltonian properties in claw-free graphs*, J. Graph Theory, 34 (2000), pp. 30–41.

[4] A. Brandstädt, F.F. Dragan, and E. Köhler, *Linear time algorithms for Hamiltonian problems on (claw,net)-free graphs*, in Proceedings of the 25th International Workshop on Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 1665, Springer-Verlag, New York, 1999, pp. 364–376.

[5] J. Brousek, *Minimal 2-connected non-hamiltonian claw-free graphs*, Discrete Math., 191 (1998), pp. 57–64.

[6] J. Brousek, Z. Ryjáček, and O. Favaron, *Forbidden subgraphs, Hamiltonicity and closure in claw–free graphs*, Discrete Math., 196 (1999), pp. 29–50.

[7] D.G. Corneil, S. Olariu, and L. Stewart, *A linear time algorithm to compute a dominating path in an AT-free graph*, Inform. Process. Lett., 54 (1995), pp. 253–257.

[8] D.G. Corneil, S. Olariu, and L. Stewart, *Asteroidal triple-free graphs*, SIAM J. Discrete Math., 10 (1997), pp. 399–430.

[9] D.G. Corneil, S. Olariu, and L. Stewart, *Linear time algorithms for dominating pairs in asteroidal triple-free graphs*, SIAM J. Comput., 28 (1999), pp. 1284–1297.

[10] P. Damaschke, *Hamiltonian-Hereditary Graphs*, manuscript, 1990.

[11] R. Diestel, *Graph Theory*, Grad. Texts in Math. 173, Springer-Verlag, New York, 1997.

[12] D. Duffus, M.S. Jacobson, and R.J. Gould, *Forbidden subgraphs and the Hamiltonian theme*, in Proceedings of the 4th International Conference on the Theory and Applications of Graphs, Kalamazoo, MI, 1980, Wiley, New York, 1981, pp. 297–316.

[13] R.J. Faudree, E. Flandrin, and Z. Ryjáček, *Claw-free graphs—a survey*, Discrete Math., 164 (1997), pp. 87–147.

[14] R.J. Faudree and R.J. Gould, *Characterizing forbidden pairs for hamiltonian properties*, Discrete Math., 173 (1997), pp. 45–60.

[15] R.J. Faudree, Z. Ryjáček, and I. Schiermeyer, *Forbidden subgraphs and cycle extendability*, J. Combin. Math. Combin. Comput., 19 (1995), pp. 109–128.

[16] E. Flandrin, J.L. Fouquet, and H. Li, *On Hamiltonian claw-free graphs*, Discrete Math., 111 (1993), pp. 221–229.

[17] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[18] P. Hell, J. Bang-Jensen, and J. Huang, *Local tournaments and proper circular arc graphs*, in Algorithms, Lecture Notes in Comput. Sci. 450, T. Asano, T. Ibaraki, H. Imai, and T. Nishizeki, eds., Springer-Verlag, New York, 1990, pp. 101–108.

[19] S. Ishizuka, *Closure, Path-Factors and Path Coverings in Claw-Free Graphs*, manuscript.

[20] J.M. Keil, *Finding hamiltonian circuits in interval graphs*, Inform. Process. Lett., 20 (1985), pp. 201–206.

[21] E. Köhler, *Linear Time Algorithms for Hamiltonian Problems in Claw-Free AT-Free Graphs*, manuscript, 1999.

[22] E. Köhler and M. Kriesell, *Edge-Dominating Trails in AT-free Graphs*, Tech. report 615, Technical University Berlin, Berlin, Germany, 1998.

[23] M. Li, *Hamiltonian cycles in 3-connected claw-free graphs*, J. Graph Theory, 17 (1993), pp. 303–313.

[24] G.K. Manacher, T.A. Mankus, and C.J. Smith, *An optimum $\Theta(n \log n)$ algorithm for finding a canonical hamiltonian path and a canonical hamiltonian circuit in a set of intervals*, Inform. Process. Lett., 35 (1990), pp. 205–211.

[25] Z. Ryjáček, *On a closure concept in claw-free graphs*, J. Combin. Theory Ser. B, 70 (1997), pp. 217–224.

[26] F.B. Shepherd, *Hamiltonicity in claw-free graphs*, J. Combin. Theory Ser. B, 53 (1991), pp. 173–194.
[27] F.B. Shepherd, *Claws*, Master's thesis, University of Waterloo, Waterloo, Ontario, Canada, 1987.
[28] D. B. West, *Introduction to Graph Theory*, Prentice–Hall, Upper Saddle River, NJ, 1996, chapter 6, problem 6.3.14.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1678–1700

SQUARISH k-d TREES*

LUC DEVROYE†, JEAN JABBOUR†, AND CARLOS ZAMORA-CURA†

Abstract. We modify the k-d tree on $[0,1]^d$ by always cutting the longest edge instead of rotating through the coordinates. This modification makes the expected time behavior of lower-dimensional partial match queries behave as perfectly balanced complete k-d trees on n nodes. This is in contrast to a result of Flajolet and Puech [*J. Assoc. Comput. Mach.*, 33 (1986), pp. 371–407], who proved that for (standard) random k-d trees with cuts that rotate among the coordinate axes, the expected time behavior is much worse than for balanced complete k-d trees. We also provide results for range searching and nearest neighbor search for our trees.

Key words. k-d trees, partial match query, range search, expected time, probabilistic analysis of algorithms, data structures

AMS subject classifications. 68P05, 68Q25, 60C05

PII. S0097539799358926

1. Introduction.

The k-d tree, or k-dimensional binary search tree, was proposed by Bentley (1975). In this paper, we propose a modification, the squarish k-d tree, and analyze its expected time performance for partial match queries, orthogonal range searching, and nearest neighbor search under the standard random model for the input (n points independently and uniformly distributed on the unit hypercube). We point out its superiority over the standard k-d tree for this model.

Bentley's k-d tree is a binary search tree that generalizes the 1-d tree or ordinary binary search tree to $\mathbb{R}^k$. A partition of space into hyperrectangles is obtained by splitting alternating coordinate axes by hyperplanes through data points. Figure 1 shows the partition and the corresponding k-d tree. Insertion and search are implemented as for the standard binary search tree algorithms. These trees are used for a variety of other operations, including orthogonal range searching (report all points within a given rectangle), partial match queries (report all points whose values match a given k-dimensional vector with possibly a number of wild cards, e.g., we may search for all points with values $(a_1, *, *, a_4, a_5, *)$, where $*$ denotes a wild card). Additionally, nearest neighbor searching is greatly facilitated by k-d trees. For orthogonal range searching, a host of particular data structures have been developed, such as the range tree and variations or improvements of it (for surveys, see Bentley and Friedman (1979); Bentley (1979); Yao (1990); Samet (1990a), (1990b); and Agarwal (1997)). However, the k-d tree offers several advantages: it takes $O(kn)$ space for n data points, it is easily updated and maintained, it is simple to implement and comprehend, and it is useful for other operations besides orthogonal range search.

Bentley's orthogonal range search algorithm simply visits recursively all subtrees of the root that have a nonempty intersection with the query rectangle. In Figure 1, for example, the left and right subtrees of the root are visited. Note that each node in the tree represents both a point of the data and a rectangle in the partition, namely,

*Received by the editors July 20, 1999; accepted for publication (in revised form) April 26, 2000; published electronically November 28, 2000. This paper was sponsored by NSERC grant A3456 and by FCAR grant 90-ER-0291.

http://www.siam.org/journals/sicomp/30-5/35892.html

†School of Computer Science, McGill University, 3480 University Street, Montreal, Canada H3A 2K6 (luc@cs.mcgill.ca, jabbour@cs.mcgill.ca, czamora@cs.mcgill.ca). The second author was on leave from the Department of Mathematics, Université de Versailles, France. The third author received a DGAPA-UNAM Scholarship.

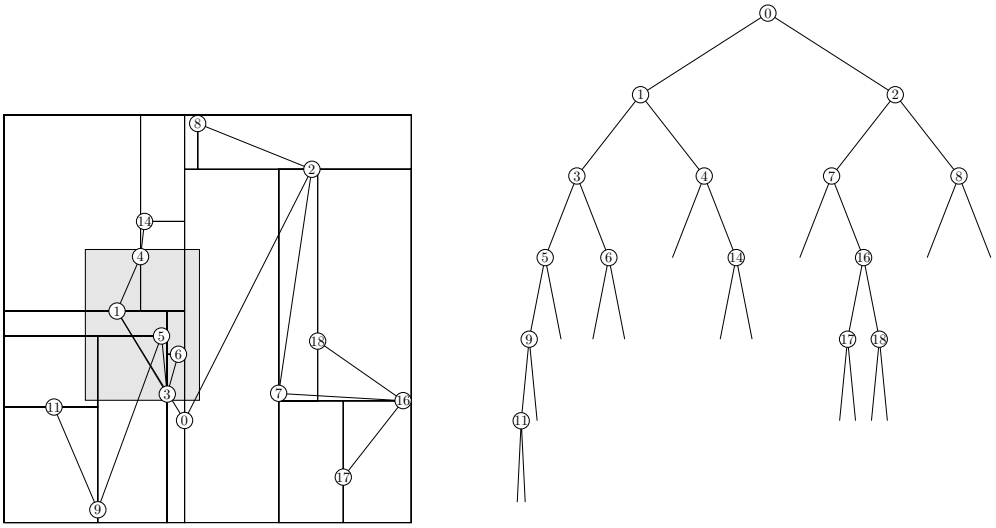


FIG. 1. *The query rectangle is shaded.*

the rectangle split by that point. Leaf regions thus have no points strictly in their interior. The query time for orthogonal search depends upon many factors, such as the location of the query rectangle and the distribution of the points. One may construct a median k-d tree off-line by splitting each time about the median, thus obtaining a perfectly balanced binary tree, in which ordinary point search takes $\Theta(\log n)$ worst-case time, and a partial match query with s coordinates specified takes worst-case time $O(n^{1-s/k}+N)$, where N is the number of points returned (see, for example, Lee and Wong (1977)).

For on-line insertion, balancing is notoriously difficult. If we assume that the data are independent and have a common distribution, then the expected query time is clearly of interest. For standard random binary search trees, it is known (Knuth (1997); Pittel (1984); Devroye (1986), (1987); Mahmoud (1992)) that most properties of balanced search trees are inherited: the expected depth of a randomly selected node is about $2\log n$ and the expected height is $O(\log n)$. One would hope that the random k-d tree, constructed by consecutive insertion of n data points, would also have a performance close to that of the median (off-line) k-d tree. Assuming that the data points are drawn from the uniform distribution on the unit k-cube, Flajolet and Puech (1986) showed that a random partial match query (carried out with s values also drawn uniformly and independently on $[0,1]$ so there are $k-s$ wild cards) has expected time performance $\Theta(n^{1-s/k+\theta(s/k)})$, where $\theta(u)$ is a strictly positive function of $u\in(0,1)$, with maximum not exceeding 0.07. Thus, random k-d trees behave a bit worse than their balanced counterparts, the median k-d trees. Surveys of related known probabilistic results are provided by Vitter and Flajolet (1990) and Gonnet and Baeza-Yates (1991).

We propose a minor modification of the insertion procedure, namely, each time a rectangle is split by a newly inserted leaf point, the longest side of its rectangle is cut, that is, the cut is a $(k-1)$-dimensional hyperplane through the new point perpendicular to the longest edge of the rectangle. It was shown by Chanzy, Devroye, and Zamora-Cura (1999) that elongated rectangles explain the poor performance of random k-d trees. In this paper, we show to what extent the proposed k-d trees have

more squarish-looking rectangles, and we will therefore call these trees *squarish k-d trees*. For the probabilistic model of Flajolet and Puech, it will be shown that the expected time for a partial match query is $\Theta(n^{1-s/k})$, just as for random median k-d trees. Furthermore, the expected complexity of any orthogonal range search in median k-d trees is asymptotically equivalent to that for the simple squarish k-d trees proposed here.

In the last part of the paper, we deal with orthogonal range search in general when the query rectangles may have dimensions that depend upon n in an arbitrary fashion. The proofs are probabilistic, rather than analytical, and do not offer explicit constants for expected times but only $\Theta(\cdot)$ results. However, they are short and explain many of the phenomena at work. Interestingly, very little probability beyond Hölder's inequality is needed. We conclude the paper by showing that a natural nearest neighbor search (with a randomly selected probe point) takes $O(\log n \log\log n)$ expected time in any dimension.

We should note that there are indeed more sophisticated data structures for some of the subproblems dealt with here. For example, if one is just interested in partial match queries, then one could just make j-d trees for each of the $2^k - 1$ nonempty subsets of size j of the k coordinates separately, so that search in the proper tree is just a point search, taking expected worst-case time $O(\log n)$, while the space used is still $O(n2^k)$. However, these would not be helpful for general orthogonal, simplex, or convex range searches. For an analysis of range search based on multiattribute trees see Gardy, Flajolet, and Puech (1989).

2. The random processes. In this section, we will try to explain the differences between alternating cuts and longest-edge cuts in sequences of randomly cut rectangles. To explain the processes at work, we consider the following simplification of our problem: in $\mathbb{R}^2$, start with a rectangle with one vertex permanently pegged at the origin and the opposite one at $(1,1)$, and let (U_n, V_n) denote the coordinates of the top right vertex after n iterations, with $(U_0, V_0) = (1,1)$. The rectangle will be reduced in size, first by alternating uniform cuts, that is, if $Z_1, Z_2, \ldots$ are independently and identically distributed (i.i.d.) uniform $[0,1]$ random variables, then we set

$$(U_n, V_n) = \begin{cases} (Z_n U_{n-1}, V_{n-1}) & \text{if } n \text{ is odd;} \\ (U_{n-1}, Z_n V_{n-1}) & \text{if } n \text{ is even.} \end{cases}$$

If we denote by $\stackrel{\mathcal{L}}{=}$ equality in distribution, clearly, at time $2n$, we have

$$U_{2n} \stackrel{\mathcal{L}}{=} V_{2n} \stackrel{\mathcal{L}}{=} \prod_{i=1}^{n} Z_i \stackrel{\mathcal{L}}{=} e^{-\sum_{i=1}^{n} E_i} \stackrel{\mathcal{L}}{=} e^{-G_n},$$

where the E_i are independent exponential random variables, and G_n denotes a gamma random variable with parameter n. As U_k and V_j are independent of each other for all k, j, we see that the ratio

$$\frac{U_{2n}}{V_{2n}} \stackrel{\mathcal{L}}{=} e^{G_n - G'_n},$$

where G_n, G'_n are i.i.d. gamma random variables. By the central limit theorem, it is easy to see that

$$\frac{1}{\sqrt{n}} \log\left(\frac{U_{2n}}{V_{2n}}\right) \stackrel{\mathcal{L}}{\to} \mathcal{N} - \mathcal{N}',$$

FIG. 2. *Two random k-d tree partitions clearly show the elongated rectangles.*

a difference of two independent standard normal random variables. Thus, the raw ratio behaves asymptotically like $\exp(\sqrt{n}(\mathcal{N}-\mathcal{N}'))$, and thus exhibits wide swings. In fact, if we stop at a large value for n, the rectangle will look very skinny indeed (see Figure 2).

Since we would like to preserve squarish rectangles, we may opt instead to always cut the longest side of the rectangle. More formally, with notation as above, $(U_0, V_0) = (1, 1)$, we have

$$(U_n, V_n) = \begin{cases} (Z_n U_{n-1}, V_{n-1}) & \text{if } U_{n-1} > V_{n-1}; \\ (U_{n-1}, Z_n V_{n-1}) & \text{if } U_{n-1} < V_{n-1}. \end{cases}$$

In case of equality $U_{n-1} = V_{n-1}$, which only occurs at $n = 1$, we flip a perfect coin and pick an edge to cut at random.

LEMMA 1. *With the longest-edge cutting method, the sequence U_n/V_n, $n \geq 1$, is identically distributed. The common distribution is that of Z_1/Z_2, the ratio of two independent uniform $[0, 1]$ random variables.*

Proof. Clearly, U_1/V_1 is distributed as Z_1 with probability $1/2$ and as $1/Z_1$ otherwise. It is easy to verify that this has the required density $1/(2\max(z, 1))^2$, $z > 0$. By induction, we need to show that if Z_1, Z_2, Z are i.i.d. uniform $[0, 1]$ random variables, then the random variable $ZZ_1/Z_2 I_{Z_1 > Z_2} + Z_1/(ZZ_2) I_{Z_1 < Z_2}$ is in turn distributed as Z_1/Z_2. This can be done by standard calculations, or even the method of characteristic functions. However, by far the quickest way to see this is by embedding. We note that Z_1/Z_2 is distributed as the random variable Z_4^S where $S = 1$ and $S = -1$ with equal probability, and Z_4 is another uniform $[0, 1]$ random variable. The case $Z_1 > Z_2$ corresponds to $S = -1$, and thus we see that $ZZ_1/Z_2 I_{Z_1 > Z_2} + Z_1/(ZZ_2) I_{Z_1 < Z_2}$ is distributed as $(Z_4/Z)^S$, which was to be shown, as S is independent of Z and Z_4. □

Lemma 1 shows that cutting the longest edge is extremely stabilizing. Nevertheless, as U_n/V_n has Cauchy-like tails, its mean does not exist, and we will often see skinny rectangles, although by and large the rectangles will be rather squarish (see Figures 3 and 4). The above observations explain why the squarish k-d trees are useful. Our analysis is of course more involved, as rectangles participate in an evolv-

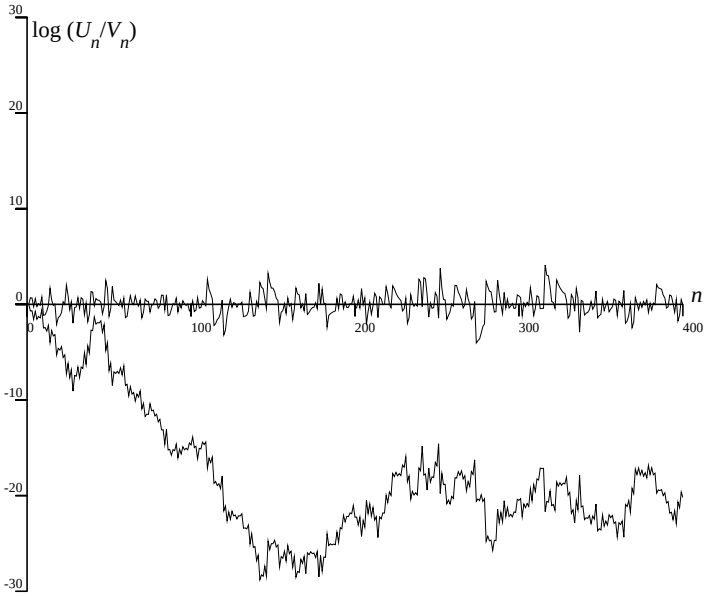


FIG. 3. *For the two processes above,* $\log(U_n/V_n)$ *is plotted versus* n*. The alternating cuts process wanders off just as a random walk. The largest edge cut strategy induces a sequence* U_n/V_n *that hovers near one and remains stationary.*

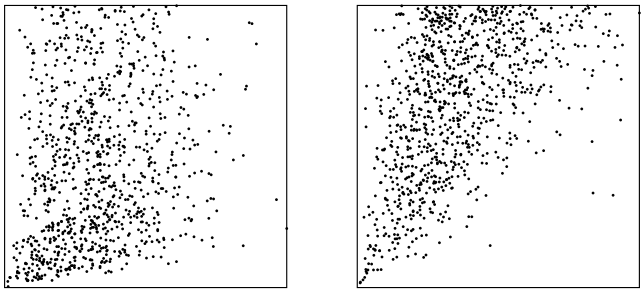


FIG. 4. *For the two processes above, let* L_n *and* S_n *be the long and short dimensions of a leaf rectangle in the* 2*-d partition. The values* $\sqrt{L_n/S_n}$ *are plotted versus* $\sqrt{L_n S_n}$ *(normalized so that the largest value is one) for the two k-d trees. For squarish k-d trees (on the right), there are many more rectangles in which* L_n *and* S_n *are close. And nearly all big rectangles are squarish. For the ordinary random* 2*-d tree (on the left) most rectangles have very small edge ratios.*

ing collection of rectangles, with very intricate dependencies. As soon as a rectangle becomes too small, it is unlikely to be picked again soon, and thus, the ratio of the sides of the rectangles must be considered in conjunction with the sizes. For this, we introduce a few new analysis methods.

3. Notation and preliminaries. In k-d trees, nodes represent rectangular regions. Bentley's algorithm for orthogonal range search and partial match queries starts at the root of a k-d tree and recursively visits all subtrees that have a nonempty overlap with the rectangular regions of the children, and reports all points that fall in the search region. Let $u_1, u_2, \ldots, u_n$, $n \geq 1$, denote the nodes in the k-d tree, and let $U_1, \ldots, U_n$ denote the data points, which are i.i.d. and uniformly distributed on

$[0,1]^k$. Thus, U_i is the data point corresponding to u_i. The rectangle split by u_i is R_i. Thus, $R_1 = [0,1]^k$. Let $|R_i|$ denote the volume of rectangle i. The $n+1$ leaf rectangles (the dangling edges in Figure 1) are also denoted R_i, with the index i now running from $n+1$ to $2n+1$. The collection of rectangles is denoted by $\mathcal{R}_n$. The collection of the indices of the $n+1$ final rectangles is $\mathcal{F}_n$. We will denote by T the k-d tree constructed by inserting successively $u_1, u_2, \ldots, u_n$ into an initially empty k-d tree. Given a node u in T, we will denote by T_u the subtree of T rooted at u. With rotating coordinate cuts, such a tree is called a *random k-d tree*. With our method of cutting the longest edges, it will be called a *random squarish k-d tree*.

The dimensions of rectangle R_i are $X_{ij}, 1 \le j \le k$. For 2-d trees, we will use the lighter notation X_i, Y_i for the x and y dimensions of R_i. The query rectangle Q is $Z + [-m_1, m_1] \times \cdots \times [-m_k, m_k]$, $m_i \ge 0$ for all i, where the m_i's are fixed (that is, they may depend upon n only) and Z is uniformly distributed on $[0,1]^k$ and independent of $(U_1, \ldots, U_n)$. Bentley's range search applied to Q is called a *random orthogonal range search*. Note that a node u_i is visited by the range search algorithm if and only if the query rectangle Q intersects R_i. Any rectangle R_i is visited if and only if it intersects Q. Let N_n be the time complexity of Bentley's orthogonal range search. Then,

$$N_n = \sum_{i=1}^{2n+1} \mathbf{1}_{[R_i \cap Q \neq \emptyset]},$$

where $\mathbf{1}_{[A]}$ is the characteristic function of the event A. This quantity will be analyzed further on for random squarish k-d trees.

In a *random partial match query*, we specify a subset of s dimensions, $j_1, \ldots, j_s$, and perform an orthogonal range query with the ith interval in the rectangle either $\{Z_i\}$ (a uniform random number on $[0,1]$) if $i \in \{j_1, \ldots, j_s\}$, or $(-\infty, \infty)$ otherwise. It is assumed that the Z_i's are independent, and independent of $(U_1, \ldots, U_n)$. In this paper, we first study random partial match queries for random squarish k-d trees and obtain results that should be compared against the following result for random k-d trees.

THEOREM 1 (Flajolet and Puech (1986)). *For a random k-d tree and a random partial match query, in which s of the k fields are specified with $k > s \ge 0$, let $N_n^{(s)}$ be the number of comparisons that Bentley's orthogonal range search performs. Define*

$$\alpha(u) = \max_{0 \le t \le 1} \left\{ t + 2 \left(\frac{1-t}{1-u} \right)^{1-u} \left(\frac{t}{u} \right)^{u} - 2 \right\}, \ 0 < u < 1,$$

and note in particular that α is decreasing on $(0,1)$, $\alpha(0) = 1$, and that $1 - u < \alpha(u) < 1.07 - u$, $0 < u < 1$. Then, the expected value of $N_n^{(s)}$ is such that

$$\mathbf{E}\left\{ N_n^{(s)} \right\} = (c + o(1)) n^{\alpha(s/k)},$$

where c is a constant depending on the indices of the s fixed coordinates.

The following proposition is useful in relating partial random partial match queries to the range search problem.

PROPOSITION 1. *Given is a random k-d tree based on i.i.d. random variables $U_1, \ldots, U_n$, distributed uniformly on $[0,1]^k$. Consider a random partial match query, in which $s \ge 0$ of the k fields are specified. Let $N_n^{(s)}$ be the number of comparisons that*

Bentley's orthogonal range search performs. Let S be the set of specified coordinates. Then

$$\mathbf{E}\left\{N_n^{(s)}\right\} = \mathbf{E}\left\{\sum_{i=1}^{2n+1}\prod_{j\in S} X_{ij}\right\},$$

where $X_{ij}, 1 \le j \le k$, are the lengths of the sides of rectangle R_i in $\mathcal{R}_n$.

Proof. Let Q be the query rectangle. Note that $\mathbf{P}\{Q \cap R_i \neq \emptyset \,|\, U_1, \ldots, U_n\} = \prod_{j\in S} X_{ij}$. Thus we have

$$\mathbf{E}\left\{N_n^{(s)}\right\} = \mathbf{E}\left\{\sum_{i=1}^{2n+1} \mathbf{1}_{[Q\cap R_i\neq\emptyset]}\right\} = \sum_{i=1}^{2n+1}\mathbf{P}\{Q\cap R_i\neq\emptyset\} = \mathbf{E}\left\{\sum_{i=1}^{2n+1}\prod_{j\in S} X_{ij}\right\}. \qquad \square$$

The next observation is important. It follows immediately by considering the random growth of our k-d trees, and, of course, it implies that the joint distribution of the ordered volumes of the $n+1$ leaf rectangles is identical for both random k-d trees considered here!

LEMMA 2. *Consider a random k-d tree or a random squarish k-d tree. Then, the volumes of the rectangles in $\mathcal{F}_n$ are distributed as the set $\mathcal{V}_n$ of the consecutive spacings between the order statistics of n i.i.d. random variables, uniformly distributed on $[0,1]$.*

4. Random partial match queries with squarish 2-d trees. In a vertical random partial match query on a 2-d tree, we take a uniformly distributed value Z and visit all nodes in the tree whose rectangle cuts the vertical line at Z. Horizontal partial match queries on 2-d trees are defined in an analogous manner. The probability of hitting a rectangle with dimensions $X_i \times Y_i$ is of course X_i, so that the expected number of nodes visited, and hence the expected time for a partial match query, is simply $\mathbf{E}\{\sum_{i=1}^{2n+1} X_i\}$, where the sum is taken over all $2n+1$ rectangles in the partition. A similar formula holds, of course, for horizontal partial match queries. In this section, we prove that a random partial match query in a random squarish 2-d tree takes expected time $\Theta(\sqrt{n})$ as opposed to $\Theta(n^{0.5616})$ for random 2-d trees (see Theorem 1).

THEOREM 2. *For random squarish 2-d trees,*

$$\frac{\sqrt{\pi n}}{3} \le \mathbf{E}\left\{\sum_{i=1}^{2n+1} Y_i\right\} \le 180\sqrt{n}.$$

The same result holds for $\mathbf{E}\{\sum_{i=1}^{2n+1} X_i\}$. Hence, the expected time for a random partial match query is $\Theta(\sqrt{n})$.

Of course, no attempt was made to optimize the constants. A few technical results will be needed in what follows. In particular, the next lemma is valid for squarish k-d trees in arbitrary dimension.

LEMMA 3. *For random squarish k-d trees, let $p \ge 0, n \ge 1$; then*

$$\left(\frac{1}{1+p}\right)^{\lfloor p\rfloor+1}\frac{\Gamma(p+1)}{n^{p-1}} \le \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}|R_i|^p\right\} \le \frac{4\Gamma(p+1)}{n^{p-1}}$$

for all n.

Proof. Let $V_1, \ldots, V_{n+1}$ be the spacings induced by n independent uniformly distributed random variables on $[0,1]$. It is known that $V_i \stackrel{\mathcal{L}}{=} \text{Beta}(1,n)$. Thus, by Lemma 2, with $\text{B}(s,t) = \frac{\Gamma(s)\Gamma(t)}{\Gamma(s+t)}$,

$$\begin{aligned}\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}|R_i|^p\right\} &= \mathbf{E}\left\{\sum_{i=1}^{n+1}V_i^p\right\} = \sum_{i=1}^{n+1}\int_0^1 v^p\frac{(1-v)^{n-1}}{\text{B}(1,n)}dv \\ &= (n+1)\frac{\text{B}(p+1,n)}{\text{B}(1,n)} = \Gamma(p+1)\frac{\Gamma(n+2)}{\Gamma(p+n+1)}\ .\end{aligned}$$

Now, $\Gamma(x+1) = x\Gamma(x)$ for any $x > 0$, and for any natural number n and any $s \in [0,1]$, $n^{1-s} \le \Gamma(n+1)/\Gamma(n+s) \le (n+1)^{1-s}$ (see Mitrinović (1970)). Thus,

$$\begin{aligned}\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}|R_i|^p\right\} &= \Gamma(p+1)(n+1)\frac{\Gamma(n+1)}{(n+p)\cdots(n+p-\lfloor p\rfloor)\Gamma(n+p-\lfloor p\rfloor)} \\ &\le \frac{\Gamma(p+1)(n+1)^{2-p+\lfloor p\rfloor}}{n^{\lfloor p\rfloor+1}} \\ &= \frac{\Gamma(p+1)}{n^{p-1}}\left(\frac{n+1}{n}\right)^{2+\lfloor p\rfloor-p} \\ &\le \frac{4\Gamma(p+1)}{n^{p-1}},\end{aligned}$$

as $2 + \lfloor p\rfloor - p \le 2$. Now, for the lower bound, note that

$$\begin{aligned}\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}|R_i|^p\right\} &= \Gamma(p+1)(n+1)\frac{\Gamma(n+1)}{(n+p)\cdots(n+p-\lfloor p\rfloor)\Gamma(n+p-\lfloor p\rfloor)} \\ &\ge \frac{\Gamma(p+1)}{n^{p-1}}\frac{n^{\lfloor p\rfloor+1}}{(n+p)\cdots(n+p-\lfloor p\rfloor)} \\ &\ge \frac{\Gamma(p+1)}{n^{p-1}}\left(\frac{n}{n+p}\right)^{\lfloor p\rfloor+1} \\ &\ge \frac{\Gamma(p+1)}{n^{p-1}}\left(\frac{1}{1+p}\right)^{\lfloor p\rfloor+1}.\end{aligned}$$ □

LEMMA 4. *In a random squarish 2-d tree constructed from the insertion of $U_1, \ldots, U_n$ independent and uniformly distributed random vectors on $[0,1]^2$ we have that for every $q \ge 1$,*

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}Y_i^q\right\} \le \begin{cases}\frac{8}{1-q/2}n^{1-q/2} & \text{for } q\in[1,2);\\ 8e\log n & \text{for } 2-\frac{2}{\log n}\le q\le 2;\\ \frac{5\Gamma(q/2+1)}{q/2-1}\left(\frac{q}{2}-\frac{1}{n^{q/2-1}}\right) & \text{for } q>2,\end{cases}$$

and for $q \in [1,2)$,

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}Y_i^q\right\} \ge \left(\frac{1}{q/2+1}\right)^{\lfloor q/2\rfloor+1}\Gamma(q/2+1)n^{1-q/2}.$$

The same result holds for $\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}X_i^q\right\}$.

Proof. Let $r > 1$, and define $S_r^{(q)} = \sum_{i\in\mathcal{F}_r} Y_i^q$. Note that, given $U_1,\dots,U_r$, $S_{r+1}^{(q)} - S_r^{(q)}$ is distributed as Y^q when $X > Y$ and as $Y^q(U^q + (1-U)^q - 1)$ when $X \le Y$, where U is a uniform $[0,1]$ random variable, and (X,Y) are the dimensions of the rectangle split when U_{r+1} is added. Thus,

$$\mathbf{E}\left\{S_{r+1}^{(q)} - S_r^{(q)}\right\} = \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} X_iY_i\left(\mathbf{1}_{[X_i>Y_i]}Y_i^q + \mathbf{1}_{[X_i<Y_i]}Y_i^q\left(U^q + (1-U)^q - 1\right)\right)\right\}.$$

Notice that $U^q + (1-U)^q - 1 \le 0$ for $q \ge 1$, and as $\min\{a,b\} \le \sqrt{ab}$, for $a,b \ge 0$, then by Lemma 3,

$$\begin{aligned}\mathbf{E}\left\{S_{r+1}^{(q)} - S_r^{(q)}\right\} &\le \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} X_iY_i\left(\mathbf{1}_{[X_i>Y_i]}Y_i^q\right)\right\}\\ &\le \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}(X_iY_i)^{q/2+1}\right\} \le \frac{4\Gamma(q/2+2)}{r^{q/2}}.\end{aligned}$$

By summing the differences, we get

$$\begin{aligned}\mathbf{E}\left\{S_n^{(q)}\right\} &= \mathbf{E}\left\{\sum_{r=1}^{n-1}\left(S_{r+1}^{(q)} - S_r^{(q)}\right) + S_1^{(q)}\right\}\\ &\le \sum_{r=1}^{n-1}\frac{4\Gamma(q/2+2)}{r^{q/2}} + 2\\ &\le 2 + 4\Gamma(q/2+2)\left(1 + \int_1^{n-1}\frac{1}{x^{q/2}}dx\right)\\ &\le \begin{cases} 10 + \frac{4\Gamma(q/2+2)}{1-q/2}(n^{1-q/2} - 1) & (q\in[1,2)),\\ 5\Gamma(q/2+2) + \frac{4\Gamma(q/2+2)}{q/2-1}(1-n^{1-q/2}) & (q>2)\end{cases}\\ &\le \begin{cases} \frac{8}{1-q/2}n^{1-q/2} & (q\in[1,2)),\\ \frac{5\Gamma(q/2+2)}{q/2-1}\left(\frac{q}{2} - n^{1-q/2}\right) & (q>2).\end{cases}\end{aligned}$$

Because $\frac{8}{1-q/2}n^{1-q/2}$, as a function of q, reaches its minimum at $q_0 = 2(1-1/\log n)$, and $\mathbf{E}\{S_n^{(q)}\}$ is a decreasing function of q, we have that $\mathbf{E}\{S_n^{(q)}\} \le 8e\log n$, for $q_0 \le q \le 2$. The result for $\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} X_i^q\right\}$ can be obtained similarly just by replacing the y-lengths for the x-lengths in the appropriate places.

Now, for the lower bound, note that as the X_i's and the Y_i's are identically distributed,

$$\begin{aligned}\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} Y_i^q\right\} &= \frac{1}{2}\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}(Y_i^q + X_i^q)\right\}\\ &\ge \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}(Y_iX_i)^{q/2}\right\}\\ &\ge \left(\frac{1}{q/2+1}\right)^{\lfloor q/2\rfloor+1}\frac{\Gamma(q/2+1)}{n^{q/2-1}},\end{aligned}$$

by Lemma 3, for $q \in [1,2)$. ∎

Proof of Theorem 2. Note that the lower bound follows from Lemma 4, as $\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} Y_i\right\}$ is less than $\mathbf{E}\{\sum_{i=1}^{2n+1} Y_i\}$. For the upper bound we will use the same technique as in the proof of Lemma 4. Let $S_n = \sum_{i=1}^{2n+1} Y_i$. Note that as the sum is over all the rectangles generated by $U_1, \ldots, U_n$, we have now that for $r \geq 1$, as X_i and Y_i are identically distributed,

$$\begin{aligned}\mathbf{E}\left\{S_{r+1} - S_r\right\} &= \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} X_i Y_i \left(\mathbf{1}_{[X_i>Y_i]} 2Y_i + \mathbf{1}_{[X_i<Y_i]}(Y_i U + Y_i(1-U))\right)\right\} \\ &\leq 3\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} X_i Y_i^2\right\},\end{aligned}$$

where $U \stackrel{\mathcal{L}}{=} \text{Uniform}[0,1]$, and independent of all $U_1, \ldots, U_n$. Let $q \in (1,2)$ and $p > 1$ such that $\frac{1}{p} + \frac{1}{q} = 1$, then by Hölder's inequality used twice,

$$\begin{aligned}\mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} X_i Y_i^2\right\} &\leq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} (X_i Y_i)^p\right\}^{1/p} \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r} Y_i^q\right\}^{1/q} \\ &\leq \left(\frac{4\Gamma(p+1)}{r^{p-1}}\right)^{1/p} \left(\frac{8}{1-q/2}\frac{1}{r^{q/2-1}}\right)^{1/q}\end{aligned}$$

by Lemmas 3 and 4. Take $p = 3$, $q = 3/2$, and verify that the upper bound is not more than $24^{1/3}32^{2/3}/\sqrt{r} < 30/\sqrt{r}$. By summing the differences we finally obtain

$$\mathbf{E}\left\{\sum_{i=1}^{2n+1} Y_i\right\} \leq \frac{5}{2} + 90\sum_{r=1}^{n-1}\frac{1}{\sqrt{r}} \leq \frac{5}{2} + 90(2\sqrt{n-1} - 1) \leq 180\sqrt{n}.$$

The result for $\mathbf{E}\{\sum_{i=1}^{2n+1} X_i\}$ can be obtained similarly just by replacing the y-lengths for the x-lengths in the appropriate places. □

5. The *k*-dimensional case. In this section, we obtain the k-dimensional generalization of the results in the previous section by induction. Given $U_1, \ldots, U_n$, we define for each $R_i \in \mathcal{R}_n$, $X_i^* = \max_{j=1,\ldots,k} X_{ij}$ and j_i^* as the index $j \in \{1, \ldots, n\}$ for which $X_{ij} = X_i^*$. Note that j_i^* is unique with probability one. Our main result generalizes Theorem 2 and establishes the expected time optimality of random squarish k-d trees.

THEOREM 3. *Consider a random squarish k-d tree. For $\ell \in \{1, \ldots, k-1\}$, there exist $C, C' > 0$ such that*

$$C' n^{1-\frac{\ell}{k}} \leq \mathbf{E}\left\{\sum_{i=1}^{2n+1}\prod_{j\in I} X_{ij}\right\} \leq C n^{1-\frac{\ell}{k}},$$

for any $I \subseteq \{1, \ldots, k\}$ of cardinality ℓ and all $n \in \mathbb{N}$. In particular, by Proposition 1 *the expected time of a random partial match query with s specified coordinates is $\Theta(n^{1-s/k})$.*

The next lemma complements Theorem 3 when $\ell = k$.

LEMMA 5. *Let $U_1, \ldots, U_n$ be independent uniformly distributed random variables over $[0,1]^k$. Let $\mathcal{R}_n = \{R_1, R_2, \ldots, R_{2n+1}\}$ be the hyperrectangles in the partition*

defined by the random squarish k-d tree based on $U_1, \ldots, U_n$. Let X_{ij} be the length on the jth coordinate of the ith hyperrectangle. Then,

$$\mathbf{E}\left\{\sum_{i=1}^{2n+1} X_{i1}\cdots X_{ik}\right\} = 2h_{n+1} - 1,$$

where h_n is the nth harmonic number.

We prove the following lemma that will allow us to prove the lower bound in the previous theorem.

LEMMA 6. *Let $\ell \in \{1, \ldots, k\}$; then for every $x_1, \ldots, x_k > 0$,*

$$\left(\prod_{j=1}^{k} x_j\right)^{\frac{1}{k}} \leq \max_{I:\ I\subseteq\{1,\ldots,k\}\atop |I|=\ell} \left(\prod_{j\in I} x_j\right)^{\frac{1}{\ell}}.$$

Proof. Let I^* be the subset of $\{1, \ldots, k\}$ of cardinality ℓ for which the maximum above is reached. It suffices to observe that

$$\left(\prod_{j=1}^{k} x_j\right)^{\ell} = \prod_{s=1}^{k}\left(\prod_{j=s}^{s+\ell-1} x_j\right) \leq \prod_{s=0}^{k-1}\left(\prod_{j\in I^*} x_j\right) = \left(\prod_{j\in I^*} x_j\right)^{k},$$

where the subindice j must be understood as (j mod k), if $j > k$. ∎

PROPOSITION 2. *Let $I \subseteq \{1, \ldots, k\}$ of cardinality $\ell \in \{1, \ldots, k\}$ and $p \in [1, \frac{k}{\ell})$; then there are positive constants C and C' such that*

$$C' n^{1-p\frac{\ell}{k}} \leq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\left(\prod_{j\in I} X_{ij}\right)^{p}\right\} \leq C n^{1-p\frac{\ell}{k}}$$

for all $n \in \mathbb{N}$.

Proof. For $I \subseteq \{1, \ldots, k\}$ with $|I| = \ell$, we define

$$S_r^{I,p} = \sum_{i\in\mathcal{F}_r}\left(\prod_{j\in I} X_{ij}\right)^{p}.$$

We first look at the upper bound. We define recursively the constants $C_k(\ell, p)$ for any integer $k > 0$, $\ell \in \{1, \ldots, k\}$ and real number $p \in [1, \frac{k}{\ell})$ as follows:

$$C_k(\ell,p) = \begin{cases} 4\Gamma(p+1) & \text{if } \ell = k; \\ (k-\ell)\left(\frac{1}{1-\frac{p\ell}{k}}\right) C_k(k,\tilde{q})^{1/\tilde{q}} C_k(\ell+1, p\tilde{p}\ell/(\ell+1))^{1/\tilde{p}} + 2 & \text{if } \ell < k, \end{cases}$$

where $\tilde{p}, \tilde{q} > 1$ depend on p, k, and ℓ, they are such that $\frac{1}{\tilde{p}} + \frac{1}{\tilde{q}} = 1$, and $1 \leq p\tilde{p}\frac{\ell}{\ell+1} < \frac{k}{\ell+1}$. For the sake of clarity we will choose $\tilde{p}$ later.

For $\ell \in \{2, \ldots, k\}$, we define the hypothesis $\mathcal{H}_\ell$ stating that the upper bound holds for all $n \in \mathbb{N}$, all $I \subseteq \{1, \ldots, k\}$ such that $|I| = \ell$, and all $p \in [1, \frac{k}{\ell})$, with constant $C_k(\ell, p)$. We will prove $\mathcal{H}_\ell$ with an inductive argument. First, note that $\mathcal{H}_k$

holds by Lemma 3. Assuming that $\mathcal{H}_\ell$ is true, we will prove $\mathcal{H}_{\ell-1}$. Let $I \subseteq \{1,\dots,k\}$ such that $\ell - 1 = |I| \geq 1$, and $p \in [1, \frac{k}{\ell-1})$. Then for any integer $r \geq 1$, we have

$$\mathbf{E}\left\{S_{r+1}^{I,p} - S_r^{I,p} | U_1,\dots,U_r\right\} = \sum_{i\in\mathcal{F}_r}\left(\prod_{j=1}^k X_{ij}\right)\left\{\mathbf{1}_{[j_i^*\notin I]}\left(\prod_{j\in I} X_{ij}\right)^p + \mathbf{1}_{[j_i^*\in I]}\left(\prod_{j\in I} X_{ij}\right)^p \int_0^1 (x^p + (1-x)^p - 1)dx\right\},$$

as we are using the longest-edge cut method. Since $\int_0^1 (x^p + (1-x)^p - 1)dx \leq 0$ for any $p \geq 1$, we can drop the second term above and take expected values so that

$$\mathbf{E}\left\{S_{r+1}^{I,p} - S_r^{I,p}\right\} \leq \sum_{t\notin I} \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}\left(\prod_{j=1}^k X_{ij}\right)\mathbf{1}_{[j_i^*=t]}\left(\prod_{j\in I} X_{ij}\right)^p\right\}.$$

Let us denote by $E(t)$ the expected value of the tth term above. Observe that $\mathbf{1}_{[j_i^*=t]} X_{ij} \leq X_{ij}^{\frac{\ell-1}{\ell}} X_{it}^{\frac{1}{\ell}}$. Thus we can bound each $E(t)$ as follows:

$$E(t) \leq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}\left(\prod_{j=1}^k X_{ij}\right)\left(\prod_{j\in I\cup\{t\}} X_{ij}\right)^{\frac{\ell-1}{\ell}p}\right\}.$$

Now, for any $\tilde{p}, \tilde{q} > 1$ such that $\frac{1}{\tilde{p}} + \frac{1}{\tilde{q}} = 1$, we have by applying Hölder's inequality twice that

$$E(t) \leq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}\left(\prod_{j=1}^k X_{ij}\right)^{\tilde{q}}\right\}^{\frac{1}{\tilde{q}}} \mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}\left(\prod_{j\in I\cup\{t\}} X_{ij}\right)^{\frac{\ell-1}{\ell}p\tilde{p}}\right\}^{\frac{1}{\tilde{p}}}.$$

We can apply hypothesis $\mathcal{H}_\ell$ to bound the second term above, if we can choose $\tilde{p} > 1$ such that $p\tilde{p}\frac{\ell-1}{\ell} \in [1, k/\ell)$. Note that $\frac{k}{p(\ell-1)} > 1$, as $p \in [1, \frac{k}{\ell-1})$. Let us define $\tilde{p} = \max\left\{\sqrt{k/p(\ell-1)}, \frac{\ell}{(\ell-1)p}\right\}$, so that $\tilde{p} > 1$, yet $1 \leq p\tilde{p}\frac{\ell-1}{\ell} < \frac{k}{\ell}$. This completely defines the constant $C_k(\ell, p)$. We can therefore use hypothesis $\mathcal{H}_\ell$ and obtain

$$\begin{aligned} E(t) &\leq \left(\frac{C_k(k,\tilde{q})}{r^{\tilde{q}-1}}\right)^{1/\tilde{q}}\left(\frac{C_k(\ell, p\tilde{p}(\ell-1)/\ell)}{r^{\frac{\ell-1}{k}p\tilde{p}-1}}\right)^{1/\tilde{p}} \\ &= \frac{C_k(k,\tilde{q})^{1/\tilde{q}} C_k(\ell, p\tilde{p}(\ell-1)/\ell)^{\frac{1}{\tilde{p}}}}{r^{\frac{\ell-1}{k}p}}. \end{aligned}$$

We can thus bound the differences as follows:

$$\mathbf{E}\left\{S_{r+1}^{I,p} - S_r^{I,p}\right\} \leq \sum_{t\notin I} E(t) \leq \frac{(k-\ell+1) C_k(k,\tilde{q})^{1/\tilde{q}} C_k(\ell, p\tilde{p}(\ell-1)/\ell)^{1/\tilde{p}}}{r^{\frac{\ell-1}{k}p}}.$$

Since $p < \frac{k}{\ell-1}$, we have that $\sum_{r=1}^n \frac{1}{r^{p\frac{\ell-1}{k}}} \leq \frac{1}{1-p\frac{\ell-1}{k}} \frac{n}{n^{p\frac{\ell-1}{k}}}$. So, by summing the differences, we get

$$\mathbf{E}\left\{S_n^{I,p}\right\} \leq \left[C_k(\ell-1,p) - 2\right] n^{1-p\frac{\ell-1}{k}} + 2 \leq C_k(\ell-1,p) n^{1-\frac{p(\ell-1)}{k}}$$

as $\mathbf{E}\{S_1^{I,p}\} \le 2$, for every $p \ge 1$, and any nonempty $I \subseteq \{1,\dots,k\}$. Thus, hypothesis $\mathcal{H}_{\ell-1}$ is proved.

We now prove the lower bound. As we flip a perfect coin at the beginning of the process to choose the side of R_1 that we cut, all the coordinates $X_{i1},\dots,X_{ik}$ of a hyperrectangle R_i are exchangeable. So, denoting by $\mathcal{S}$ the set of all $I' \subseteq \{1,\dots,k\}$ of cardinality ℓ, all the random variables $\sum_{i\in\mathcal{F}_n}\prod_{j\in I'} X_{ij}^p$ are equally distributed so that

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j\in I} X_{ij}^p\right\} = \frac{1}{|\mathcal{S}|}\,\mathbf{E}\left\{\sum_{I'\in\mathcal{S}}\sum_{i\in\mathcal{F}_n}\prod_{j\in I'} X_{ij}^p\right\}.$$

Then, by Lemmas 3 and 6,

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\left(\prod_{j\in I} X_{ij}\right)^p\right\} \ge \frac{1}{|\mathcal{S}|}\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\left(\prod_{j=1}^k X_{ij}\right)^{\frac{p\ell}{k}}\right\} \ge C'\frac{n}{n^{\frac{p\ell}{k}}}. \qquad \square$$

We must note that by Lemma 3, if $\ell = k$, then for any $p \ge 0$, there are positive constants C and C', depending on p such that the previous result holds. We are now ready to prove Theorem 3.

Proof of Theorem 3. The lower bound follows immediately from the previous proposition. For any subset $I \subseteq \{1,\dots,k\}$ of cardinality $\ell \in \{1,\dots,k-1\}$, we define

$$S_n^I = \sum_{i=1}^{2n}\prod_{j\in I} X_{ij}.$$

As we are using the longest-edge cut method we have that

$$\begin{aligned}\mathbf{E}\left\{S_{r+1}^I - S_r^I | U_1,\dots,U_n\right\} &= \sum_{i\in\mathcal{F}_r}\prod_{j=1}^k X_{ij}\left\{\mathbf{1}_{[j_i^*\notin I]}2\prod_{j\in I}X_{ij} + \mathbf{1}_{[j_i^*\in I]}\prod_{j\in I}X_{ij}\right\}\\ &\le 3\sum_{i\in\mathcal{F}_r}\prod_{j=1}^k X_{ij}\prod_{j\in I}X_{ij}.\end{aligned}$$

We choose now $p = \sqrt{k/\ell}$, $q = 1/(1-\sqrt{\ell/k})$, so that $\frac{1}{p}+\frac{1}{q}=1$, and apply Hölder's inequality with these values to get

$$\mathbf{E}\left\{S_{r+1}^I - S_r^I\right\} \le 3\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}\left(\prod_{j=1}^k X_{ij}\right)^p\right\}^{1/p}\mathbf{E}\left\{\sum_{i\in\mathcal{F}_r}\left(\prod_{j\in I}X_{ij}\right)^q\right\}^{1/q}.$$

Then by Lemma 3 and Proposition 2, there exists a positive constant C depending upon ℓ and k such that

$$\mathbf{E}\left\{S_{r+1}^I - S_r^I\right\} \le \frac{C}{r^{\frac{\ell}{k}}}.$$

We add the differences to get

$$\mathbf{E}\left\{S_n^I\right\} \le C\left(\sum_{r=1}^n \frac{1}{r^{\frac{\ell}{k}}}\right) + 2 \le \frac{C}{1-\frac{\ell}{k}}\left(\frac{n}{n^{\frac{\ell}{k}}}\right) + 2. \qquad \square$$

Proof of Lemma 5. First, note that for any $1 \le i \le n$, $X_{i1} \cdots X_{ik}$ is the volume $|R_i|$ of the hyperrectangle R_i. Note that if $U_1, \ldots, U_i$ have already been inserted in $[0,1]^k$, and U_{i+1} is a new point, then the size of the two hyperrectangles generated by U_{i+1} is equal to the size of the hyperrectangle in the final partition of $[0,1]^k$ in which U_{i+1} falls. Let us denote by $R(U_{i+1})$ this hyperrectangle. Thus,

$$\mathbf{E}\left\{\sum_{i=1}^{2n+1} X_{i1}\cdots X_{ik}\right\} = 1 + \sum_{i=0}^{n-1} \mathbf{E}\left\{\mathbf{E}\left\{|R(U_{i+1})| \mid U_1, \ldots, U_i\right\}\right\},$$

where the 1 accounts for the root hyperrectangle. We claim that $\mathbf{E}\{|R(U_{i+1})|\} = \frac{2}{i+2}$. Note that the claim is obviously true for $i = 0$. Now, suppose that $U_1, \ldots, U_i$ have already been inserted in the squarish k-d tree, so that there are $i+1$ external nodes. These external nodes represent the $i+1$ hyperrectangles partitioning $[0,1]^k$. Let these hyperrectangles be $S_1, \ldots, S_{i+1}$, and let the numbering be so that the leaves are taken from left to right, in order of appearance as leaves in the squarish k-d tree of $U_1, \ldots, U_i$. Then,

$$\begin{aligned}
\mathbf{E}\{|R(U_{i+1})|\} &= \mathbf{E}\left\{\mathbf{E}\left\{\sum_{\ell=1}^{i+1} \mathbf{1}_{[U_{i+1}\in S_\ell]}\,|S_\ell| \;\middle|\; U_1, \ldots, U_i\right\}\right\} \\
&= \mathbf{E}\left\{\sum_{\ell=1}^{i+1} |S_\ell|\,\mathbf{P}\left\{U_{i+1} \in S_\ell \;\middle|\; U_1, \ldots, U_i\right\}\right\} \\
&= \mathbf{E}\left\{\sum_{\ell=1}^{i+1} |S_\ell|^2\right\}.
\end{aligned}$$

By Lemma 2, $(|S_1|, \ldots, |S_{i+1}|)$ are jointly distributed as uniform spacings. All these spacings are identically distributed following a Beta$(1,i)$ distribution. If B is a Beta$(1,i)$ random variable, then we have $\mathbf{E}\{B\} = 1/(i+1)$ and $\mathbf{E}\{B^2\} = 2/((i+1)(i+2))$. Therefore,

$$\mathbf{E}\{|R(U_{i+1})|\} = (i+1)\,\mathbf{E}\{B^2\} = \frac{2}{i+2},$$

and thus

$$1 + \sum_{i=0}^{n-1} \mathbf{E}\{|R(U_{i+1})|\} = 1 + 2(h_{n+1} - 1). \qquad \square$$

6. Orthogonal range search. In this section, we obtain tight upper bounds for the expected complexity for Bentley's range search algorithm. For random orthogonal range search, the following theorem establishes the standard for comparisons. Theorem 5 below then states that random squarish k-d trees are superior to random k-d trees for any kind of random orthogonal range search.

THEOREM 4 (Chanzy, Devroye, and Zamora-Cura (1999)). *Given is a random k-d tree of size n. Let Q be a random query hyperrectangle of dimensions $\Delta_1 \times \cdots \times \Delta_k$ (which are deterministic functions of n taking values in $[0,1]$), with center at Z which is uniformly distributed on $[0,1]^k$, and independent of the k-d tree. Let N_n be the*

number of comparisons that Bentley's orthogonal range search algorithm performs. Then, there exist constants $\gamma > \gamma' > 0$ depending upon k only such that

$$\gamma' \leq \frac{\mathbf{E}\{N_n\}}{\left(\log n + \sum_{\substack{I\subseteq\{1,\ldots,k\}\\ 0\leq|I|<k}} \left(\prod_{j\notin I}\Delta_j\right) n^{\alpha(|I|/k)}\right)} \leq \gamma,$$

where $\alpha(\cdot)$ is the function defined in Theorem 1.

THEOREM 5. *Given is a random squarish k-d tree of size n. Let Q be a random query hyperrectangle of dimensions $\Delta_1 \times \cdots \times \Delta_k$ (which are deterministic functions of n taking values in $[0,1]$), with center at Z which is uniformly distributed on $[0,1]^k$, and independent of the k-d tree. Let N_n be the number of comparisons that Bentley's orthogonal range search algorithm performs. Then, there exist constants $\gamma > \gamma' > 0$ depending upon k only such that*

$$\gamma' \leq \frac{\mathbf{E}\{N_n\}}{\left(\log n + \sum_{\substack{I\subseteq\{1,\ldots,k\}\\ 0\leq|I|<k}} \prod_{j\notin I}\Delta_j n^{1-\frac{|I|}{k}}\right)} \leq \gamma\ .$$

We can rewrite the previous result as

$$\mathbf{E}\{N_n\} \leq \gamma\left(n\prod_{j=1}^{k}\Delta_j + \sum_{\ell=1}^{k-1} n^{1-\frac{\ell}{k}} \sum_{\substack{I\subseteq\{1,\ldots,k\}\\ |I|=\ell}} \prod_{j\notin I}\Delta_j + \log n\right),$$

and therefore by allowing any r of the Δ_j's to be zero, the term that will dominate the previous bound is

$$n^{1-\frac{r}{k}} \sum_{I;|I|=r} \prod_{j\notin I}\Delta_j.$$

For example, when $k=2$, $\Delta = \Theta(1/n^\alpha)$, and $\Delta' = \Theta(1/n^\beta)$, then

$$\mathbf{E}\{N_n\} \leq \gamma\left(n^{1-\alpha-\beta} + n^{\frac{1}{2}-\alpha} + n^{\frac{1}{2}-\beta} + \log n\right).$$

By looking at the different regions in the α-β plane, we obtain

$$\mathbf{E}\{N_n\} \leq \begin{cases} \Theta(\log n) & \text{for } \alpha \geq 1/2 \text{ and } \beta \geq 1/2; \\ \Theta(\max\{n^{1/2-\alpha} n^{1/2-\beta}\}) & \text{for } \alpha > 1/2, \beta < 1/2, \text{ or } \alpha < 1/2, \beta > 1/2; \\ \Theta(n^{1-\alpha-\beta}) & \text{for } \alpha \leq 1/2, \beta \leq 1/2. \end{cases}$$

Note that if $\alpha = 0$ and $\beta \geq 1/2$, or $\beta = 0$ and $\alpha \geq 1/2$, we recover the expected complexity time of the random partial match query problem (see Figure 5).

LEMMA 7. *Let $U_1, \ldots, U_n$ be independent and uniformly distributed over $[0,1]^k$ random variables; let X_i^* be the largest side of the ith hyperrectangle generated by $U_1, \ldots, U_n$. Then, for all $n \geq 0$,*

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} \mathbf{1}_{\left[X_i^* > \frac{1}{2}\right]}\right\} \leq 2^{4k-3}.$$

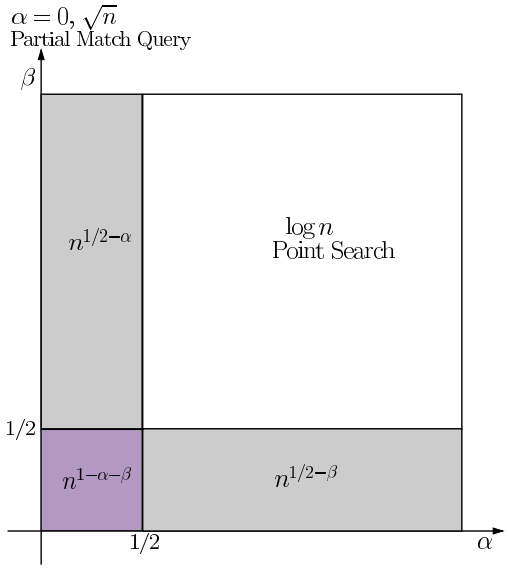


FIG. 5. *The complexity regions for* $\Delta = \Theta(1/n^{\alpha})$ *and* $\Delta' = \Theta(1/n^{\beta})$.

Proof. Note that $\mathbf{E}\{\sum_{i\in\mathcal{F}_n} \mathbf{1}_{[X_i^* > \frac{1}{2}]}\} \le 2^k \mathbf{E}\{\sum_{i\in\mathcal{F}_n} \prod_{j\in I_i} X_{ij}\}$, where $I_i = \{j : X_{ij} > \frac{1}{2}\}$. Define $S_n = \sum_{i\in\mathcal{F}_n}(\prod_{j:X_{ij}>\frac{1}{2}} 8X_{ij})$. We are going to prove that $\mathbf{E}\{S_n\}$ is decreasing so that for $n \ge 1$,

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} \mathbf{1}_{[X_i^* > \frac{1}{2}]}\right\} \le 2^{k-3}\,\mathbf{E}\{S_n\} \le 2^{k-3}\,\mathbf{E}\{S_0\} = 2^{4k-3}.$$

To show $\mathbf{E}\{S_n\} \le \mathbf{E}\{S_0\}$, we look at the differences once again. Set $P_i = \prod_{j\in I_i} 8X_{ij}$. Then,

$$\begin{aligned} S_{r+1} - S_r = \sum_{i\in\mathcal{F}_r} |R_i| \mathbf{1}_{[X_i^* > \frac{1}{2}]} \Bigg\{ & -P_i + \mathbf{1}_{[XX_i^* > \frac{1}{2}]} \left(P_i X + \mathbf{1}_{[|I_i|>1]} \frac{P_i}{8X_i^*} \right) \\ & + \mathbf{1}_{[(1-X)X_i^* > \frac{1}{2}]} \left(P_i(1-X) + \mathbf{1}_{[|I_i|>1]} \frac{P_i}{8X_i^*} \right) \\ & + \mathbf{1}_{[XX_i^* \le \frac{1}{2};\ (1-X)X_i^* \le \frac{1}{2}]} \left(2\mathbf{1}_{[|I_i|>1]} \frac{P_i}{8X_i^*} \right) \Bigg\}, \end{aligned}$$

where $X \stackrel{\mathcal{L}}{=} \text{Uniform}[0,1]$, and it is independent of $U_1, \ldots, U_r$. Therefore,

$$\begin{aligned} \mathbf{E}\{S_{r+1} - S_r | U_1, \ldots, U_r\} &\le \sum_{i\in\mathcal{F}_r} |R_i| \mathbf{1}_{[X_i^* > \frac{1}{2}]} P_i \left(-1 + \int_{\frac{1}{2X_i^*}}^{1} \left(x + \frac{1}{4} \right) dx \right. \\ &\qquad \left. + \int_0^{1-\frac{1}{2X_i^*}} ((1-x) + 1/4) dx + \int_{1-\frac{1}{2X_i^*}}^{\frac{1}{2X_i^*}} 1/2 dx \right) \\ &= \sum_{i\in\mathcal{F}_r} |R_i| \mathbf{1}_{[X_i^* > \frac{1}{2}]} P_i \left(\frac{1}{4X_i^*} - \frac{1}{(2X_i^*)^2} \right) \\ &\le 0. \qquad \square \end{aligned}$$

Proof of Theorem 5. Let T be the squarish k-d tree constructed from $U_1, \ldots, U_n$. Note that a node U_i in T is visited if and only if the query hyperrectangle Q intersects R_i, where R_i is the hyperrectangle in the final partition of $[0,1]^k$ generated by $U_1, \ldots, U_{i-1}$, in which U_i falls. Thus, the running time of the range search algorithm is exactly the number of hyperrectangles in $\mathcal{R}_n$ that Q intersects

$$N_n = \sum_{i=0}^{2n} \mathbf{1}_{[R_i \cap Q \neq \emptyset]}.$$

Also, given $U_1, \ldots, U_n$, the probability that Q intersects R_i is the probability that Z has some coordinate that is within distance $\Delta_j/2$ of R_i, and this probability is clearly bounded by the volume of R_i expanded by Δ_j in the jth direction for all j. Therefore,

$$\begin{aligned}
\mathbf{E}\{N_n\} &\leq \mathbf{E}\left\{\sum_{i=1}^{2n}\prod_{j=1}^{k}(X_{ij}+\Delta_j)\right\} \\
&= \sum_{I\subseteq\{1,\ldots,k\}}\prod_{j\notin I}\Delta_j\,\mathbf{E}\left\{\sum_{i=1}^{2n}\prod_{j\in I}X_{ij}\right\}+1 \\
&\leq \gamma\left(\sum_{\substack{I\subseteq\{1,\ldots,k\}\\ 0\leq|I|<k}}\prod_{j\notin I}\Delta_j n^{1-\frac{|I|}{k}}+\log n\right)
\end{aligned}$$

for some $\gamma > 0$ by Theorem 3 and Lemma 5. For the lower bound we may assume that $\Delta_j \leq 1/2$ and do the following:

$$\begin{aligned}
\mathbf{E}\{N_n\} &\geq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\mathbf{1}_{[Q\cap R_i\neq\emptyset]}\mathbf{1}_{[\forall j\in\{1,\ldots,k\}:X_{ij}\leq 1/2]}\right\} \\
&\geq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j=1}^{k}\left(X_{ij}+\frac{\Delta_j}{2}\right)\mathbf{1}_{[\forall j\in\{1,\ldots,k\}:X_{ij}\leq 1/2]}\right\} \\
&= \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j=1}^{k}\left(X_{ij}+\frac{\Delta_j}{2}\right)\right\}-\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j=1}^{k}\left(X_{ij}+\frac{\Delta_j}{2}\right)\mathbf{1}_{[\exists j\in\{1,\ldots,k\}:X_{ij}>1/2]}\right\} \\
&= \sum_{I\subseteq\{1,\ldots,k\}}\prod_{j\notin I}\frac{\Delta_j}{2}\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j\in I}X_{ij}\right\} \\
&\quad -\sum_{I\subseteq\{1,\ldots,k\}}\prod_{j\notin I}\frac{\Delta_j}{2}\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j\in I}X_{ij}\mathbf{1}_{[\exists j\in\{1,\ldots,k\}:X_{ij}>1/2]}\right\}.
\end{aligned}$$

We can bound the second term above for any given $I \subseteq \{1, \ldots, k\}$ as

$$\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\prod_{j\in I}X_{ij}\mathbf{1}_{[\exists j\in\{1,\ldots,k\}:X_{ij}>1/2]}\right\} \leq \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n}\mathbf{1}_{\left[X_i^*>1/2\right]}\right\} \leq 2^{4k-3}$$

by the previous lemma. Thus, for all n large enough, we can choose $\gamma' > 0$ such that

$$\mathbf{E}\{N_n\} \geq \gamma' \left(\sum_{\substack{I \subseteq \{1,\ldots,k\} \\ 0 \leq |I| < k}} \prod_{j \notin I} \Delta_j n^{1-\frac{|I|}{k}} + \log n \right). \qquad \square$$

7. Nearest neighbor search. We consider two natural nearest neighbor search algorithms. In algorithm **A**, start with an orthogonal range search with a square box of size $1/n^{1/k}$ centered at the query point X. Repeat with boxes of sizes $k^{i/2}/n^{1/k}$ for $i = 0, 1, 2, 3, \ldots$ until $i+1$, where i is the index of the first nonempty box. Report the nearest point in the $(i+1)$st box. Each orthogonal range search taken individually (for fixed i) takes expected time $O(\log n)$ by Theorem 5. We show in fact that the total expected time is $O(\log n \log\log n)$.

THEOREM 6. *Let X be a point uniformly distributed on $[0,1]^k$. Consider a squarish k-d tree based on n i.i.d. points on $[0,1]^k$. Then the expected time of algorithm* **A** *is $O(\log n \log\log n)$.*

Proof. Let $\mathcal{T}$ be the total time it takes algorithm **A** to finish. Let $\mathcal{T}_i$ be the running time of Bentley's range search algorithm on n i.i.d. points on $[0,1]^k$ and a cube Q_i centered at X of length $k^{i/2}/n^{1/k}$, and let M_i be the number of points in Q_i. Note that

$$\mathbf{E}\{\mathcal{T}\} \leq O(\log n) + \mathbf{E}\left\{ \mathcal{T}_1 + \mathcal{T}_2 + \sum_{i=3}^{m} \mathcal{T}_i \mathbf{1}_{[M_{i-2}=0]} \right\},$$

where $m = \lfloor \frac{2}{k} \log_k(2^k n) \rfloor$ bounds the maximum number of iterations the algorithms can perform. Thus, it is enough to prove that $\mathbf{E}\left\{\sum_{i=3}^{m} \mathcal{T}_i \mathbf{1}_{[M_{i-2}=0]}\right\} = O(\log n \log\log n)$. Let $t = \lceil \frac{2}{k} \log_k(2^k \log n) \rceil$; then

$$\mathbf{E}\left\{ \sum_{i=3}^{m} T_i \mathbf{1}_{[M_{i-2}=0]} \right\} \leq (t+1)\,\mathbf{E}\{\mathcal{T}_{t+1}\} + 2n \sum_{i=t+2}^{m} \mathbf{P}\{M_{i-2}=0\}.$$

Now, by Theorem 5,

$$\begin{aligned}
&(t+1)\,\mathbf{E}\{\mathcal{T}_{t+1}\} \\
&\leq \gamma \left(\frac{2}{k} \log_k(2^k \log n) + 2 \right) \left(k^{(t+1)k/2} + \sum_{\ell=1}^{k-1} n^{1-\ell/k} \sum_{\substack{I \subseteq \{1,\ldots,k\} \\ |I|=\ell}} \prod_{j \notin I} \frac{k^{(t+1)/2}}{n^{1/k}} + \log n \right) \\
&= \gamma \left(\frac{2}{k} \log_k(2^k \log n) + 2 \right) \left(k^{(t+1)k/2} + \sum_{\ell=1}^{k-1} n^{1-\ell/k} \binom{k}{\ell} \frac{k^{(t+1)(k-\ell)/2}}{n^{1-\ell/k}} + \log n \right) \\
&= \gamma \left(\frac{2}{k} \log_k(2^k \log n) + 2 \right) \left(k^{(t+1)k/2} + k^{(t+1)k/2} \sum_{\ell=1}^{k-1} \binom{k}{\ell} k^{-(t+1)\ell/2} + \log n \right) \\
&\leq \gamma \left(\frac{2}{k} \log_k(2^k \log n) + 2 \right) \left(k^{(t+1)k/2} \left(k^{-(t+1)/2} + 1 \right)^k + \log n \right) \\
&\leq \gamma \left(\frac{2}{k} \log_k(2^k \log n) + 2 \right) \left(k^k 2^k \log n \left(\frac{1}{\sqrt{k}(2^k \log n)^{1/k}} + 1 \right)^k + \log n \right) \\
&= O(\log n \log\log n)
\end{aligned}$$

for all $n \geq e$. Finally, for $i \leq m$,

$$\mathbf{P}\{M_{i-2} = 0\} \leq \left(1 - \frac{k^{k(i-2)/2}}{2^k n}\right)^n \leq e^{-k^{k(i-2)/2}/2^k},$$

and therefore $\mathbf{P}\{M_{t+2} = 0\} \leq 1/n$. Thus,

$$2n \sum_{i=t+2}^{m} \mathbf{P}\{M_{i-2} = 0\} \leq 2m = O(\log n). \qquad \square$$

Theorem 6 is in contrast with the situation for standard random k-d trees, where algorithm **A** is shown to take expected time $\Theta(n^\rho)$, where $\rho \in (0.061, 0.064)$ depends upon k only (Chanzy, Devroye, and Zamora-Cura (1999)). In algorithm **B**, insert X in the squarish k-d tree, and let Q be the rectangle associated with X. Let X' be the parent of X in the tree (note: $X' \in Q$). Perform an orthogonal range search centered at X with dimensions $2\|X' - X\|$ in all directions. Report the nearest neighbor among all points returned by this orthogonal range search. We will analyze this algorithm for $k = 2$ only.

THEOREM 7. *Let X be a point uniformly distributed on $[0,1]^2$. Consider a squarish 2-d tree based on n i.i.d. points on $[0,1]^2$. Then the expected time of algorithm* **B** *is $O(\log^2 n)$.*

The bound on algorithm **B** is a bit worse than that for algorithm **A**, because while most rectangles are squarish, a sufficient number of them are elongated. In fact, for given $M > 1$, about $1/M$ of the final (leaf) rectangles or more should have an edge ratio exceeding M. For edge ratio M, and considering that all rectangle areas are about $1/n$, we see that the orthogonal range search should take about M points. (The longest edge is about $\sqrt{M/n}$.) The expected number of returned elements is at least $\Theta(\log n)$. And the expected number of leaf rectangles visited is of the same order. But each visited leaf rectangle also induces a visit to all of its ancestors, and there are about $\log n$ of those, hence the claim. The remainder of this section contains the proof of Theorem 7.

LEMMA 8. *Let $Z, U_1, \ldots, U_n$ be independent and uniformly distributed random variables on $[0,1]^2$. Let $X_n(Z)$ and $Y_n(Z)$ be the x-length and y-length of the rectangle in the final partition (of the squarish 2-d tree) induced by $U_1, \ldots, U_n$ in which Z falls. Then, both $n\,\mathbf{E}\{X_n^2(Z)\}$ and $n\,\mathbf{E}\{Y_n^2(Z)\}$ are $O(\log^2 n)$.*

Proof. By Lemmas 3 and 4, for any $p, q > 1$ such that $\frac{1}{p} + \frac{1}{q} = 1$, we have that

$$\begin{aligned}
\mathbf{E}\{X_n^2(Z)\} &= \mathbf{E}\left\{\sum_{i \in \mathcal{F}_n} X_i^3 Y_i\right\} \\
&\leq \mathbf{E}\left\{\sum_{i \in \mathcal{F}_n} (X_i Y_i)^p\right\}^{1/p} \mathbf{E}\left\{\sum_{i \in \mathcal{F}_n} X_i^{2q}\right\}^{1/q} \\
&\leq \left(\frac{4\Gamma(p+1)}{n^{p-1}}\right)^{1/p} \left(\frac{5\Gamma(q+1)}{q-1}\left(q - \frac{1}{n^{q-1}}\right)\right)^{1/q} \\
&= \frac{4^{1/p} 5^{1/q} \left(\Gamma(p+1)\right)^{1/p} \left(\Gamma(q+1)\right)^{1/q}}{(q-1)^{1/q}} \frac{(qn^{q-1} - 1)^{1/q}}{n}.
\end{aligned}$$

Let us choose $q = 1 + \frac{1}{\log n}$, $p = \log n + 1$, and assume $n > e$. As $\Gamma(p+1) \leq \sqrt{2\pi}\left(\frac{p}{e}\right)^p e^{1/12p}$ (see, for example, Abramowitz and Stegun (1970)), there is $c > 0$,

such that $(\Gamma(p+1))^{1/p} \le cp = c(\log n+1)$, and there is $c' > 0$, such that $(\Gamma(q+1))^{1/q} \le c'q \le 4c'$. Furthermore, $(q-1)^{-1/q} = (\log n)^{\frac{\log n}{\log n+1}} \le \log n$, and $(qn^{q-1}-1)^{1/q} \le 2e-1$. Therefore $n\,\mathbf{E}\left\{X_n^2(Z)\right\} = O(\log^2 n)$. The result for $n\,\mathbf{E}\left\{Y_n^2(Z)\right\}$ follows in the same manner. □

LEMMA 9 (see Devroye (1986)). *Let H_n be the height of a random binary search tree of size n; then for any integer $k \ge \max\{1, \log n\}$ we have*

$$\mathbf{P}\left\{H_n \ge k\right\} \le \frac{1}{n}\left(\frac{2e\log n}{k}\right)^k.$$

LEMMA 10. *Let $Z, U_1, \ldots, U_n$ be independent and uniformly distributed random variables over $[0,1]^2$. Let $X_n(Z)$ and $Y_n(Z)$ be the x-length and y-length of the rectangle in the final partition induced by $U_1, \ldots, U_n$ in which Z falls. Then $\mathbf{E}\{X_n(Z)\sum_{i=1}^{2n} X_i\}$, $\mathbf{E}\{Y_n(Z)\sum_{i=1}^{2n} Y_i\}$, $\mathbf{E}\{X_n(Z)\sum_{i=1}^{2n} Y_i\}$, and $\mathbf{E}\{Y_n(Z)\sum_{i=1}^{2n} X_i\}$ are $O(\log^2 n)$.*

Proof. Let $\mathcal{F}_n$ denote the collection of final rectangles in the squarish 2-d tree T constructed from $U_1, \ldots, U_n$. For a final rectangle R_i, denote by $D(R_i)$ its depth. Then $\sum_{i=1}^{2n} X_i \le \sum_{i\in\mathcal{F}_n} D(R_i)X_i + 1$. Thus if H_n is the height of T,

$$\begin{aligned}
\mathbf{E}\left\{\sum_{i=1}^{2n} X_i X_n(Z)\right\} &\le \mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} D(R_i)X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\right\} + 1 \\
&\le \mathbf{E}\left\{H_n \sum_{i\in\mathcal{F}_n} X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\right\} + 1 \\
&\le t\log n\,\mathbf{E}\left\{\sum_{i\in\mathcal{F}_n} X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\right\} + 1 \\
&\quad + \mathbf{E}\left\{\mathbf{1}_{[H_n\ge t\log n]} H_n \sum_{i\in\mathcal{F}_n} X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\right\} + 1
\end{aligned}$$

for any $t > 1$. Using Lemma 9, we see that

$$\mathbf{E}\left\{\mathbf{1}_{[H_n\ge t\log n]} H_n \sum_{i\in\mathcal{F}_n} X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\right\} \le n^3\,\mathbf{P}\left\{H_n \ge t\log n\right\} \le n^2 n^{t\log\left(\frac{2e}{t}\right)}.$$

We choose t such that $t\log\left(\frac{2e}{t}\right) < -2$ so that

$$\mathbf{E}\left\{\mathbf{1}_{[H_n\ge t\log n]} H_n \sum_{i\in\mathcal{F}_n} X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\right\} = O(1).$$

We complete the proof by showing that $\mathbf{E}\{\sum_{i\in\mathcal{F}_n} X_i \sum_{j\in\mathcal{F}_n} X_j^2 Y_j\} = O(\log n)$. For this, let $S_r = \sum_{i\in\mathcal{F}_r} X_i \sum_{j\in\mathcal{F}_r} X_j^2 Y_j$, for $r = 1, \ldots, n-1$. Note that

$$\begin{aligned}
S_{r+1} - S_r = \sum_{m\in\mathcal{F}_r} X_m Y_m \Bigg[&\mathbf{1}_{[X_m<Y_m]} X_m \sum_{j\in\mathcal{F}_r} X_j^2 Y_j \\
&+ \mathbf{1}_{[X_m>Y_m]}((XX_m)^2 Y_m + ((1-X)X_m)^2 Y_m - X_m^2 Y_m) \sum_{i\in\mathcal{F}_r} X_i \Bigg],
\end{aligned}$$

where $X \stackrel{\mathcal{L}}{=} \text{Uniform}[0,1]$, and is independent of all $U_1, \ldots, U_n$. Now, as $(XX_m)^2Y_m + ((1-X)X_m)^2Y_m - X_m^2Y_m \leq 0$, we have that

$$S_{r+1} - S_r \leq \sum_{i \in \mathcal{F}_r} (X_i Y_i)^{3/2} \sum_{j \in \mathcal{F}_r} X_j^2 Y_j.$$

Note that for any $p, q > 1$, such that $\frac{1}{p} + \frac{1}{q} = 1$,

$$\mathbf{E}\{S_{r+1} - S_r\} \leq \mathbf{E}\left\{\left(\sum_{i \in \mathcal{F}_r} (X_i Y_i)^{3/2}\right)^p\right\}^{1/p} \mathbf{E}\left\{\left(\sum_{j \in \mathcal{F}_r} X_j^2 Y_j\right)^q\right\}^{1/q},$$

and again by Hölder's inequality, and Lemma 3, by choosing $q = \sqrt{1.4}$ and $p = \frac{\sqrt{1.4}}{\sqrt{1.4}-1}$,

$$\mathbf{E}\left\{\left(\sum_{i \in \mathcal{F}_r} (X_i Y_i)^{3/2}\right)^p\right\}^{1/p} \leq \mathbf{E}\left\{r^{p/q} \sum_{i \in \mathcal{F}_r} (X_i Y_i)^{3p/2}\right\}^{1/p} \leq \frac{12}{\sqrt{r}}.$$

By applying Hölder's inequality inside the expected value,

$$\begin{aligned}
\mathbf{E}\left\{\left(\sum_{j \in \mathcal{F}_r} X_j^2 Y_j\right)^q\right\}^{1/q} &\leq \mathbf{E}\left\{r^{q/p} \sum_{j \in \mathcal{F}_r} (X_j^2 Y_j)^q\right\}^{1/q} \\
&\leq r^{1/p} \left(\mathbf{E}\left\{\sum_{j \in \mathcal{F}_r} (X_j Y_j)^{qp}\right\}^{1/p} \mathbf{E}\left\{\sum_{j \in \mathcal{F}_r} X_j^{q^2}\right\}^{1/q}\right)^{1/q} \\
&\leq 46\, r^{1/p} \left(\left(\frac{1}{r^{qp-1}}\right)^{1/p} \left(\frac{1}{r^{q^2/2-1}}\right)^{1/q}\right)^{1/q} \\
&= \frac{46}{\sqrt{r}}.
\end{aligned}$$

Thus, $\mathbf{E}\{S_{r+1} - S_r\} \leq 552/r$, and by summing the differences we finally can conclude that $\mathbf{E}\{\sum_{i \in \mathcal{F}_n} X_i \sum_{j \in \mathcal{F}_n} X_j^2 Y_j\}$ is indeed $O(\log n)$. The other expected values can be bounded in the same way. □

Proof of Theorem 7. Given $U_1, \ldots, U_n$, we define $L_n(Z) = 2(X_n(Z) + Y_n(Z))$. Note that as the expected height of T is $O(\log n)$, the expected time complexity of the nearest neighbor algorithm is bounded by $O(\log n)$ plus the expected time of random orthogonal range search with query rectangle Q having all sides of length $L_n(Z)$, and centered at Z. Let N_n be the time complexity of a range search. By the same arguments followed in Theorem 3, we have

$$\mathbf{E}\{N_n\} \leq \mathbf{E}\left\{\sum_{i=1}^{2n+1} X_i Y_i\right\} + 2\,\mathbf{E}\left\{\sum_{i=1}^{2n+1} L_n(Z)(X_i + Y_i)\right\} + 8n\,\mathbf{E}\left\{L_n^2(Z)\right\} + 1.$$

By Lemma 5, $\mathbf{E}\{\sum_{i=1}^{2n+1} X_i Y_i\} = O(\log n)$. For $\mathbf{E}\{\sum_{i=1}^{2n+1} L_n(Z)(X_i + Y_i)\}$, Lemma 10 above shows that it is $O(\log^2 n)$. As $n\,\mathbf{E}\{X_n(Z)Y_n(Z)\} = n\,\mathbf{E}\{\sum_{i \in \mathcal{F}_n} (X_i Y_i)^2\}$, Lemma 3 shows that it is $O(1)$. Finally, by Lemma 8 we have that $n\,\mathbf{E}\{L_n^2(Z)\} = O(\log^2 n)$. Thus the expected running time of algorithm $\mathbf{B}$ is $O(\log^2 n)$. □

8. Further work and open problems.

QUADTREES. For quadtree splitting in k dimensions (Finkel and Bentley (1974), Bentley and Stanat (1975)), it is easy to see that the analysis and thus Theorem 1 are not valid. In fact, for random quadtrees, the expected performance for partial match queries was shown to be of the order of that for standard random k-d trees (Flajolet, Gonnet, Puech, and Robson (1991), (1992)). For orthogonal range search with query rectangles depending upon n, see Chanzy, Devroye, and Zamora-Cura (1999).

EXPECTED WORST-CASE COMPLEXITY. We conjecture that the expected worst-case complexity over all range search rectangles of dimensions Δ_i (but with worst-case location of the center) is also bounded from above by the bound given in Theorem 2. And the expected worst-case time for an s-dimensional partial match query is conjectured to be $O(n^{1-s/k})$ for $s < k$. (For $s = k$, the complexity is clearly bounded by the expected height of the tree, $O(\log n)$.)

NONUNIFORM DISTRIBUTIONS. Finally, we also intend to study the behavior of squarish k-d trees for nonuniform distributions, although it appears once again that the upper bound of Theorem 2 remains valid for all distributions with a joint density on $[0, 1]^k$.

REFERENCES

M. Abramowitz and I. A. Stegun (1970), *Handbook of Mathematical Functions*, Dover Publications, New York.

P. K. Agarwal (1997), *Range searching*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press, Boca Raton, FL, pp. 575–598.

J. L. Bentley (1975), *Multidimensional binary search trees used for associative searching*, Comm. ACM, 18, pp. 509–517.

J. L. Bentley (1979), *Multidimensional binary search trees in database applications*, IEEE Trans. Software Engrg., SE-5, pp. 333–340.

J. L. Bentley and J. H. Friedman (1979), *Data structures for range searching*, ACM Computing Surveys, 11, pp. 397–409.

J. L. Bentley and D. F. Stanat (1975), *Analysis of range searches in quad trees*, Inform. Process. Lett., 3, pp. 170–173.

P. Chanzy, L. Devroye, and C. Zamora-Cura (1999), *Analysis of Range Search for Random k-d Trees*, Technical Report, School of Computer Science, McGill University, Montreal; Acta Inform., to appear.

L. Devroye (1986), *A note on the height of binary search trees*, J. Assoc. Comput. Mach., 33, pp. 489–498.

L. Devroye (1987), *Branching processes in the analysis of the heights of trees*, Acta Inform., 24, pp. 277–298.

R. A. Finkel and J. L. Bentley (1974), *Quad trees: A data structure for retrieval on composite keys*, Acta Inform., 4, pp. 1–9.

P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson (1991), *The analysis of multidimensional searching in quad-trees*, in Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, pp. 100–109.

P. Flajolet, G. Gonnet, C. Puech, and J. M. Robson (1992), *Analytic variations on quadtrees*, Algorithmica, 10, pp. 473–500.

P. Flajolet and C. Puech (1986), *Partial match retrieval of multidimensional data*, J. Assoc. Comput. Mach., 33, pp. 371–407.

D. Gardy, P. Flajolet, and C. Puech (1989), *Average cost of orthogonal range queries in multiattribute trees*, Information Systems, 14, pp. 341–350.

G. H. Gonnet and R. Baeza-Yates (1991), *Handbook of Algorithms and Data Structures*, Addison-Wesley, Workingham.

D. E. Knuth (1997), *The Art of Computer Programming,* Vol. 3*: Sorting and Searching,* 2nd ed., Addison-Wesley, Reading, MA.

D. T. Lee and C. K. Wong (1977), *Worst-case analysis for region and partial region searches in multidimensional binary search trees and quad trees*, Acta Inform., 9, pp. 23–29.

H. M. Mahmoud (1992), *Evolution of Random Search Trees*, John Wiley, New York.

D. S. Mitrinović (1970), *Analytic Inequalities*, Springer-Verlag, New York.
B. Pittel (1984), *On growing random binary trees*, J. Math. Anal. Appl., 103, pp. 461–480.
H. Samet (1990a), *Applications of Spatial Data Structures*, Addison-Wesley, Reading, MA.
H. Samet (1990b), *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA.
J. S. Vitter and P. Flajolet (1990), *Average-case analysis of algorithms and data structures*, in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. van Leeuwen, ed., MIT Press, Amsterdam, pp. 431–524.
F. F. Yao (1990), *Computational geometry*, in Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity, J. van Leeuwen, ed., MIT Press, Amsterdam, pp. 343–389.

SIAM J. COMPUT.
Vol. 30, No. 5, pp. 1701–1738

COMPUTATIONS OF UNIFORM RECURRENCE EQUATIONS USING MINIMAL MEMORY SIZE*

BRUNO GAUJAL†, ALAIN JEAN-MARIE‡, AND JEAN MAIRESSE§

Abstract. We consider a system of uniform recurrence equations of dimension 1 and we show how its computation can be carried out using minimal memory size with several synchronous processors. This result is then applied to register minimization for digital circuits and parallel computation of task graphs.

Key words. uniform recurrence equation, register minimization, circuit design, task graph, (max,+) linear system

AMS subject classifications. 68M07, 93C65, 94C15

PII. S0097539795290350

1. Introduction.

We start by the definition of uniform recurrence equations.

DEFINITION 1.1 (uniform recurrence equations). *We consider $\mathcal{Q}$-valued variables $X_i(n), i \in V, n \in K$, where $\mathcal{Q}$ is an arbitrary set, V is a finite set, and $K \subset \mathbb{Z}^p$ for some $p \in \mathbb{N}$. These variables satisfy*

$$X_i(n) = F_i\Big(X_j(n-\gamma), (j,\gamma) \in \Delta_i\Big) \qquad \forall n \in K\,. \tag{1}$$

The sets Δ_i, called dependence sets*, are finite subsets of $V \times \mathbb{Z}^p$. The collection of equations* (1) *is called a set of* uniform recurrence equations (UREs).

There is no restriction on the generality of the functions F_i except the fact that they are computable. The system $\mathcal{S}$ defined by (1) is said to be uniform because the dependence sets Δ_i do not depend on n. The integers γ are called the *delays*. It is possible to have two delays $\gamma, \gamma' \in \mathbb{Z}^p, \gamma \neq \gamma'$, such that $(j,\gamma) \in \Delta_i$ and $(j,\gamma') \in \Delta_i$.

There are various motivations to study UREs. They appear in the description of differential equations using finite difference methods and in the study of discrete event systems. The case $p > 1$ and $K = \mathbb{Z}^p$ has often been studied in the literature; see [16]. In this case, some of the major issues are constructivity [16] and loop parallelization [8]. These problems and others appearing in this framework will be discussed in Appendix A.

In this paper, we consider only the simple case where $K = \mathbb{Z}$ (systems of dimension 1). The computational model considered is that of parallel processors with a shared memory (CREW-PRAM model: *concurrent read exclusive write parallel random access memory*). More precisely, a computation is performed by a processor, using data stored in the memory. For example, to compute $X_i(n)$, it is necessary to have at least $|\Delta_i|$ memory locations, each location containing one of the data $\{X_j(n-\gamma), (j,\gamma) \in \Delta_i\}$. In a model of parallel processors with shared memory, there

*Received by the editors August 16, 1995; accepted for publication (in revised form) February 5, 2000; published electronically December 7, 2000. Partially supported by the European grant BRA-QMIPS of CEC DG XIII.

http://www.siam.org/journals/sicomp/30-5/29035.html

†INRIA, technopôle de Nancy-Brabois, 615 rue du Jardin Botanique, B.P. 101, F-54602 Villers lès Nancy, France (bruno.gaujal@loria.fr).

‡LIRMM, 161 Rue Ada, 34392 Montpellier Cedex 05, France (ajm@lirmm.fr).

§LIAFA, CNRS-Université Paris 7, Case 7014, 2 place Jussieu, 75251 Paris Cedex 05, France (mairesse@liafa.jussieu.fr).

are several processors that can make computations simultaneously and also access the same memory locations simultaneously.

The problem investigated consists in minimizing the "memory size":

What is the minimal number of memory locations that is needed to compute all the variables $X_i(n)$ of (1) using a CREW-PRAM computational model?

We solve this problem in the recycled case (see section 2.2 for the definition) by proving that it is equivalent to the search for minimal cuts in the dependence graph associated with the system of UREs. This provides polynomial algorithms to compute the minimal memory requirements.

We show that the solution of this problem has many applications. Indeed, UREs appear in the modeling of logical circuits, systolic arrays, and program loops. Our result can be used practically for the optimization of circuit design. Given a digital circuit, we show how to find another circuit with the same functional behavior and using a minimal number of registers. This application will be discussed in section 7.

Our results can also be used in another context, namely, in order to obtain the most efficient representation of task graph systems for parallel computation purposes. The evolution of a task graph can be represented as a linear system over the (max,+) algebra of the form $x(n+1) = A(n)x(n)$, where $x(.) \in \mathbb{R}^k_{\max}$ and $A(n) \in \mathbb{R}^{k\times k}_{\max}$. Our results enable us to obtain a linear representation of a task graph with a minimal dimension k for the matrices $A(n)$. This application will be treated in section 8.

The paper is organized as follows. In section 2, we make more precise the definition of a system of UREs and present two associated graphs, the *dependence graph* and the *reduced graph*. In section 3 we describe the problem that we are going to address. In particular, we restrict our attention to recycled systems of UREs. Sections 4 and 5 investigate the relations that can be found between *cuts* in the dependence graph and the memory size required for an execution of the URE; section 6 presents the interpretation of the above quantities in the reduced graph. Finally in sections 7 and 8, two applications are described, for digital circuits and (max,+) linear systems, respectively. In Appendices A and B, we consider the related problems of scheduling and sequential executions.

2. Basic models. From now on, we consider UREs of dimension 1. More precisely, we consider the set of variables $X_i(n), i \in V, n \in \mathbb{Z}$, and the equations

$$X_i(n) = F_i\Big(X_j(n-\gamma), (j,\gamma) \in \Delta_i\Big),\ n \in \mathbb{Z}\ , \tag{2}$$

where the sets Δ_i are finite subsets of $V \times \mathbb{Z}$.

A system of UREs is *constructive* if given the values of the "negative" variables $X_i(n), n \le 0$ (*initial data*), there exists an ordering of the equations such that, $\forall i, \forall n > 0$, all the variables present in the right-hand side of the equation defining $X_i(n)$ can be computed before $X_i(n)$. Equivalently, the constructivity assumption can be written as follows: For each cycle $(i_1,\gamma_1),\dots,(i_p,\gamma_p), i_{p+1} = i_1$ such that $(i_{j+1},\gamma_{j+1}) \in \Delta_{i_j}$, $j \in \{1,\dots,p\}$, then $\sum_{j=1}^{p} \gamma_j > 0$.

Remark 2.1. Under the constructivity assumption, Farkas' lemma states that it is possible to come back to (1) with all the sets Δ_i included in $V \times \mathbb{N}$ using a simple renumbering of the variables (i.e., $X_i(n) := X_i(n + c_i)$ for some constants $c_i \in \mathbb{Z}$ independent of n). This renumbering actually amounts to a retiming of the system. This notion will be studied in detail in section 6.1.

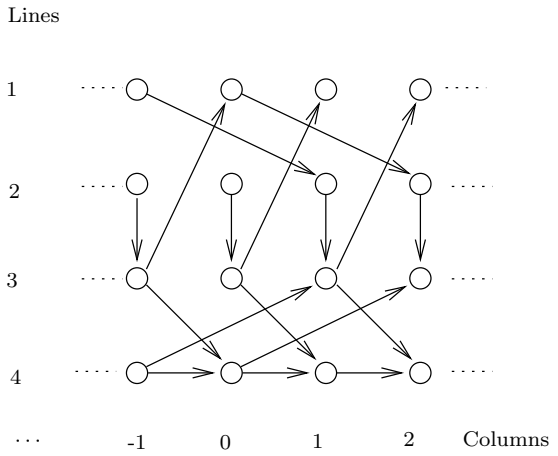


FIG. 1. *Dependence graph associated with the system* $\mathcal{S}$ *of* (3).

From now on, the system $\mathcal{S}$ that we consider is always assumed to be constructive. We present two equivalent ways of describing $\mathcal{S}$: the dependence graph and the reduced graph.

Example 2.2. The illustrative examples in this section correspond to the system

$$\text{(3)} \qquad \begin{cases} X_1(n) &= F_1(X_3(n-1)), \\ X_2(n) &= F_2(X_1(n-2)), \\ X_3(n) &= F_3(X_2(n), X_4(n-2)), \\ X_4(n) &= F_4(X_3(n-1), X_4(n-1)). \end{cases}$$

2.1. Dependence graph. We introduce the graph $\mathcal{D}$ of the dependences between the variables $X_i(n)$.

DEFINITION 2.3 (dependence graph). *The dependence graph associated with a system of UREs is the graph* $\mathcal{D}$ *with* $(V \times \mathbb{Z})$ *as the set of nodes. There is an arc from the node* (i,n) *to the node* (j,m) *if* $X_j(m) = F_j(X_i(n), \dots)$ *or, equivalently, if* $(i, m-n) \in \Delta_j$ *(notation:* $(i,n) \to (j,m)$*).*

The nth *column* in $\mathcal{D}$ is the set of nodes $\{(i,n), i \in V\}$. The ith *line* in $\mathcal{D}$ is the set of nodes $\{(i,n), n \in \mathbb{Z}\}$. In the following, we will refer to nodes $(i,n),\ n \le 0$, as *negative* nodes and nodes $(i,n),\ n \ge 0$, as *positive* nodes.

It is immediate from the definition of a URE that $\mathcal{D}$ is 1-periodic, i.e.,

$$(i,n) \to (j,m) \Longleftrightarrow (i, n+1) \to (j, m+1)\,.$$

The constructivity assumption implies that the graph $\mathcal{D}$ is acyclic. We have represented in Figure 1 the dependence graph corresponding to the system of Example 2.2.

The dependence graph appears under various forms and names in the literature, for example, developed graph, PERT graph, unfolded process graph, and activity network.

2.2. Reduced graph. Since the dependence graph $\mathcal{D}$ is 1-periodic, it can be folded into a *reduced graph* $\mathcal{R}$.

DEFINITION 2.4 (reduced graph). *The reduced graph is an arc-valued graph* $\mathcal{R} = (V, E, \Gamma)$*. The set of nodes is* V *and there is an oriented arc* $e \in E$ *from* i

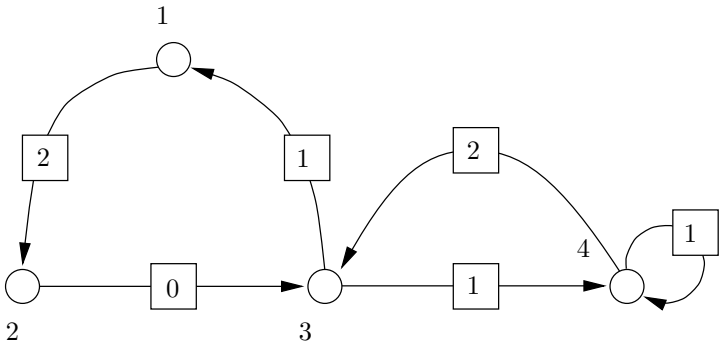


FIG. 2. *Reduced graph associated with the system* $\mathcal{S}$ *of* (3).

to j *if*

$$\exists \gamma \in \mathbb{Z} \text{ such that (s.t.) } (i,\gamma) \in \Delta_j \,. \tag{4}$$

This arc is valued with the delay $\Gamma(e) = \gamma$. *If there exist several delays* γ *verifying condition* (4), E *contains several arcs between the nodes* i *and* j, *with corresponding values. Furthermore, we consider the functions* $F_i, i \in V$, *to be associated with the nodes of* $\mathcal{R}$.

There is an arc from i to j in E if and only if there are arcs from the line $(i,.)$ to the line $(j,.)$ in the dependence graph. The system $\mathcal{S}$ is constructive if and only if the sum of the delays along any circuit in $\mathcal{R}$ is strictly positive.

The reduced graph associated with the system $\mathcal{S}$ of Example 2.2 is represented in Figure 2. The delays γ associated with the arcs are depicted in boxes.

Reduced graphs appear in the literature under the following names: computation graph, synchronous data flow graphs, process graphs, and uniform graphs.

It should be clear from the definitions that there is a one-to-one correspondence between the three models. Indeed, a system can be given by its reduced graph as well as its dependence graph or system of equations.

2.3. Recycled assumption. In the following (sections 4, 5, 7, and 8), we will study only a special case of URE, where the computation of the variable $X_i(n)$ cannot be completed before the computation of $X_i(n-1)$. This case appears naturally in task graphs (see section 8) and in other applications. This constraint can be modeled by imposing a dependence between $X_i(n-1)$ and $X_i(n)$ $\forall i$ and n. Formally, it results in having $(i,1) \in \Delta_i \, \forall i$ for the system of UREs. Equivalently, it results in having a self loop with delay 1 (hence the name recycled) at each node of $\mathcal{R}$, or in having arcs between the nodes (i,n) and $(i,n+1)$ in $\mathcal{D}$. Such arcs will be called *recycling arcs* in what follows. Figure 3 depicts an example of a recycled system.

2.4. Connectedness. We say that a system of UREs is (strongly) connected if the graph $\mathcal{R}$ is (strongly) connected. In the remainder of the paper, we will always consider systems of connected (but not necessarily strongly connected) UREs.

In fact, we will see that, for a recycled connected URE and for games $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$ (to be defined below), a valid computation with a minimal number of memory locations requires all its memories at each instant. It implies that the minimal number of memories necessary to compute a nonconnected recycled URE is the sum of the minimal numbers of memories necessary to compute the different connected components independently.

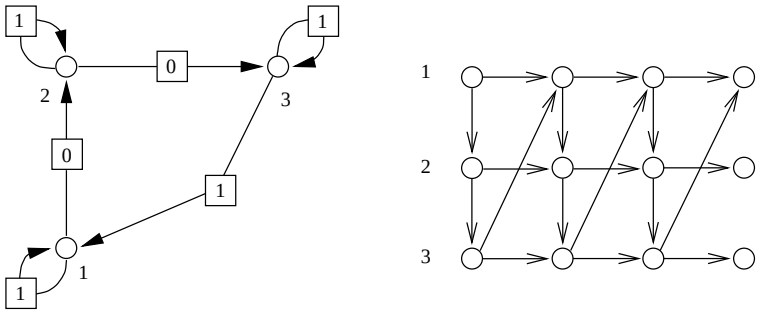


FIG. 3. *Recycled reduced graph and recycled dependence graph.*

3. Synchronous executions. We want to minimize the memory size required in the synchronous computation of a system $\mathcal{S}$.

Among the related problems that have been studied in the field of UREs, we can mention the *basic scheduling* problem and the *sequential* computations. These questions and their relation with the one considered in this paper are discussed in Appendices A and B.

3.1. Pebble games. Let us work with a URE and its associated dependence graph $\mathcal{D}$ as defined in section 2.1. We want to compute iteratively all the variables $X_i(n), n \in \mathbb{N}$. At each step, the variables which are necessary to carry out the computations have to be stored in some memory locations. Our general objective will be to solve the following problem:

What is the minimal number of memory locations
needed to compute all the variables $X_i(n)$*?*

We give a description of this problem in terms of a *pebble game*. A pebble game is played on a graph. At each step, a finite number of pebbles are located on the nodes of the graph, with at most one pebble per node. The position of the pebbles evolves by adding or removing pebbles according to some rules.

Different variants of pebble games have been used in the literature to model memory allocation problems; see, for example, [21, 25]. A pebble corresponds to a memory location, and putting a pebble on a node corresponds to the computation of the variable associated with the node and its storage into the memory. Removing a pebble from a node corresponds to erasing these data from the memory.

Now, we give a more formal definition of the pebble game in our framework. Here, the graph considered is the dependence graph $\mathcal{D}$ of a URE.

DEFINITION 3.1 (configuration). *A configuration is a finite subset of* $V \times \mathbb{Z}$*, the set of nodes of* $\mathcal{D}$*. A configuration represents the position of the pebbles at some stage of the game. There is at most one pebble per node.*

DEFINITION 3.2 (execution, successful execution, step). *An* execution *of the pebble game is a sequence of configurations* $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$*, such that* $\forall t$*, the configuration* $\mathcal{A}(t+1)$ *can be obtained from* $\mathcal{A}(t)$ *through the rules of the pebble game. The passage from* $\mathcal{A}(t-1)$ *to* $\mathcal{A}(t)$ *is called the* step t *of the game. An execution of the game is* successful *if all positive nodes receive a pebble during the execution, i.e.,* $\forall (i,n) \in V \times \mathbb{N}$, $\exists\, t \in \mathbb{N}$, *s.t.* $(i,n) \in \mathcal{A}(t)$.

In the following, we will always consider successful executions and refer to them simply as executions. An execution corresponds to a computation of all the variables

$\{X_i(n), i \in V, n \in \mathbb{N}\}$. The number of pebbles used by an execution $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ is

$$\mathcal{P}(e) \stackrel{\text{def}}{=} \sup_{t\in\mathbb{N}} |\mathcal{A}(t)| , \tag{5}$$

where $|\mathcal{A}(t)|$ represents the cardinal of $\mathcal{A}(t)$. Our general objective is redefined below. It will be referred to as the problem *MinPeb*.

PROBLEM 1 (MinPeb). *Determine* $\min_e \mathcal{P}(e)$ *and an execution* e_o *such that* $\mathcal{P}(e_o) = \min_e \mathcal{P}(e)$.

In the following, we define several sets of rules, each of them defining a different pebble game. The different sets of rules, called $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$, correspond to different computation models for the URE and are related to different notions of cuts and delays (see sections 4 and 5). Also, their relevance will be justified by the applications given in sections 7 and 8. We use the expressions "set of rules $\mathcal{M}_i$" or "game $\mathcal{M}_i$" indifferently.

$\mathcal{M}_1$ *: Synchronous execution.* The set of rules $\mathcal{M}_1$ is as follows:

- (R1) *(starting rule).* Initially, a finite number of pebbles are put on negative nodes only, with at least one pebble on column 0: $\mathcal{A}(0) \subset V \times \mathbb{Z}^-$, $\mathcal{A}(0) \cap (V \times \{0\}) \neq \emptyset$.
- (R2) *(playing rule).* One step of the game consists of any number of moves of type (R3), followed by any number of moves of type (R4).
- (R3) *(adding pebbles).* Put a pebble on an empty node (i, n). At step t, this is possible if and only if each infinite oriented path (see Definition 4.1) ending in (i, n) intersects $\mathcal{A}(t-1)$.
- (R4) *(removing pebbles).* Remove a pebble from a node.

Remark 3.3 (comments of rule (R2)). Note that our definition of $\mathcal{P}(e)$ considers only the number of pebbles at the end of the step and not at intermediate stages (after (R3) and before (R4), for example). It corresponds to the assumption that all the moves done in one step can be performed simultaneously. This is why this is called a synchronous execution. This remark also applies to games $\mathcal{M}_2$ and $\mathcal{M}_3$.

Remark 3.4 (comments on rule (R3)). Rule (R3) may look cumbersome since one may put a pebble on a node which is very far to the right from the current position of the pebbles. Its intuitive meaning for the calculation in a system of UREs is the following: At the beginning of step t, the variables which are in memory are the ones corresponding to $\mathcal{A}(t-1)$. *A new pebble can be put on a node* (i, n) *if the corresponding variable* $X_i(n)$ *can be computed given the variables in memory.* This does not mean that this computation has to be direct. It may be done using the variables in memory and the appropriate compositions of the initial functions F_i. Since the initial functions are arbitrary, no notion of the "complexity of a function" is used here. Hence the function obtained by composition of a finite number of initial functions can be considered just as yet another arbitrary function, and its computation does not require any additional memory.

However, it seems reasonable to consider that function compositions should have a "cost," not in terms of space as mentioned above, but in terms of time. A step of the game may have a duration which depends on the "complexity" of the compositions. The discussion of this aspect of the problem is postponed until Appendix A.2.

We illustrate rule $\mathcal{M}_1$ with the example of Figure 4. We have represented a small part of the dependence graph of the URE: $X_1(n) = F_1(X_1(n-1), X_2(n-1), X_3(n-1)), X_i(n) = F_i(X_{i-1}(n), X_1(n-1), X_2(n-1), X_3(n-1))$ for $i = 2, 3$.

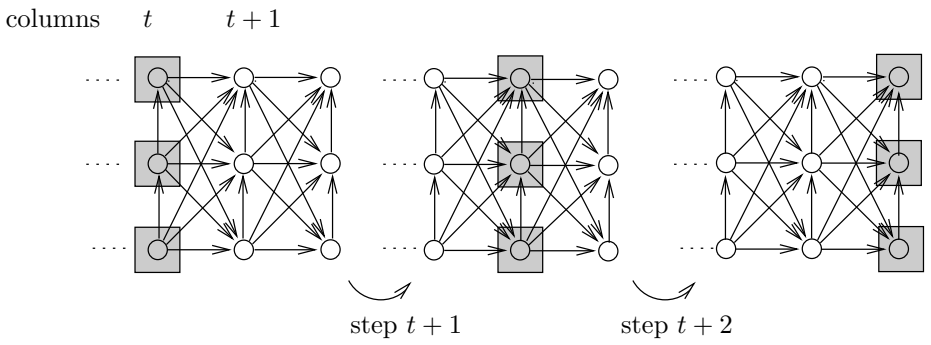


FIG. 4. *Rule $\mathcal{M}_1$. Three pebbles are needed.*

At step 0, we have three pebbles on nodes $(i, 0)$, $i = 1, 2, 3$ (rule (R1)). At step 1, it follows from rule (R3) that a pebble can be put on any positive node. For instance, let us consider node $(2, 1)$. The associated variable can be computed as follows:

$$X_2(1) = F_2\big(F_1\left(X_1(0), X_2(0), X_3(0)\right), X_1(0), X_2(0), X_3(0)\big)\,.$$

By keeping the original pebbles untouched, we can use one additional pebble to mark all the nodes one by one. In this way, we obtain a successful execution using four pebbles. It is, however, possible to do better.

Consider the following execution, illustrated in Figure 4. After step $t-1$, assume there are three pebbles on nodes $(i, t-1)$, $i = 1, 2, 3$. At step t, we can put simultaneously three pebbles on nodes (i, t), and we remove the initial pebbles (rule (R3) used three times followed by rule (R4) applied three times). At step $t+1$, we put the three pebbles on nodes $(i, t+1)$ and so on. The number of pebbles needed by this execution is three.

Game $\mathcal{M}_1$ can be seen as a model of computation of a URE where several synchronous processors are used in parallel during the computations. These processors can access the same memory locations at the same time. More precisely, this is a model of a CREW-PRAM (see, for instance, Reif [22]) computation of the URE. The number of processors needed at one step is equal to the number of moves of type (R3) (i.e., the number of computations realized). For more details, see Appendix A.2.

$\mathcal{M}_2$: Synchronous regular execution. The rules of $\mathcal{M}_2$ are obtained by restricting $\mathcal{M}_1$ as follows:

- (R1). Unchanged.
- (R2b) *(playing rule).* Same as before, with the additional restriction that configurations must be 1-periodic, i.e., $\mathcal{A}(t+1) = \mathcal{A}(t) + 1$, where

$$(i, n) \in \mathcal{A}(t) + 1 \iff (i, n-1) \in \mathcal{A}(t). \tag{6}$$

- (R3). Unchanged.
- (R4). Unchanged.

The example of Figure 4 was also verifying the set of rules $\mathcal{M}_2$. To see that $\mathcal{M}_1$ and $\mathcal{M}_2$ are indeed different, let us consider the example of Figure 5. It corresponds to the URE $X_1(n) = F_1(X_2(n-1)), X_2(n) = F_2(X_1(n))$.

In Figure 5(I), only one pebble is needed under rule $\mathcal{M}_1$. The corresponding execution verifies rule (R2) (game $\mathcal{M}_1$) but not rule (R2b) (game $\mathcal{M}_2$). In Figure 5(II),

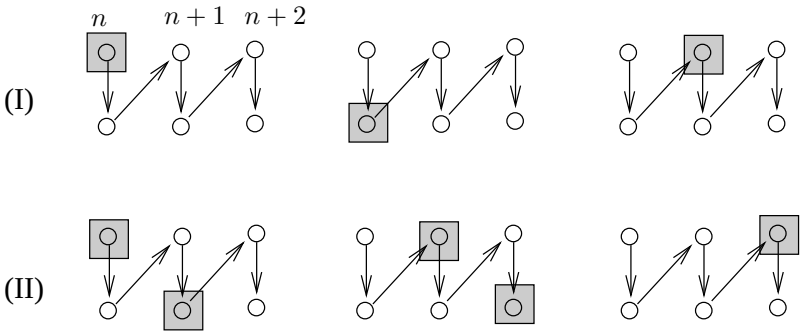


FIG. 5. *Rule* $\mathcal{M}_1$ (I) *and rule* $\mathcal{M}_2$ (II).

two pebbles are needed. The corresponding execution verifies rule (R2b). The computations are performed according to the following patterns:

(a) Rule $\mathcal{M}_1$ (Figure 5(I)).
Step t: $X_2(n) = F_2(X_1(n))$.
Step $t+1$: $X_1(n+1) = F_1(X_2(n))$.
Step $t+2$: $X_2(n+1) = F_2(X_1(n+1))\ldots$

(b) Rule $\mathcal{M}_2$ (Figure 5(II)).
Step t: $(X_1(n+1), X_2(n+2))\,(F_1 \circ F_2(X_1(n)), F_2 \circ F_1(X_2(n+1)))$.
Step t+1: $(X_1(n+2), X_2(n+3)) = (F_1 \circ F_2(X_1(n+1)), F_2 \circ F_1(X_2(n+2)))$.

Note that in the execution under rule $\mathcal{M}_2$, we have to perform the function compositions $F_2 \circ F_1$ and $F_1 \circ F_2$. In Appendix A.2, we discuss the "cost" of function compositions.

Game $\mathcal{M}_2$ corresponds to the same computational model as game $\mathcal{M}_1$, which is the CREW-PRAM model. The difference is that in an execution of $\mathcal{M}_2$, the variables in $\mathcal{A}(t)$ are obtained from those in $\mathcal{A}(t-1)$ by always applying the same operator. This is interesting for implementation purposes. A nonregular execution of $\mathcal{M}_1$ would be practically very intricate to implement since each step would be essentially different. Another advantage of an execution of $\mathcal{M}_2$ is that the number of memory locations needed to carry out the calculations is easy to compute: it is equal to $|\mathcal{A}(t)|$ (independent of t).

$\mathcal{M}_3$*: Synchronous one-pass execution.* The rules of $\mathcal{M}_3$ are obtained by restricting the ones of $\mathcal{M}_2$ as follows:

- (R1). Unchanged.
- (R2c) *(playing rule).* Same as (R2b), with the following additional restriction. Each node in $\mathcal{D}$ must be computed only once during the whole execution.
- (R3). Unchanged.
- (R4). Unchanged.

In rule (R2c), the important point is the difference that exists between computing a node and keeping the result in memory.

Let us consider the example of Figure 6(I). Each node on line 1 is computed twice, whereas each node on line 2 is computed only once. Let us detail this. Node $(1, n+2)$ is computed at step t (it is needed as an auxiliary for the computation of node $(2, n+2)$), but it is not kept in memory. Node $(1, n+2)$ is then computed a second time at step $t+1$. On the other hand, node $(2, n+2)$ is computed at step t and is kept in the memory. It does not have to be computed a second time at step $t+1$, as the computed value is just moved from one register to another.

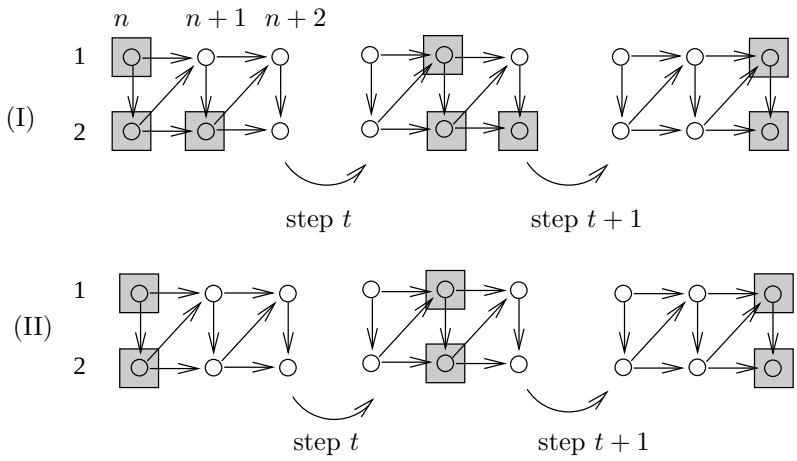


FIG. 6. *Rule* $\mathcal{M}_2$ (I) *and rule* $\mathcal{M}_3$ (II).

In Figure 6(I), we have an example of an execution satisfying rule $\mathcal{M}_2$ but not $\mathcal{M}_3$. On the other hand, in Figure 6(II), we have an execution which verifies rule $\mathcal{M}_3$. The corresponding computation pattern is as follows:

- Rule $\mathcal{M}_3$ (Figure 6(II)).
  Step t: $(X_1(n+1), X_2(n+1)) = (F_1(X_1(n), X_2(n)), F_2(F_1(X_1(n), X_2(n)), X_2(n)))$.
  Step $t+1$: $(X_1(n+2), X_2(n+2)) = (F_1(X_1(n+1), X_2(n+1)), F_2(F_1(X_1(n+1), X_2(n+1)), X_2(n+1)), X_2(n+1)))$.

The computational model corresponding to game $\mathcal{M}_3$ is still the CREW-PRAM model. Rule (R2c) may look cumbersome, but it actually corresponds to a natural notion for the applications to be detailed later on.

Notation. We will use the following notation:

- $\mathcal{E}$: the set of all possible (synchronous) executions under rule $\mathcal{M}_1$.
- $\mathcal{RE}$: the set of all possible executions under rule $\mathcal{M}_2$. Elements of $\mathcal{RE}$ will be called regular executions.
- $\mathcal{ORE}$: the set of all possible executions under rule $\mathcal{M}_3$. Elements of $\mathcal{ORE}$ will be called one-pass regular executions.

Since the rules are increasingly restrictive, we have $\mathcal{ORE} \subset \mathcal{RE} \subset \mathcal{E}$.

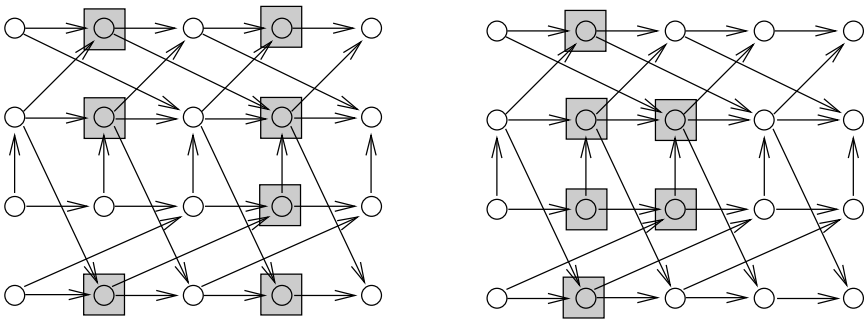
4. Cuts in the dependence graph and their relation with $\mathcal{M}_1, \mathcal{M}_2$. From now on, it is always implicitly assumed that the system under study is *recycled*; see section 2.3. In this section, we concentrate on games $\mathcal{M}_1$ and $\mathcal{M}_2$.

We introduce the notions of cuts and consecutive cuts in a dependence graph. We show that cuts (resp., consecutive cuts) are closely related to executions of the pebble game under game $\mathcal{M}_1$ (resp., game $\mathcal{M}_2$).

We show that there always exists a minimal cut which is consecutive (Lemma 4.7). It will allow us to prove Theorem 4.11, the main result of the section:

$$\min_{e \in \mathcal{RE}} \mathcal{P}(e) = \min_{e \in \mathcal{E}} \mathcal{P}(e) = \min_{C \text{ cut of } \mathcal{D}} |C| ,$$

where the notation is defined in section 3.1. As a direct consequence, we show in section 4.3 that problem MinPeb can be solved with a polynomial algorithm for games $\mathcal{M}_1$ and $\mathcal{M}_2$.



A nonconsecutive cut A consecutive cut

FIG. 7. *Consecutive and nonconsecutive cuts.*

4.1. Definitions. Let us recall some classical definitions of graph theory, all defined on the dependence graph $\mathcal{D}$. For further references, see [9, 14], for example.

DEFINITION 4.1 (path). *A path is a sequence of nodes and arcs in $\mathcal{D}$ of the form $\cdots \to (i_0, n_0) \to (i_1, n_1) \to (i_2, n_2) \to \cdots \to (i_k, n_k) \to \cdots$. A path is bi-infinite if it contains an infinite number of negative nodes and an infinite number of positive nodes.*

DEFINITION 4.2 (cut). *A cut C is a set of nodes in $\mathcal{D}$ such that any bi-infinite path contains at least one node of C. A cut with a minimal number of nodes is called a minimal cut.*

DEFINITION 4.3 (flow). *A flow is a set of bi-infinite paths such that any two paths do not share any node. A flow containing a maximal number of paths is called a maximal flow. A flow $\mathcal{F}$ is 1-periodic if the following holds: the arc $(i, n) \to (j, m)$ belongs to $\mathcal{F}$ if and only if $(i, n+1) \to (j, m+1)$ belongs to $\mathcal{F}$.*

The most classical notion of cut involves arcs rather than nodes, and a flow is a set of paths which do not share arcs rather than nodes. However, a simple transformation—each node being replaced by two nodes connected by an arc—would allow us to go back to the original definitions.

DEFINITION 4.4 (section). *A section S in $\mathcal{D}$ is a set of nodes with exactly one node per line, $S = \{(i, n_i), i \in V\}$.*

Note that since $\mathcal{D}$ is recycled, a cut contains at least one node per line. Using this property, one can define the left and right sections of a cut.

DEFINITION 4.5 (left, right sections). *The left (resp., right) section C_w, with w for* west *(resp., C_e, with e for* east*), of a finite cut C is the set of nodes (i, n) in C such that the nodes $(i, n-h), h > 0$ (resp., $(i, n+h), h > 0$), do not belong to C.*

DEFINITION 4.6 (consecutive cut). *A cut C in $\mathcal{D}$ is consecutive if on each line of $\mathcal{D}$, C contains only consecutive nodes, i.e.,*

$$\forall i \in V,\ (i, n) \in C \text{ and } (i, n+1) \notin C \Rightarrow (i, n+k) \notin C \ \ \forall k > 0.$$

Examples of consecutive and nonconsecutive cuts are displayed in Figure 7.

LEMMA 4.7. *There exists a minimal cut of $\mathcal{D}$ which is a minimal consecutive cut.*

Proof. Let C be a minimal consecutive cut. We will prove that C is a minimal cut. First, we prove that there are no arcs from C_w to $C_e + k, k \geq 2$ (where $(i, n) \in C_e + k$ if and only if $(i, n-k) \in C_e$). Let us assume that there exists such an arc, which we denote by $(i, n) \to (j, m)$. By 1-periodicity, there is an arc between nodes $(i, n-1)$

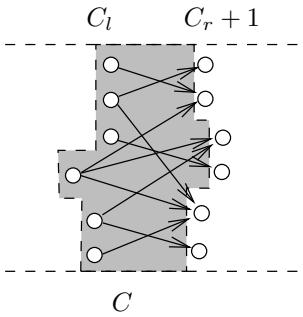


FIG. 8. *Graph G made from the right and left sections of C.*

and $(j, m-1)$. Now consider the bi-infinite path

$$\cdots \to (i, n-3) \to (i, n-2) \to (i, n-1) \to (j, m-1) \to (j, m) \to (j, m+1) \to \cdots .$$

It does not intersect C, which is a contradiction.

We consider the subgraph G of $\mathcal{D}$ made of the nodes $C_w \cup (C_e + 1)$ and the arcs between C_w and $C_e + 1$ in $\mathcal{D}$; see Figure 8. We recall that a cut in a *finite* graph G is a set of nodes such that, when removed from G, there is no arc remaining. The set C_w is a cut in G. Let Δ be a cut in G of minimal size. We have $|\Delta| \leq |C_w|$. If $|\Delta| < |C_w|$, then $(C \backslash C_w) \cup \Delta$ would be a consecutive cut in $\mathcal{D}$ strictly smaller than C, which would contradict the fact that C is a minimal consecutive cut. Therefore, we have $|\Delta| = |C_w|$.

An adapted version of a famous "minimax" theorem first proved by König (1931) states that we can find $|\Delta|$ node-disjoint arcs in G. Since $|\Delta| = |C_w| = |C_e + 1|$, these arcs define a one-to-one mapping ϕ from C_w to $C_e + 1$. From ϕ, we construct a flow in C in the following way. Select all the arcs of the form $((i, n) + k) \to (\phi(i, n) + k)$ $\forall (i, n) \in C_w$ and $\forall k \in \mathbb{Z}$. These arcs form a 1-periodic flow $\mathcal{F}$ in $\mathcal{D}$ of size $|C|$.

Let C_m be a minimal cut in $\mathcal{D}$. Since $\mathcal{F}$ is formed by node-disjoint paths, C_m must contain at least $|\mathcal{F}|$ nodes, $|C_m| \geq |\mathcal{F}| = |C|$. We conclude that $|C_m| = |C|$. ∎

This lemma is interesting on its own. In particular, it gives a proof of the minimax theorem (which exists in many versions) for an infinite 1-periodic and recycled graph.

COROLLARY 4.8. *The size of the minimal cut is equal to the size of the maximal flow in $\mathcal{D}$. Furthermore, there exists a maximal flow $\mathcal{F}$ in $\mathcal{D}$ which is 1-periodic.*

4.2. Cuts and pebbles.

LEMMA 4.9. *Let C be a finite consecutive cut. There is a regular execution $e \in \mathcal{RE}$ such that C is a configuration of e.*

Proof. Let C be a consecutive cut in $\mathcal{D}$. We want to prove that it is possible to have $\mathcal{A}(t) = C$ and $\mathcal{A}(t+1) = C + 1$ (note that $C + 1 = (C \backslash C_w) \cup (C_e + 1)$). It is enough to prove that for each node (i, n) in $C_e + 1$, there is no infinite path P terminating in (i, n) that does not intersect the cut. But if such a path could be found, then the bi-infinite path $P \cup \{(i, n+h), h \in \mathbb{N}\}$ would not intersect C. It would contradict the fact that C is a cut. ∎

The converse of Lemma 4.9 is not true: a configuration of a regular execution need not be a consecutive cut. This is illustrated in Figure 9(a). Also note that there are nonconsecutive cuts which are not configurations of a regular execution, as

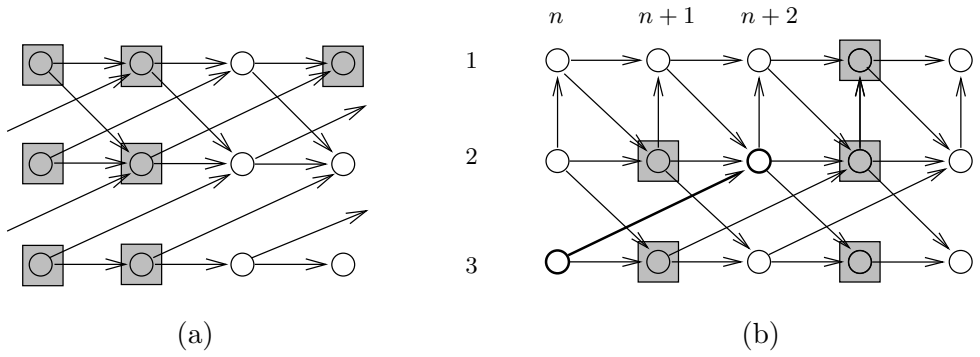


FIG. 9. *Two counterexamples.*

illustrated in Figure 9(b). In this example, the node $(2, n+2)$ belongs to $C+1$ but cannot be computed using only variables in C (as it depends on $(3, n)$, for example). Therefore, the cut C cannot belong to a regular execution.

LEMMA 4.10. *A configuration of any execution $e \in \mathcal{E}$ is a finite cut in $\mathcal{D}$. Conversely, let C be a finite cut in $\mathcal{D}$. There is an execution $e \in \mathcal{E}$ such that C is a configuration of e.*

Proof. Let $\mathcal{A}(t)$ be the tth configuration of some execution e belonging to $\mathcal{E}$. All the configurations of e are finite by definition. Therefore the total number of nodes that received a pebble up to step t is finite.

Now, assume that $\mathcal{A}(t)$ is not a cut. By definition, there exists a bi-infinite path P which does not have any node in $\mathcal{A}(t)$. According to rule (R3) of game $\mathcal{M}_1$, no node on P will receive a pebble during the execution, after step t. Combining this and the fact that the total number of nodes that received a pebble up to step t is finite, only a finite number of positive nodes on P receive a pebble during the execution e. This contradicts the fact that e has to put pebbles on all nodes.

Let us prove the converse result. Let C be a finite cut. Let $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ be any regular execution of game $\mathcal{M}_1$. Such executions exist (see Lemma 4.9). Let $\mathcal{N}$ be the set of positive nodes (i, n) such that there exists an infinite path ending in (i, n) and which does not intersect C. As C is a cut, $\mathcal{N}$ is finite. Let $T = \sup\{t \mid (\mathcal{N} \cup C) \cap \mathcal{A}(t) \neq \emptyset\}$. Note that T is finite since $\{\mathcal{A}(t)\}$ is regular and $\mathcal{N}$ is finite. We define $\tilde{e} = \{\tilde{\mathcal{A}}(t)\}$ as follows:

$$\tilde{\mathcal{A}}(t) = \begin{cases} \bigcup_{n=0}^{t} \mathcal{A}(n) & \text{if } t \leq T, \\ C & \text{if } t = T+1, \\ \mathcal{A}(t-1) & \text{if } t > T+1\,. \end{cases}$$

Let us show that $\tilde{e}$ is an execution of $\mathcal{M}_1$. We have $C \subset \bigcup_{n=0}^{T} \mathcal{A}(n)$; therefore, it is possible to set $\tilde{\mathcal{A}}(T+1) = C$. By definition of T, $\mathcal{A}(T+1)$ does not intersect $\mathcal{N}$; therefore, we can set $\tilde{\mathcal{A}}(T+2) = \mathcal{A}(T+1)$. Finally, $\tilde{e}$ contains all nodes in $\{\mathcal{A}(t), t \in \mathbb{N}\}$ and therefore all positive nodes. ∎

We are now ready to give the main result of this section, which states that, within all executions in $\mathcal{E}$, regular executions are dominant for problem MinPeb.

THEOREM 4.11. *Let us consider a recycled system of UREs. We play the pebble game on its associated dependence graph $\mathcal{D}$ under rules $\mathcal{M}_1$ and $\mathcal{M}_2$. We have*

$$\min_{e \in \mathcal{RE}} \mathcal{P}(e) = \min_{e \in \mathcal{E}} \mathcal{P}(e) = \min_{C \text{ cut of } \mathcal{D}} |C|,$$

that is, there exists a regular execution which requires a minimal number of pebbles, this number being equal to the size of a minimal cut.

Proof. It is a direct consequence of Lemmas 4.10, 4.9, and 4.7. First, note that all configurations are cuts, according to Lemma 4.10. Let C be a consecutive cut of minimal size, which exists by Lemma 4.7. By Lemma 4.9, C is a configuration of a regular execution. ∎

Theorem 4.11 has several interesting corollaries. First, it allows one to focus on regular executions since no fancy irregular execution of the URE can be done with fewer pebbles. Then it provides a polynomial method to find an optimal execution as shown in section 4.3.

4.3. Complexity results for $\mathcal{M}_1$ and $\mathcal{M}_2$.

PROPOSITION 4.12. *Let $\mathcal{R} = (V, E, \Gamma)$ be the reduced graph associated with a recycled system of UREs with nonnegative delays. We set $\Gamma_A = \sum_{e\in E} \Gamma(e)$. For games $\mathcal{M}_1$ and $\mathcal{M}_2$, problem* MinPeb *can be solved using an algorithm having a complexity $O\left(\Gamma_A^2 |V|^2\right)$.*

If the system of UREs has negative delays, it is possible to go back to nonnegative delays (see Remark 2.1) and apply Proposition 4.12 to the new system.

In order to prove Proposition 4.12, we are going to compute a maximal flow in $\mathcal{D}$ and then apply Corollary 4.8. If we want to use the algorithm of Ford and Fulkerson [9] to compute a maximal flow, we first need to restrict ourselves to a finite graph.

The *span* of a cut is the difference between the largest and the smallest of the numberings of columns containing a node of the cut.

A *slice* of $\mathcal{D}$ of dimension n is defined as the subgraph of $\mathcal{D}$ having nodes $\{(i,k), i \in V, 0 \le k \le n\} \cup \{T, B\}$, where T (top) and B (bottom) are two special nodes. There is an arc $T \to (i,k),\ 0 \le k \le n$, if $\exists (j,l), l < 0$, such that there is an arc $(j,l) \to (i,k)$ in $\mathcal{D}$. There is an arc $(i,k) \to B,\ 0 \le k \le n$, if $\exists (j,l), l > n$, such that there is an arc $(i,k) \to (j,l)$ in $\mathcal{D}$.

If a consecutive minimal cut spans less than n columns, then $\mathcal{D}$ and a slice of dimension n have the same minimal consecutive cut (the special nodes T and B are not allowed to belong to the cut). So it is important to determine, or at least to bound, the span of a consecutive minimal cut.

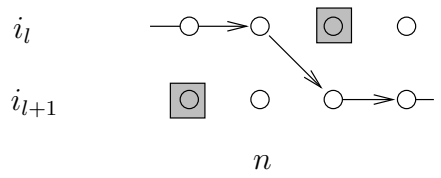
LEMMA 4.13. *If all the delays are nonnegative, the span of a minimal consecutive cut is smaller than the total sum of the delays in $\mathcal{R}$, i.e., smaller than $\Gamma_A = \sum_{e\in E} \Gamma(e)$.*

Proof. Let C be a minimal consecutive cut, and let $\mathcal{F}$ be a maximal 1-periodic flow; see Corollary 4.8.

The associated maximal 1-periodic flow (see Corollary 4.8) $\mathcal{F}$ is a set of paths in $\mathcal{D}$. First, these paths cover all the nodes in $\mathcal{D}$. Indeed, by the 1-periodicity of $\mathcal{F}$, if a node (i,n) is not in $\mathcal{F}$, then the whole line $(i,.)$ is not in $\mathcal{F}$, but this means that the bi-infinite path $\{(i,n), n \in \mathbb{Z}\}$ can be added to the flow $\mathcal{F}$, and this contradicts the maximality of $\mathcal{F}$.

Let P_1 be any path in $\mathcal{F}$. It follows from the 1-periodicity of $\mathcal{F}$ that P_1 is periodic. Let $i_0, i_1, \dots, i_{l_1}, i_0, i_1, \dots$ be the successive lines visited by the path P_1. Let (i_0, n) and $(i_0, n + k_1)$ be the consecutive nodes visited by the path P_1 on line $(i_0, .)$. Using the 1-periodicity of $\mathcal{F}$, the total number of paths intersecting lines $i_0, i_1, \dots, i_{l_1}$ in $\mathcal{F}$ is k_1. It implies that the cardinal of C over the lines $i_0, i_1, \dots, i_{l_1}$ is exactly k_1 (Corollary 4.8). Assume that the span of C on lines $i_0, i_1, \dots, i_{l_1}$ is strictly greater than k_1. Then there exists a column, say, n, not intersecting C and such that on some of the lines $\{i_0, i_1, \dots, i_{l_1}\}$, C is on the "right" of column n and on some others C is on the "left"

of column n. Let l be such that C is on the "right" of n at line i_l and on the "left" at line i_{l+1}. There exists an arc of the type $(i_l, n) \to (i_{l+1}, n+h), h \geq 0$ (the delays are



nonnegative) in the flow $\mathcal{F}$. Then the path $\{(i_l, n-u), (i_{l+1}, n+h+v), u, v \in \mathbb{N}\}$ does not intersect the cut C; see the above figure. This is a contradiction. We conclude that the span of C over the lines $i_0, i_1, \ldots, i_{l_1}$ is smaller than k_1. By definition of $\mathcal{R}$, there exists a circuit L_1 in $\mathcal{R}$ containing the nodes $i_0, i_1, \ldots, i_{l_1}$ and of total delay k_1.

The path P_1 and all its shifts are in the flow $\mathcal{F}$ and contain all the nodes in the lines $i_0, i_1, \ldots, i_{l_1}$. If $i_0, i_1, \ldots, i_{l_1}$ do not cover all the lines, a new path P_2 in $\mathcal{F}$ not intersecting the lines $i_0, i_1, \ldots, i_{l_1}$ ranges over different lines, say, $i_{l_1+1}, i_{l_1+2}, \ldots, \ldots, i_{l_2}$, and defines a circuit L_2 in $\mathcal{R}$ similarly. The span of C on the lines $i_{l_1+1}, \ldots, i_{l_2}$ is smaller than k_2, the total delay of circuit L_2. We apply the same argument until all lines in $\mathcal{D}$ are covered. This defines a set H of circuits partitioning the nodes of $\mathcal{R}$.

We build a new multigraph $\mathcal{G}$ starting with $\mathcal{R}$ and where each circuit in H is merged into one single node. The graph $\mathcal{G}$ has $|H|$ nodes and the arcs of $\mathcal{G}$ correspond to the arcs of $\mathcal{R}$ which do not belong to any circuit in H. Considering two nodes in $\mathcal{G}$, say L_1 and L_2, the span of C over lines $i_0, \ldots, i_{l_2}$ can be chosen to be smaller than $k_1 + k_2 + d$, where d is the maximum delay on all arcs between the nodes L_1 and L_2 in $\mathcal{G}$. Overall, the cut C can be chosen to have a span which is smaller than the sum of the delays on all the circuits in H plus the sum of the delays on all the arcs in $\mathcal{G}$. No delay is counted twice in this upper bound. Therefore, the total span of C is smaller than the total sum of the delays in $\mathcal{R}$. □

Proof of Proposition 4.12. A slice of $\mathcal{D}$ of size Γ_A has the same minimal consecutive cut as $\mathcal{D}$ itself. The computation of the maximal flow in a finite slice can be done using the augmenting path algorithm; see [9, 14]. Starting with a 1-periodic flow (the recycled lines) and maintaining the 1-periodicity throughout the construction yields a maximal 1-periodic flow. The complexity of this construction of the maximal flow is $O(\Gamma_A^2 |V|^2)$. By Corollary 4.8, it provides the size of a minimal cut in $\mathcal{D}$. Furthermore, a standard procedure provides a minimal consecutive cut starting from a maximal flow (with a complexity $O(\Gamma_A |V|)$). Using Lemma 4.9, an execution of game $\mathcal{M}_2$ (or $\mathcal{M}_1$), using a minimal number of pebbles, is obtained from the minimal consecutive cut.

For the game $\mathcal{M}_3$, the problem MinPeb is solved by working on the reduced graph. A polynomial algorithm is given in section 6.5.

5. Compatible cuts and their relation with $\mathcal{M}_3$. We introduce the notion of compatible cuts. It enables us to show Theorem 5.5, the main result of the section, which is the analogue of Theorem 4.11:

$$\min_{e \in \mathcal{ORE}} \mathcal{P}(e) = \min \{|C|, \ C \text{ compatible cut of } \mathcal{D}\}.$$

Let us introduce some new definitions.

DEFINITION 5.1 (crossings). *We say that an arc crosses a section* $S = \{(i, n_i), i \in V\}$ from left to right *if it is an arc of the form* $(i, n_i - h) \to (j, n_j + l)$ *with* $h \geq 0$ *and* $l \geq 1$. *An arc* $(i, n_i + l) \to (j, n_j - h)$ *crosses* S from right to left *if* $l \geq 1$ *and* $h \geq 0$.

DEFINITION 5.2 (compatible section, compatible cut). *A section in $\mathcal{D}$ is* compatible *if no arc crosses the section from right to left. A consecutive cut is said to be* compatible *if its right section is compatible.*

Roughly speaking a compatible cut is a consecutive cut which agrees with the dependence relations in the system of UREs. Compatible cuts are connected to one-pass executions through the next two lemmas.

LEMMA 5.3. *Let C be a compatible cut. There is a one-pass regular execution $e \in \mathcal{ORE}$ such that C is a configuration of e.*

Proof. Let C be a compatible cut. Since C is consecutive by definition, Lemma 4.9 tells us that C is a configuration of a regular execution which can be written as $e = \{C + t, t \in \mathbb{N}\}$. Suppose that e is not one-pass. This means that there exists a node, say, (j, m), that is computed twice in e.

Let us assume that node (j, m) receives a pebble at step t_0 and that this pebble is removed at step t_1, $t_1 > t_0$. By regularity of the execution e, node (j, m) will not be used at step $t > t_1 + 1$. Indeed, a node in $C + t$ depends only on variables in $C + t - 1$ or "below."

Now assume that node (j, m) is used at step $t < t_0$ to compute another node, say, (i, n). The path $(j, m) \to (i, n)$ crosses the right section of $C + t$ from right to left. Therefore it contains an arc that crosses the right section of $C + t$ from right to left. This contradicts the fact that C is compatible. ∎

LEMMA 5.4. *The configuration of a one-pass regular execution is a compatible cut.*

Proof. Let $e = \{C + t, t \in \mathbb{N}\}$ be a one-pass regular execution. First, C is a cut by Lemma 4.10. Next, C is consecutive. Indeed, if C is not consecutive on line i, then each variable $X_i(n)$ receives a pebble at least twice in e, and a fortiori this means it is computed at least twice.

It remains to show that C is compatible. Assume that C is not compatible. Then there exists a node (i, n) belonging to the right section of $C + t$, a positive integer k, and a node $(j, m) \in (C + t + k) \backslash (C + t)$ such that the arc $(j, m) \to (i, n)$ belongs to $\mathcal{D}$. But this implies that in the execution e, the computation of $X_j(m)$ is performed twice, once as an auxiliary computation at step t to compute $X_i(n)$ and once at step $t + k$. This contradicts the fact that e is one-pass. ∎

THEOREM 5.5. *Let us consider a recycled system of UREs. We play the pebble game on its associated dependence graph $\mathcal{D}$ under rules $\mathcal{M}_3$. We have*

$$\min_{e \in \mathcal{ORE}} \mathcal{P}(e) = \min \{|C|, \ C \text{ compatible cut of } \mathcal{D}\}.$$

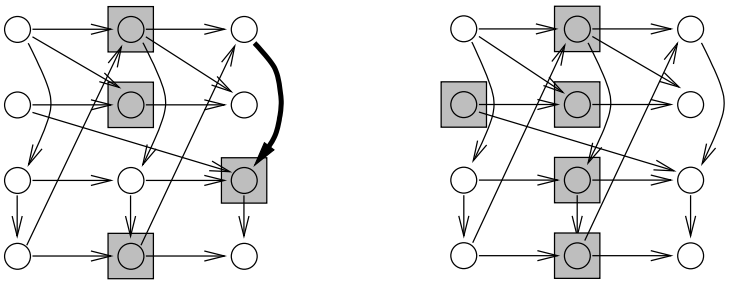
Proof. The proof is an immediate corollary of Lemmas 5.3 and 5.4. ∎

It can be that no minimal consecutive cut in $\mathcal{D}$ is compatible. This is the case in Figure 10, where the minimal compatible cut contains 5 nodes while there is a minimal consecutive cut of size 4.

Therefore, rule $\mathcal{M}_3$ requires more memory in general than rule $\mathcal{M}_2$.

6. Delays in the reduced graph. In the previous sections, we have investigated the relations between executions of a system of UREs and cuts in the dependence graph. In this section, we connect these two notions with values of the delays in the reduced graph.

Let us consider a system of URE $\mathcal{S}$ with variables $\{X_i(n), i \in V, n \in \mathbb{Z}\}$ and a regular execution $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ of the system. We introduce the modified system $\tilde{\mathcal{S}}$ with variables $\{\tilde{X}_i(n), i \in V, n \in \mathbb{Z}\}$ and the execution $\tilde{e} = \{\tilde{\mathcal{A}}(t), t \in \mathbb{N}\}$ defined as



Minimal (noncompatible) cut Minimal compatible cut

FIG. 10. *Noncompatible and compatible cuts.*

follows:

$$\begin{aligned} c_i &= \max\{n \in \mathbb{Z}^- \mid (i,n) \in \mathcal{A}(0)\}, \\ \tilde{X}_i(n) &= X_i(n + c_i), \\ \tilde{\mathcal{A}}(t) &= \{(i,n) \text{ s.t. } (i, n + c_i) \in \mathcal{A}(t)\}. \end{aligned}$$

The above definition is such that the right section of $\tilde{\mathcal{A}}(0)$ is $S_0 = \{(i,0), i \in V\}$. Viewed on the dependence graphs, the passage from $\mathcal{S}$ to $\tilde{\mathcal{S}}$ corresponds to a shift of the lines. Viewed on the reduced graphs, it corresponds to a *retiming*, i.e., a modification of the value of the delays, while preserving the graph topology.

We show (Lemma 6.6 and 6.7) that the total number of delays in $\tilde{\mathcal{R}}$ is closely related to regular executions. As a consequence, we obtain a polynomial algorithm to solve problem MinPeb under rule $\mathcal{M}_3$; see section 6.5.

6.1. Retiming.

DEFINITION 6.1 (retiming). *Let $\mathcal{R}$ be the reduced graph of a system of UREs. A* retiming *of $\mathcal{R}$ is a node function $r : V \to \mathbb{Z}$ which specifies a new graph $\mathcal{R}_r$, with the same nodes and arcs as $\mathcal{R}$. The value of the delay on an arc $e = (i,j)$ in $\mathcal{R}_r$ is equal to $\Gamma_r(e) = \Gamma(e) + r(i) - r(j)$.*

The notion of retiming is classic in digital circuits (see [17] and section 7, where we provide a detailed discussion of its usefulness in this context) and in Petri nets, where it corresponds to the firing of transitions (see section 8).

In the example of Figure 11, the new values of the delays correspond to a retiming r such that $r(1) = 1, r(2) = 1,$ and $r(3) = 0$.

Retiming may create negative delays, as in Figure 11.

LEMMA 6.2. *Two retimings r and r' yield the same value of the delays in a connected graph $\mathcal{R}$ if and only if there exists a constant $h \in \mathbb{Z}$ such that $\forall i \in V, r(i) = r'(i) + h$.*

Proof. First, if $r(i) = r'(i) + h \ \forall i \in V$, then on any arc $e = (i,j)$, $\Gamma_r(e) = \Gamma(e) + r(i) - r(j) = \Gamma(e) + r'(i) - r'(j) = \Gamma_{r'}(e)$. Conversely, if $\Gamma_{r'}(e) = \Gamma_r(e)$, then $r(i) = r'(i) + h$ and $r(j) = r'(j) + h$ for some $h \in \mathbb{Z}$. As $\mathcal{R}$ is connected, the constant h is the same for all the nodes in V. □

The question that arises now is, What is the notion corresponding to retiming in the system of UREs $\mathcal{S}$ and in the dependence graph $\mathcal{D}$? To answer this question, let us consider the graph $\mathcal{D}_r$ associated with the retimed reduced graph $\mathcal{R}_r$. This dependence graph can be constructed directly from $\mathcal{D}$ by shifting the lines as described in Lemma 6.3.

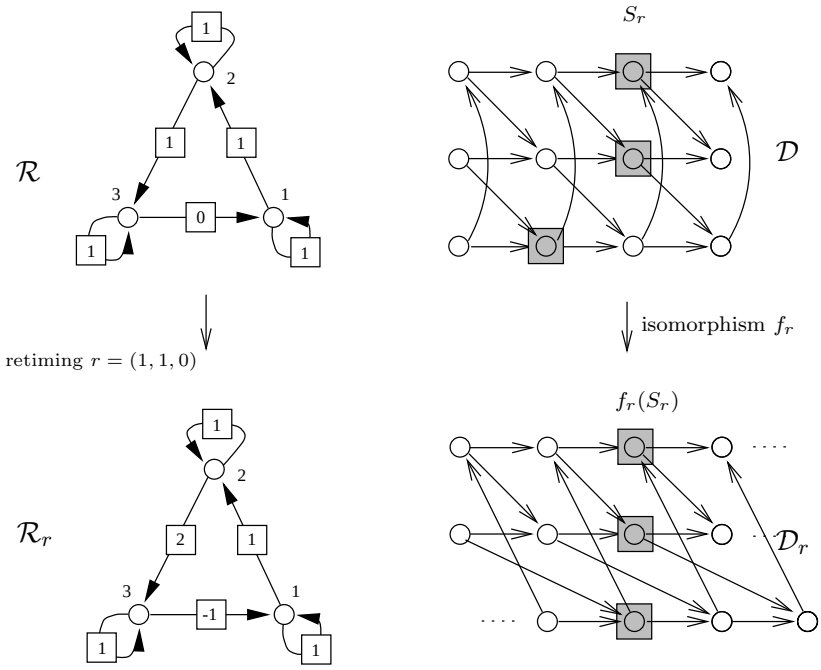


FIG. 11. *Retimed reduced graph and dependence graph.*

LEMMA 6.3. *A retiming r in $\mathcal{R}$ corresponds to a transformation f_r between $\mathcal{D}$ and $\mathcal{D}_r$ defined by*

$$\begin{array}{rccc} f_r: & \mathcal{D} & \to & \mathcal{D}_r \\ & (i,n) & \to & (i, n - r(i)). \end{array}$$

The transformation f_r is an isomorphism of graphs, meaning that there is an arc between u and v in $\mathcal{D}$ if and only if there is one between $f_r(u)$ and $f_r(v)$ in $\mathcal{D}_r$. It will also be called a retiming of $\mathcal{D}$.

Proof. By definition of $\mathcal{D}$, there is an arc from (i,n) to (j,m) in $\mathcal{D}$ if the delay in $\mathcal{R}$ on arc (i,j) is $\gamma = m - n$. The delay in $\mathcal{R}_r$ on arc (i,j) is $\gamma_r = \gamma + r(i) - r(j) = (m - r(j)) - (n - r(i))$. It implies that there is an arc between $(i, n - r(i))$ and $(j, n - r(j))$ in $\mathcal{D}_r$. Therefore, f_r is an isomorphism between $\mathcal{D}$ and $\mathcal{D}_r$. ∎

We recall that the notion of *section* was introduced in Definition 4.4. We associate with a retiming r in $\mathcal{R}$ the section $S_r = \{(i, r(i)), i \in V\}$ in $\mathcal{D}$.

Lemmas 6.2 and 6.3 tell us that two retimings r and r' are similar (in the sense that they yield the same value of the delays) if and only if they are associated with two sections S_r and S'_r with $S_r = S_{r'} + h$ for some $h \in \mathbb{Z}$. This relation enables us to define an equivalence relation between sections in $\mathcal{D}$ as well as between retimings in $\mathcal{R}$. We say that section S_r (resp., retiming r) is equivalent to section $S_{r'}$ (resp., retiming r') if $S_r = S_{r'} + h$ for some $h \in \mathbb{Z}$. In the following, we will always consider one arbitrary section among the equivalence class and call it the section associated with the retiming r; for instance, S_0 corresponds to the equivalence class of $\{(i,0), i \in V\}$.

6.2. Counting the delays. Given a graph $\mathcal{R} = (V, E, \Gamma)$, we define the *total number of delays* of $\mathcal{R}$ as follows:

$$\Gamma_A(\mathcal{R}) = \sum_{i \in V} \sum_{(j,\gamma) \in \Delta_i} \gamma \,. \tag{7}$$

It corresponds to the number of delays appearing in the graphical representation of the reduced graph $\mathcal{R}$ as defined in section 2.2. See, for example, the graph $\mathcal{R}$ on Figure 12.

Given a graph $\mathcal{R} = (V, E, \Gamma)$, another quantity of interest is the following:

$$\Gamma_B(\mathcal{R}) = \sum_{j \in V} \max\{\gamma \mid \exists i \text{ s.t. } (j,\gamma) \in \Delta_i\} \,. \tag{8}$$

When the delays are positive ($\forall e \in E, \Gamma(e) \geq 0$), Γ_B corresponds to the total number of delays (Γ_A) in a modified reduced graph obtained by performing a forward splitting of the nodes. In the context of digital circuits, this is also called register sharing; see section 7.2. Given under the form of an algorithm, here is the formal construction of the `Forward Splitting` algorithm.

ALGORITHM 6.4 (Forward Splitting).

Input: *Reduced graph $\mathcal{R} = (V, E, \Gamma)$ with $\Gamma \geq 0$, functions associated with the nodes $\{F_i, i \in V\}$.*

1. Set $V' = V$ and $E' = \emptyset$. Associated functions $F'_i = F_i, i \in V$.
2. For all node $v \in V$, let δ be the maximum delay on all the output arcs of v.
   - Set $v_0 = v$.
   - If $\delta > 0$, create δ new nodes in V', called $v_1, \ldots, v_\delta$. Set $F'_{v_i} = Id$, the identity function, for $i = 1, \ldots, \delta$.
   - For each arc $e = (v, u) \in E$ with delay $\Gamma(e) = \gamma$, create an arc $e' = (v_\gamma, u)$ in E' with delay $\Gamma'(e') = 0$.
   - Add the arcs (v_i, v_{i+1}), $0 \leq i \leq \delta - 1$, in E', with delay $\Gamma' = 1$.

Output: *Split reduced graph $\mathcal{R}' = (V', E', \Gamma')$ with $\Gamma' \geq 0$, functions associated with the nodes $\{F'_i, i \in V\}$.*

It is important to remark that the split graph $\mathcal{R}'$ is not necessarily recycled, as opposed to $\mathcal{R}$. We will come back to this point in section 6.4.

The following proposition, easy to prove, justifies the `Forward Splitting` operation.

PROPOSITION 6.5. *Let $\mathcal{R}$ be a reduced graph with positive delays and $\mathcal{R}'$ the associated split graph.*

(i) *The associated systems of UREs $\mathcal{S}$ and $\mathcal{S}'$ have the same behavior. More precisely, borrowing the notation of the algorithm, we have*

$$\forall v \in V, n \in \mathbb{Z},\ X'_v(n) = X_v(n) \quad \textit{and} \quad \forall v_i \in V' \setminus V, n \in \mathbb{Z},\ X'_{v_i}(n) = X_v(n - i) \,.$$

(ii) *Furthermore, we have*

$$\Gamma_B(\mathcal{R}) = \Gamma_A(\mathcal{R}') = \Gamma_B(\mathcal{R}') \,. \tag{9}$$

Note that in order for (9) to make sense, it is necessary to extend the definitions of Γ_A and Γ_B to nonrecycled graphs.

In Figure 12, $\mathcal{R}'$ is the `Forward Splitting` of $\mathcal{R}$. In the split graph, there are two "dummy" nodes (associated with the identity function), represented by black dots. We have $\Gamma_A(\mathcal{R}) = 8$ and $\Gamma_B(\mathcal{R}) = \Gamma_A(\mathcal{R}') = \Gamma_B(\mathcal{R}') = 5$.

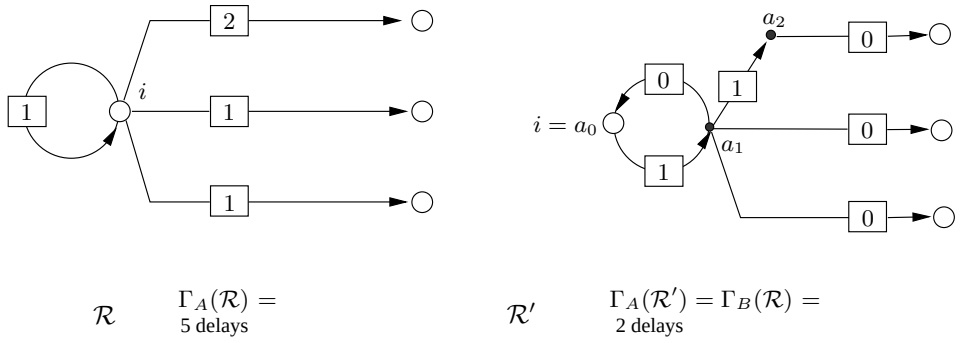


FIG. 12. *A reduced graph and the associated split graph.*

6.3. Delays, cuts, and pebbles. Given a section $S = \{(i, n_i), i \in V\}$ in $\mathcal{D}$, we define the cut $\mathcal{C}(S)$ in the following way:

$$\mathcal{C}(S) \stackrel{\text{def}}{=} \{(i,n), i \in V, n \leq n_i \mid \exists j \in V, m > n_j, (i,n) \to (j,m)\} . \tag{10}$$

The cut $\mathcal{C}(S)$ is consecutive and its right section is S. Furthermore, if any node is removed from the left section of $\mathcal{C}(S)$, then it is not a cut anymore.

In a cut C, a node $(i,n) \in C$ is *redundant* if $C \backslash \{(i,n)\}$ is a cut. Any consecutive cut C with no redundant node on its left section is characterized by its right section $\underline{S}$ only. More precisely, it verifies $C = \mathcal{C}(\underline{S})$.

We are now ready to state the relations between delays in $\mathcal{R}$ and consecutive cuts in $\mathcal{D}$.

LEMMA 6.6. (i) *The number of delays $\Gamma_B(\mathcal{R})$ is equal to the cardinal of $\mathcal{C}(S_0)$.* (ii) *Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. Then the number of delays $\Gamma_B(\mathcal{R}_r)$ is equal to the cardinal of the cut $\mathcal{C}(S_r)$ in $\mathcal{D}$ (resp., $\mathcal{C}(S_0)$ in $\mathcal{D}_r$).*

Proof. The isomorphism f_r transforms the section $\mathcal{S}_r$ in $\mathcal{D}$ into the section $\mathcal{S}_0$ in $\mathcal{D}_r$. Hence (ii) is implied by (i). Let us work on graph $\mathcal{D}$. We are going to prove that $\Gamma_B(\mathcal{R}) = |\mathcal{C}(\mathcal{S}_0)|$. We consider a node i of $\mathcal{R}$. Let $m = \max\{\gamma \mid \exists j,\ (i,\gamma) \in \Delta_j\}$ (we have $m \geq 1$ as $(i,1) \in \Delta_i$) and let j be such that $(i,m) \in \Delta_j$. There is an arc in $\mathcal{D}$ from $(i,-m)$ to $(j,0)$ and no arc from a node on the "left" of $(i,-m)$ (and on line i) to a node on the "right" of S_0. Hence $\mathcal{C}(S_0)$ contains the nodes $(i,-m+1), \ldots, (i,0)$ on line i. The same argument repeated on each line finishes the proof. □

We recall that the arcs crossing a section *from right to left* and *from left to right* are defined in Definition 5.1.

LEMMA 6.7. (i) *The number of delays $\Gamma_A(\mathcal{R})$ is equal to the number of arcs crossing S_0 from left to right minus the number of arcs crossing it from right to left.* (ii) *Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. The number of delays $\Gamma_A(\mathcal{R}_r)$ is equal to the number of arcs in $\mathcal{D}$ (resp., $\mathcal{D}_r$) crossing section S_r (resp., S_0) from left to right minus the number of arcs crossing S_r (resp., S_0) from right to left.*

Proof. For the same reason as in Lemma 6.6, it is enough to prove (i). Let (i,j) be an arc of $\mathcal{R}$ with delay $\gamma \geq 0$. In $\mathcal{D}$, this arc induces exactly γ arcs crossing S_0 from left to right, the arcs

$$(i,-\gamma+1) \to (j,1) \quad \cdots \quad (i,-1) \to (j,\gamma-1), \quad (i,0) \to (j,\gamma) .$$

Similarly, an arc (i,j) with delay $\gamma < 0$ induces exactly $-\gamma$ arcs crossing S_0 from right

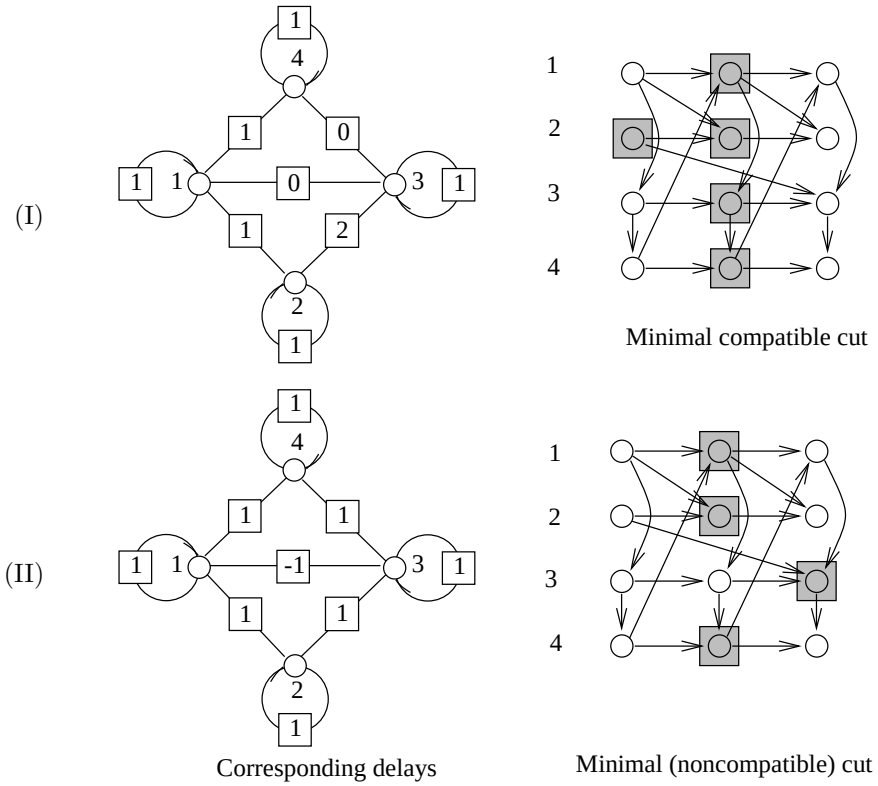


FIG. 13. *Compatible and noncompatible cuts, nonnegative and negative delays.*

to left:

$$(i,1) \to (j,\gamma+1) \quad \cdots \quad (i,-\gamma-1) \to (j,-1), \quad (i,-\gamma) \to (j,0)\,.$$

The same argument applied to all the lines finishes the proof. □

The notion of compatible cuts introduced in Definition 5.2 has a very natural interpretation in terms of delays.

LEMMA 6.8. *Let r be a retiming of $\mathcal{R}$ and S_r an associated section in $\mathcal{D}$. The retimed reduced graph $\mathcal{R}_r$ has only nonnegative delays if and only if the cut $\mathcal{C}(S_r)$ is compatible in $\mathcal{D}$ (equivalently, the cut $\mathcal{C}(S_0)$ is compatible in $\mathcal{D}_r$).*

Proof. If $\mathcal{R}_r$ has only nonnegative delays, the argument used in the proof of Lemma 6.7 shows that all the arcs crossing S_r in $\mathcal{D}$ cross it from left to right. It implies that the cut $\mathcal{C}(S_r)$ is compatible. The converse result is proved by contradiction, using again the proof of Lemma 6.7. □

In Figure 13 (this example is the same as the one in Figure 10), we have represented the retimed reduced graphs associated with two sections (cuts) of $\mathcal{D}$. One of them is compatible and the other one is not compatible.

6.4. Summary. In section 4.2, we have established the relations between executions of the pebble game and cuts in the dependence graph. In section 6.3, we have established the relations between cuts and delays. As a by-product, we obtain the relations between delays and pebble configurations.

More precisely, let $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ be an execution such that $\mathcal{A}(t)$ is a consecutive and nonredundant (see section 6.3) cut $\forall t$. Note that we did not assume that e

is regular. With the configurations $\mathcal{A}(t)$, we associate a retiming $r(t)$ and a reduced graph $\mathcal{R}_{r(t)}$ as follows:

$$r(t)_i = \max\{n_i \mid (i, n_i) \in \mathcal{A}(t)\} .$$

We have the following situations:

- If the execution e belongs to $\mathcal{E}$, the configurations $\mathcal{A}(t)$ may have different shapes at each step. Then the reduced graphs $\mathcal{R}_{r(t)}$ may have changing values for the delays.
- For an execution e belonging to $\mathcal{RE}$, the configurations are just shifted between two steps. It implies that the reduced graphs $\mathcal{R}_{r(t)}$ are all identical, with a fixed value of the delays.
- Finally, an execution e belonging to $\mathcal{ORE}$ corresponds to identical reduced graphs $\mathcal{R}_{r(t)}$ with fixed and nonnegative values of the delays.

In the following table, we provide a summary of the main relations established so far between executions of a system of UREs, cuts in $\mathcal{D}$, and delays in $\mathcal{R}$.

Games	Executions	Cuts in $\mathcal{D}$	Delays in $\mathcal{R}$
$\mathcal{M}_1$	execution in $\mathcal{E}$	arbitrary cut	changing delays
$\mathcal{M}_2$	regular execution, $\mathcal{RE}$	consecutive cut	fixed delays
$\mathcal{M}_3$	one-pass reg. exec., $\mathcal{ORE}$	compatible cut	nonnegative fixed delays

A general remark is that the main theoretical results, Theorems 4.11 and 5.5, apply only to recycled graphs. Using them, we have been able to solve problem MinPeb for an initial graph which is recycled. However, the solution proposed involves the construction of an associated graph, which is not recycled. It is not a problem, as we do not need to apply Theorems 4.11 or 5.5 on this associated graph.

6.5. Complexity results for $\mathcal{M}_3$.

PROPOSITION 6.9. *Let $\mathcal{R} = (V, E, \Gamma)$ be the reduced graph associated with a recycled system of UREs. Under game $\mathcal{M}_3$, problem* MinPeb *can be solved using an algorithm of complexity $O(|E|^2 \log |V| + |V||E| \log^2 |V|)$.*

Proof. As detailed above, there is a one-to-one correspondence between minimal one-pass regular executions and retimed reduced graphs with nonnegative delays.

In Leiserson and Saxe (section 8 of [17]) an algorithm is given to solve the following problem: find a retimed reduced graph $\mathcal{R}_r$ with only nonnegative delays and minimizing $\Gamma_B(\mathcal{R}_r)$. It is a minimum-cost flow algorithm; it provides an explicit solution and its complexity is $O(|E|^2 \log |V| + |V||E| \log^2 |V|)$. Using Lemmas 5.3 and 6.6, the cut $\mathcal{C}(S_r)$ is compatible and the execution $\{\mathcal{C}(S_r) + t, t \in \mathbb{N}\}$ is one-pass regular and solves problem MinPeb under rules $\mathcal{M}_3$ according to Theorem 5.5. □

The algorithm of [17] was developed in the context of digital circuits; see section 7. An efficient implementation of this algorithm can be found in Shenoy and Rudell [24].

7. Application 1: Registers in circuit design. In this section we will show how the previous results relate to the problem of register minimization in digital circuits. The interest of the relation is two-fold. First, algorithms developed for digital circuits can be used to get optimal executions of a system of UREs; see the previous section. Second, we will show that the results proved so far enable us to prove some new results for recycled digital circuits; see Theorem 7.4 below.

7.1. Definition of a circuit. A digital circuit is constituted by functional gates, wires, and registers. More precisely, (i) a functional element computes output data from one or several input data. For example, in the case of a logical circuit, the functional elements will be boolean logical gates (AND, OR, ...); (ii) a wire between element i and element j enables us to transfer the output data of i which becomes an input data for j; (iii) a register corresponds to a storage facility, or a memory cell of finite size. If there are p registers between elements i and j, it enables us to keep in memory the last p values computed by the element i.

The model of the behavior of the system is the following. There is a global clock for the system. Between two clock ticks, here are the operations taking place.

- Functional element: (1) receive the input data from upstream registers or elements; (2) compute a new output data; (3) send the output data to downstream registers or elements.
- Register: (1) transmit the stored data downstream to another register or a functional element; (2) remove the stored data; (3) receive and store new data from upstream from another register or a functional element.

Between two clock ticks, these operations are performed at *all* functional elements and registers.

Let $X_i(n)$ be the nth variable computed at element i. Since registers are finite size memory cells, the variables $X_i(n)$ can only take a finite number of values. The set of all these possible values is denoted by $\mathcal{W}$.

After n clock ticks, for each element i, the variables $\{X_i(m), m \leq n\}$ have been computed. The number of registers on a wire between i and j corresponds to the number of variables $X_i(n-k)$ which need to be still in the memory in order to carry on the computation of the variables $X_j(n+m), m > 0$.

It follows from the previous description that a digital circuit can be viewed as the reduced graph $\mathcal{R}$ of some system of UREs. The functional elements of the circuit correspond to the nodes of $\mathcal{R}$, the wires to the arcs, and the registers to the delays. The computation operation corresponding to the functional element i is denoted by F_i to be consistent with previous notations. In the remainder of the section, we will use interchangeably the terminology of digital circuits and the one of reduced graphs. We consider only *constructive circuits*, i.e., circuits whose associated URE is constructive, and *recycled circuits*, i.e., circuits whose associated reduced graph is recycled.

The specificity of digital circuits (with respect to general reduced graphs) is that *only nonnegative registers (delays) have a physical meaning.*

In Figure 14, we have represented the flow of data between clock ticks in a digital circuit. The graphical convention is consistent with that of reduced graphs.

7.2. Minimizing the number of registers. We consider problem *MinReg*, which is classic in circuit design; see, for instance, Leiserson and Saxe [17, section 8].

PROBLEM 2 (MinReg). *Given a recycled circuit, find a new circuit preserving the functional behavior and having a minimal number of registers.*

To make this statement precise, we have to define rigorously the *functional behavior* and the *number of registers* of a circuit.

Let $\{X_i(n), i \in V, n \in \mathbb{Z}\}$ and $\{\tilde{X}_i(n), i \in \tilde{V}, n \in \mathbb{Z}\}$ be the variables computed by the original and the new circuits, respectively. The preservation of the functional behavior means that we have

$$\forall i \in V,\ \exists u \in \tilde{V},\ \exists c_i \in \mathbb{Z} \text{ s.t. } \forall n \in \mathbb{Z},\ X_i(n) = \tilde{X}_u(n + c_i)\,. \tag{11}$$

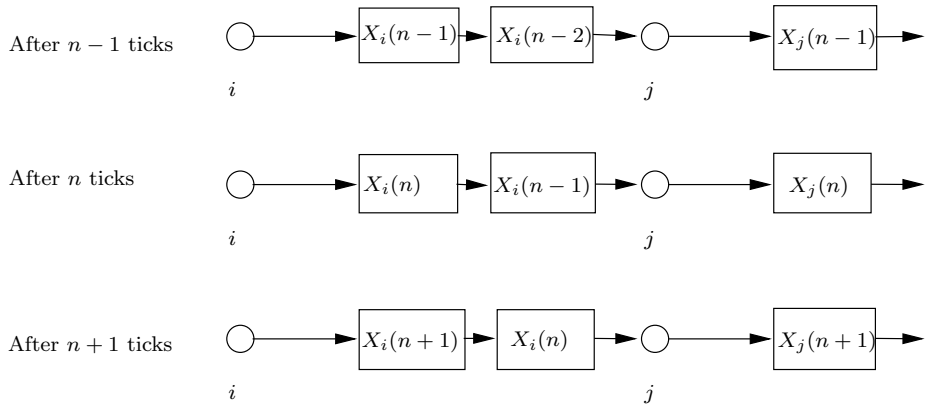


FIG. 14. *Digital circuit computing* $X_j(n) = F_j(X_i(n-2), \dots)$.

Registers can be viewed as delays in a reduced graph and the number of registers of a circuit $\mathcal{R}$ is the quantity $\Gamma_A(\mathcal{R})$ defined in section 6.2. Now, starting from a circuit $\mathcal{R}$, we can always perform the `Forward Splitting` algorithm (Algorithm 6.4) to obtain a circuit $\mathcal{R}'$. As an illustration, consider the example given in Figure 12. The `Forward Splitting` algorithm is called *register sharing* in the context of circuits; see [17].

The problem MinReg is directly connected with the notions introduced in sections 4 and 5. It enables us to propose some complements to the results of [17] for the special case of recycled circuits.

In [17], Leiserson and Saxe define a notion of retiming which is exactly that of Definition 6.1. They restrict their attention to *legal* retimings, as they are the only ones to have a physical meaning for circuits.

DEFINITION 7.1. *A retiming* r *is* legal *if* $\mathcal{R}_r$ *has only nonnegative delays.*

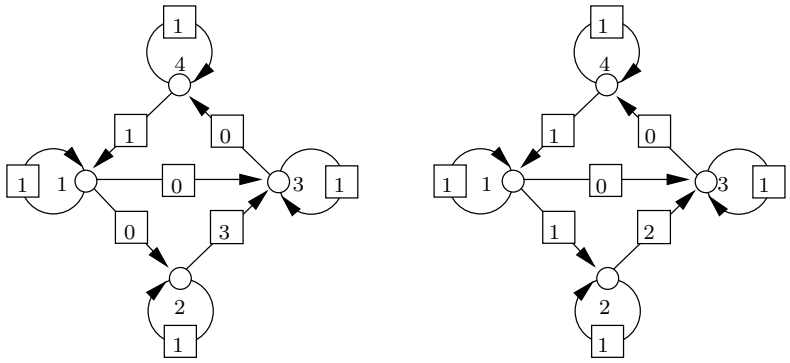
An example of legal retiming is given in Figure 15. If we perform register sharing on the original circuit (Figure 15(a)), we would obtain a circuit with six registers. After a legal retiming and register sharing, we have only five registers (Figure 15(c)).

Leiserson and Saxe proved that retiming and register sharing preserve the functional behavior of the circuit (this is also a direct consequence of Lemma 6.3 and Proposition 6.5). Then they proposed an algorithm to compute the optimal legal retiming and also the optimal legal retiming when register sharing is allowed; see section 6.5.

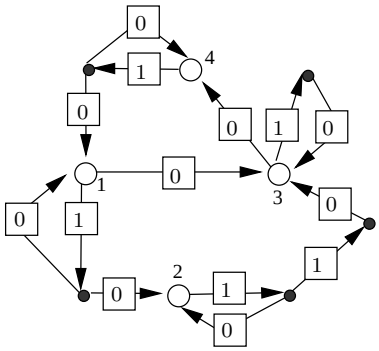
However, the question whether other circuit transformations can be used to get a circuit with even fewer registers remains to be answered.

Let us consider the best possible retiming in the original circuit without restricting ourselves to legal retimings. It corresponds to the choice of a minimal consecutive (but not necessarily compatible) cut in the associated dependence graph $\mathcal{D}$; see section 6.4. In the corresponding reduced graph, there may be some negative delays that cannot represent registers. However, it is possible to perform some appropriate modifications to go back to positive delays. It is done by duplicating some nodes in the circuit.

The `Duplicate` algorithm takes as input a graph $\mathcal{R} = (V, E, \Gamma)$ and produces a new graph $\mathcal{R}' = (V', E', \Gamma'), \Gamma' \geq 0$. In the description to follow, we use the notion of delays on paths: if P is an oriented path in $\mathcal{R}$, then the delay of P is the sum of all the delays on the arcs of P.



(a) Before retiming (8 registers). (b) After optimal legal retiming.



(c) After retiming and register sharing (5 registers).

FIG. 15. *Reducing the number of registers using legal retiming and register sharing.*

ALGORITHM 7.2 (Duplicate).

Input: *Reduced graph* $\mathcal{R} = (V, E, \Gamma)$, *functions associated with the nodes* $\{F_i, i \in V\}$.

1. Set $V' = V$ and $E' = \emptyset$. Associated functions $F'_i = F_i, i \in V$.
2. For each node v in V, let $k(v)$ be the minimum delay of all paths in $\mathcal{R}$ starting in v.
   - Set $v_0 = v$.
   - If $k(v) < 0$, then create $|k(v)|$ additional nodes in V', $v_1, \ldots, v_{|k(v)|}$, with associated functions, $F'_{v_i} = F_v$.
3. For each arc $(u, v) \in E$ with delay $\Gamma(u, v) = \gamma$, create in E' all the arcs of type $(u_{\max(0, j-\gamma)}, v_j)$ with delay $\max(0, \gamma - j) \, \forall 0 \leq j \leq \max(0, k(v))$.

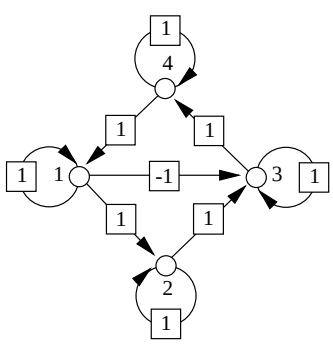
Output: *Reduced graph* $\mathcal{R}' = (V', E', \Gamma')$, $\Gamma' \geq 0$. *Associated functions* $\{F'_i, i \in V'\}$.

For each node $v \in V$, $k(v)$ is finite and is reached on a finite path since the constructivity of $\mathcal{R}$ implies that all circuits in $\mathcal{R}$ have a strictly positive delay.

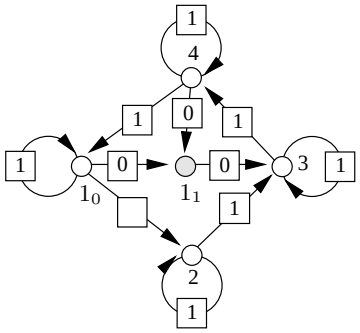
The following proposition justifies the use of the Duplicate algorithm.

PROPOSITION 7.3. *Let* $\mathcal{S}$ *and* $\mathcal{S}'$ *be the systems of UREs associated with* $\mathcal{R}$ *and* $\mathcal{R}'$, *respectively.*

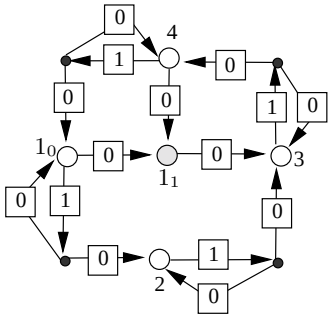
(i) S *and* S' *have the same functional behavior. More precisely, borrowing the*



(a) Circuit after optimal general retiming.



(b) After duplication.



(c) After duplication and register sharing (4 registers).

FIG. 16. *Reducing the number of registers using retiming, duplication and register sharing.*

notation of the `Duplicate` *algorithm, we have*

$$X'_{v_j}(n) = X_v(n+j) \quad \forall v_j \in V', \forall n \in \mathbb{Z}\,. \tag{12}$$

(ii) *We have* $\Gamma_B(\mathcal{R}) = \Gamma_B(\mathcal{R}')$.

Proof. The proof of (i) follows directly from the construction rules of $\mathcal{R}'$. Considering (12), specialized to $j = 0$, we get $X'_{v_0}(n) = X_v(n)$. The original circuit is embedded in the new circuit. As for (ii), note that all the arcs exiting a duplication node (of type $v_j, j > 0$) have a zero delay. As for the new arcs from the nodes of type v_0, they all have delays smaller than or equal to the delays of the arcs of the original graph. The original arcs are kept with their original delays unchanged. We conclude that $\Gamma_B(\mathcal{R}) = \Gamma_B(\mathcal{R}')$. □

It is important to remark that the reduced graph $\mathcal{R}'$ (hence the associated system of UREs) obtained by duplication is not recycled anymore.

We illustrate the algorithm `Duplicate` on an example. The circuit of Figure 16(a) is obtained from the one of Figure 15(a) by performing a nonlegal retiming. By applying the `Duplicate` algorithm, we obtain the circuit of Figure 16(b), where node 1 has been duplicated into nodes 1_0 and 1_1.

Now, after performing register sharing, the resulting circuit has only 4 registers (see Figure 16(c)). The minimal number of registers we could get with only legal retimings and register sharing was 5; see Figure 15(c).

We now state the general result.

THEOREM 7.4. *Let us consider a recycled digital circuit. When the functions computed at each node are general, a circuit with the same functional behavior and a*

minimal number of registers can be obtained by performing solely the three following operations (in this order):

1. *General retiming;* 2. `Duplicate` *algorithm;* 3. `Forward Splitting` *algorithm (register sharing).*

Proof. Let us consider a recycled graph $\mathcal{R} = (V, E, \Gamma)$ (associated dependence graph $\mathcal{D}$). We realize the following operations: 1. Perform the optimal general retiming to obtain a graph $\mathcal{R}_1$. 2. Apply the `Duplicate` algorithm to $\mathcal{R}_1$ to obtain the graph $\mathcal{R}_2$. 3. Transform the graph $\mathcal{R}_2$ into $\mathcal{R}_3$ by applying the `Forward Splitting` algorithm.

Let us detail the first operation. We find a minimal consecutive cut C of $\mathcal{D}$ (Proposition 4.12). Let $\{(i, r(i)), i \in V\}$ be the right section of C. We define the retimed graph $\mathcal{R}_1 = \mathcal{R}_r$. Using Lemma 6.6, $\Gamma_B(\mathcal{R}_1)$ is equal to the cardinal of C.

The three operations preserve the functional behavior of the circuit; see Lemma 6.3 and Propositions 6.5 and 7.3. Furthermore, as a consequence of Proposition 7.3 and (9), we have $\Gamma_B(\mathcal{R}_1) = \Gamma_B(\mathcal{R}_2) = \Gamma_A(\mathcal{R}_3)$. We conclude that $\Gamma_A(\mathcal{R}_3) = |C|$. It remains to be proved that there exists no other circuit having the same functional behavior as $\mathcal{R}$ and with fewer registers than $\mathcal{R}_3$.

We consider $\mathcal{R}' = (V', E', \Gamma')$ another circuit which has the same functional behavior as $\mathcal{R}$. Let $\mathcal{D}'$ be the dependence graph associated with $\mathcal{R}'$. The preservation of the functional behavior implies that the set of nodes in $\mathcal{D}$ is included in the set of nodes in $\mathcal{D}'$. The mapping of the nodes of $\mathcal{D}$ onto $\mathcal{D}'$ that preserves the functional behavior is denoted by ϕ. By definition, $X_i(n) = X'_{\phi(i)}(\phi(n))$ if the node (i, n) in $\mathcal{D}$ is mapped on the node $(\phi(i), \phi(n)) = \phi(i, n)$ in $\mathcal{D}'$.

We consider a minimal cut C of $\mathcal{D}$. In $\mathcal{D}'$, we suppose that there exists a cut C' such that $|C'| < |C|$. We can assume that in $\mathcal{D}'$, we have $\phi(C)$ on the "left" of C' and $\phi(C + k)$ on the "right" of C' by choosing k large enough.

We recall that $\mathcal{W}$ denotes the finite set of possible values for a variable $X_i(n)$. In the dependence graph $\mathcal{D}$, there exists a periodic flow (node-disjoint paths) from C to $C + k$ of size $|C|$ (Corollary 4.8). It implies that there is a general dependence between the variables attached to C and the variables attached to $C + k$, which can be put under the form of a general function $F : \mathcal{W}^{|C|} \to \mathcal{W}^{|C|}$. In particular, the functions F_i in (2) can be chosen such that F is bijective and does not depend on n. For example, choose $F_i(X_j(n - \gamma), \dots) = X_j(n - \gamma)$ if the arc $(j, n - \gamma) \to (i, n)$ belongs to the periodic flow. In this case, the function F is merely a permutation of the coordinates.

Now let us consider the graph $\mathcal{D}'$; using the existence of the cut C', the function F can be decomposed as $F : \mathcal{W}^{|C|} \to \mathcal{W}^{|C'|} \to \mathcal{W}^{|C|}$. As $\mathcal{W}$ is finite, it contradicts the fact that F can be bijective.

The smallest cut C' in $\mathcal{D}'$ is at least as large as C. Finally, by using Lemma 6.6, we conclude that $\Gamma_A(\mathcal{R}') \geq \Gamma_B(\mathcal{R}') = |C'| \geq |C| = \Gamma_A(\mathcal{R}_3)$. □

Recall that, in Leiserson and Saxe [17], only legal retimings were considered. In Theorem 7.4, by considering all possible retimings, we were able to obtain a circuit with fewer registers, and even a minimal number of registers. However, in doing so, we obtain a circuit with a possibly larger number of functional elements. Hence, the practical interest of Theorem 7.4 also needs to be discussed in terms of the compared costs of functional elements and registers. In fact, the number of functional elements is increased both by the `Duplicate` and by the `Forward Splitting` algorithms. On the one hand, in the `Forward Splitting` case, only "dummy functions" are added. In the context of circuits, they consist of simple wire connections and do not perform

any operation, so that they should be very cheap to implement. On the other hand, in the `Duplicate` case, the added elements are equivalent to some of the original functional elements. Hence they could be more expensive to implement.

7.3. Summary and complexity. The transformations done to a circuit in order to obtain an equivalent circuit with the minimal number of registers can be summarized by the following scheme.

$$\begin{array}{ccccccc} & \text{optimal} & & & & \text{forward} & \\ & \text{retiming} & & \text{duplicate} & & \text{splitting} & \\ \mathcal{R} & \longrightarrow & \mathcal{R}_1 & \longrightarrow & \mathcal{R}_2 & \longrightarrow & \mathcal{R}_3 \\ & & \Gamma(\mathcal{R}_1) \in \mathbb{Z}^{E_1} & & \Gamma(\mathcal{R}_2) \in \mathbb{N}^{E_2} & & \Gamma(\mathcal{R}_3) \in \{0,1\}^{E_3} \\ & & \Gamma_B(\mathcal{R}_1) \text{ minimal} & & \Gamma_B(\mathcal{R}_2) = \Gamma_B(\mathcal{R}_1) & & \Gamma_A(\mathcal{R}_3) = \Gamma_B(\mathcal{R}_2) \end{array}$$

COROLLARY 7.5. *Let us consider a recycled digital circuit. A new circuit solving problem MinReg can be obtained with an algorithm of complexity* $O(\Gamma_A(\mathcal{R})^2|V|^2 + |V|^3)$.

Proof. In the proof of Theorem 7.4, the algorithms used to go from $\mathcal{R}$ to $\mathcal{R}_1$, from $\mathcal{R}_1$ to $\mathcal{R}_2$, and from $\mathcal{R}_2$ to $\mathcal{R}_3$ are all polynomial, and $\mathcal{R}_3$ is a solution to the problem MinReg.

To obtain $\mathcal{R}_1$, we apply the algorithm of Proposition 4.12 whose complexity is $O(\Gamma_A^2|V|^2)$. To obtain $\mathcal{R}_2$, we apply the `Duplicate` algorithm (Algorithm 7.2), whose complexity is $O(|V|^3 + \Gamma_A|E|)$. Let us justify this complexity. The first step consists of computing the quantities $k(u), u \in V$. It is equivalent to the search of a minimal weight path in a weighted graph. This can be done using the Floyd algorithm with a complexity $O(|V|^3)$; see, for instance, Gondran and Minoux [14, Chapters 2 and 3]. Let $M = \max_{v\in V}(0, -k(v))$. From the constructiveness, it follows that M has to be smaller than Γ_A. Now, the second step of the algorithm consists of creating at most $M|V|$ nodes and $M|E|$ arcs. The complexity of this step is at most $O(\Gamma_A|E|)$.

To obtain $\mathcal{R}_2$, we apply the `Forward Splitting` algorithm (Algorithm 6.4). In this algorithm, we create at most Γ_B nodes and $|E|$ arcs, which accounts for a complexity $O(\Gamma_B + |E|)$. □

Corollary 7.5 is interesting, as it is not straightforward to extend the original algorithm of Leiserson and Saxe (for legal retimings, see section 6.5) to general retimings.

8. Application 2: Task graphs evaluation. Task graphs are widely used in the modeling and analysis of parallel programs and architectures [1]. Yet, the performance evaluation of task graphs is difficult in general.

The term task graphs covers a wide variety of models, with the following common feature: each task depends on a finite number of tasks and can be executed only when all the tasks it depends on are completed. Here, we consider *repetitive task graphs,* which are bi-infinite task graphs generated by the periodic replication of a given finite task graph (with set of nodes V); see [3].

Let us denote by $X_i(n), i \in V, n \in \mathbb{Z}$, the epoch when the nth occurrence of task i is completed. Let the sets $\Delta_i, i \in V$, describe the dependences between tasks. The variables $X_i(n)$ are given by a recursion of the following form:

$$X_i(n) = \max_{(j,\gamma)\in\Delta_i} (X_j(n-\gamma) + \sigma_{j,i,\gamma}(n)),\ i \in V, n \in \mathbb{Z},\ \sigma_{j,i,\gamma}(n) \in \mathbb{R}\,. \tag{13}$$

We will present the optimization problem which arises in the fast parallel computation of the evolution equations of task graphs and apply the preceding results to solve it.

8.1. Max-plus recurrences. The evolution equations (13) can be viewed as both a specialization and a generalization of a system of UREs. On the one hand, the functions have a specific form, implying only the operations max and +. On the other hand, the functions depend on n. We call a "max-plus recurrence" (MPR) an equation of the form (13). From now on, we assume that the MPR is *constructive* and *recycled*, i.e., that $\forall i, (i,1) \in \Delta_i$. It is a natural assumption for task graphs, as it means that the nth occurrence of a task cannot start before the completion of the $(n-1)$th occurrence of the same task.

The (max,+) formalism is a convenient tool to work with MPR. We briefly introduce it.

DEFINITION 8.1. *The* (max,+) semiring $\mathbb{R}_{\max}$ *is the set* $\mathbb{R} \cup \{-\infty\}$*, equipped with the two operations* max *and* + *denoted, respectively, by* $\oplus$ *and* $\otimes$ *(*$a \oplus b = \max(a,b)$ *and* $a \otimes b = a + b$*). The elements* $-\infty$ *and* 0 *are the neutral elements of the laws* $\oplus$ *and* $\otimes$*, respectively.*

For matrices of appropriate sizes, we define $(A \oplus B)_{ij} = A_{ij} \oplus B_{ij} = \max(A_{ij}, B_{ij})$, $(A \otimes B)_{ij} = \bigoplus_l A_{il} \otimes B_{lj} = \max_l (A_{il} + B_{lj})$, and for a scalar a, $(a \otimes A)_{ij} = a \otimes A_{ij} = a + A_{ij}$. When no confusion is possible, we abbreviate $A \otimes B$ to AB.

We can rewrite (13) with the previously defined notation. Let $X(n)$ be the column vectors of coordinates $X_i(n)$, and let $A(\gamma, n)$ be the matrix with coordinates $A(\gamma,n)_{ij} = \sigma_{j,i,\gamma}(n)$ if $(j,\gamma) \in \Delta_i$ and $A(\gamma,n)_{ij} = -\infty$ otherwise. Now let $U = \{\gamma \mid \exists i, j \text{ s.t. } (j,\gamma) \in \Delta_i\}$ and $\overline{\gamma} = \max_U \gamma$. We have

$$X(n) = \bigoplus_{\gamma \in U} A(\gamma, n) \otimes X(n-\gamma)\,. \tag{14}$$

This is a *linear system* in the $(\max, +)$ semiring.

Representation of order 1. A standard step in the analysis of linear systems is the transformation of a recurrence like (14) into an "equivalent" system of order 1, such as (15), where $\tilde{X}(n), \tilde{X}(n-1) \in \mathbb{R}_{\max}^{\tilde{V}}$, and $A(n-1) \in \mathbb{R}_{\max}^{\tilde{V} \times \tilde{V}}$.

$$\tilde{X}(n) \;=\; A(n-1) \otimes \tilde{X}(n-1). \tag{15}$$

The system in (15) is "equivalent" to the original one if it preserves the functional behavior, meaning that $\forall i \in V$, $\exists u \in \tilde{V}$, $\exists c_i \in \mathbb{Z}$ s.t. $\forall n \in \mathbb{Z}$, $X_i(n) = \tilde{X}_u(n + c_i)$. We say that the system in (15) is an *order* 1 *representation* of the original one.

Assume that all the delays γ in (14) are greater or equal to 1. Then the transformation can be done by setting

$$\tilde{X}_{i|V|+j}(n) = X_j(n-i), \;\; i = 0, \ldots, \overline{\gamma} - 1, \;\; j \in V.$$

In this case, the dimension of the order 1 representation is $|\tilde{V}| = |V|\overline{\gamma}$. We will see below that an order 1 representation can be obtained without any assumption on the delays (except constructivity).

We can now define the main problem to be addressed in this section.

PROBLEM 3 (MinSize). *Given a recycled MPR, find an equivalent MPR of order* 1 *and of minimal dimension.*

For strictly positive delays, it follows from the discussion above that the minimal dimension is at most equal to $|V|\overline{\gamma}$. We will see that in general it is much lower. Problem MinSize is very natural. A practical motivation for it is provided in next section.

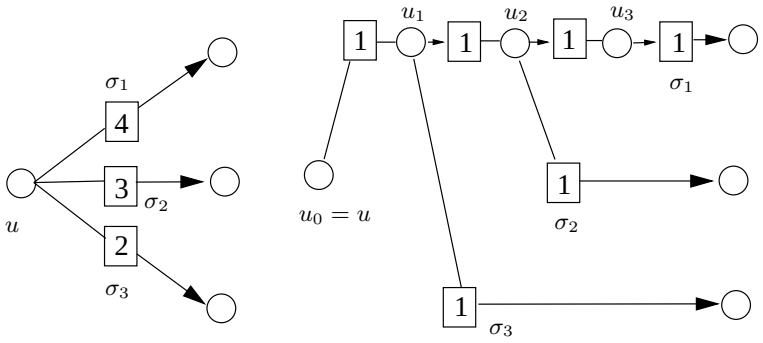


FIG. 17. `Forward Splitting-II` *of a reduced graph.*

Remark 8.2. In problem MinSize, the optimality of the size of the representation should be understood as the best possible that can be obtained without making assumptions on the value of the numbers $\sigma_{i,j,\gamma}(n)$. When these numbers are constant and known, this knowledge may be exploited to obtain a *minimal realization* in the sense of linear system theory [20, 10], which is normally smaller than ours. Finding a minimal realization is a difficult problem, and algorithms are known only in very specific cases. A deeper investigation of the relations between the two approaches is an interesting direction for further research.

8.2. Parallel evaluation of MPR. The *evaluation* of an MPR consists of computing all the variables $X(n)$. We assume that we want to perform this evaluation using a parallel machine. If we have an order 1 representation of the system, a possible and efficient algorithm is the *parallel prefix principle*: as the multiplication of matrices in (max,+) is associative, it is possible to divide the computation of $A(n) \otimes \cdots \otimes A(1)$ into smaller products $A(p) \otimes \cdots \otimes A(q)$ which may be computed by different processors.

The number of operations required to compute the variables up to $X(n)$ on a CREW-PRAM machine with P processors is $O(\ell^3(n/P + \log(P)))$, where ℓ is the size of the matrix of the linear system. Since n and P are fixed parameters, the complexity is minimized by having an order 1 representation of the MPR of minimal dimension.

8.3. Reduced graphs. As for any system of UREs, we can associate a reduced graph $\mathcal{R} = (V, E, \Gamma)$ to a given MPR. To each node $i \in V$, we associate the sequences $\{\sigma_{j,i,\gamma}(n), n \in \mathbb{Z}\}$, $(j, \gamma) \in \Delta_i$.

We will show that the solution to problem MinSize is a matrix of dimension $\min_r \Gamma_B(\mathcal{R}_r), r \in \mathbb{Z}^V$. This result seems to be new.

We transform any reduced graph by the following procedure. For each node in the reduced graph, we create new (dummy) nodes and new arcs in a tree-like fashion, as in Figure 17, such that each arc in the new reduced graph has a delay at most 1. The added dummy nodes are recycled. (The recyclings are not shown on the figure.)

More formally, the transformation can be done using the algorithm `Forward Splitting-II` described below. We use the notation $u^\bullet$ to denote the set of successor arcs of u. It is not assumed that the delays are positive.

ALGORITHM 8.3 (Forward Splitting-II).

Input: *Recycled reduced graph* $\mathcal{R} = (V, E, \Gamma)$. *Sequences* $\{\sigma_{j,i,\gamma}(n), n \in \mathbb{Z}\}$.

`1. We set` $V' = V, E' = \emptyset$.

`2. For all node` $u \in V$`, let` $\delta(u) = \max_{a \in u^\bullet} \Gamma(a)$.

(The graph is recycled, so $\delta(u) \geq 1$)
If $\delta(u) = 1$, then u and $u^\bullet$ remain unchanged.
If $\delta(u) > 1$, then
- in E', set $u_0 = u$ and create $\delta(u) - 1$ recycled nodes $u_1, \ldots, u_{\delta(u)-1}$.
  Create the arcs $a_i = (u_i, u_{i+1})$ with delay $\Gamma'(a_i) = 1$ for $i = 0, \ldots, \delta(u) - 2$.
  The sequences associated with nodes $u_i, i > 0$ are $\{\sigma'_{u_i,u_{i+1},1}(n), n \in \mathbb{Z}\} = 0$.
- For each arc (u, v) in E with delay γ,
  - if $\gamma \leq 1$, then create in E' the arc (u_0, v_0) with delay γ and sequence $\{\sigma'_{u_0,v_0,\gamma}(n)\} = \{\sigma_{u,v,\gamma}(n)\}$
  - else create in E' the arc $(u_{\gamma-1}, v_0)$ with delay 1 and sequence $\{\sigma'_{u_0,v_{\gamma-1},1}(n)\} = \{\sigma_{u,v,\gamma}(n)\}$.

Output: *Recycled reduced graph* $\mathcal{R}' = (V', E', \Gamma')$. *Sequences* $\{\sigma'_{u,v,\gamma}(n), n \in \mathbb{Z}\}$.

The new reduced graph has a maximum delay per arc equal to 1.

PROPOSITION 8.4. *We use the notation defined in the above algorithm.*

(i) *The reduced graph* $\mathcal{R}'$ *has the same functional behavior as the original graph* $\mathcal{R}$*:*

$$X'_{u_i}(n) = X_u(n - i) \quad \forall u \in V,\ \forall n \in \mathbb{Z}\,. \tag{16}$$

(ii) *The number of nodes in* $\mathcal{R}'$ *is* $|V'| = \Gamma_B(\mathcal{R})$.

Proof. Point (i) follows directly from the algorithm. Let us prove point (ii). Using the notation of Algorithm 8.3, in the new reduced graph, we have $\delta(u)$ nodes $(u_0, u_1, \ldots, u_{\delta(u)-1})$ for each node u in $\mathcal{R}$. The total number of node in $\mathcal{R}'$ is $\sum_{u\in\mathcal{R}} \delta(u) = \Gamma_B(\mathcal{R})$. □

Proposition 8.4 is already an improvement over the standard representation as we obtain an order 1 MPR of dimension $\Gamma_B(\mathcal{R})$ instead of $\overline{\gamma}|V|$. Another improvement consists of finding first a retiming r of the graph such that $\Gamma_B(\mathcal{R}_r)$ is minimized.

To fix the notation, let $\mathcal{R} = (V, E, \Gamma)$ be the original reduced graph and $\mathcal{R}_1 = (V, E, \Gamma_1)$ be a retimed reduced graph minimizing Γ_B. We perform the `Forward Splitting-II` algorithm on $\mathcal{R}_1$ to get a new graph $\tilde{\mathcal{R}} = (\tilde{V}, \tilde{E}, \tilde{\Gamma})$ with all delays smaller or equal to 1.

Some of the delays of $\tilde{\Gamma}$ might be negative. However, we prove that it is still possible to get an order 1 representation of dimension $|\tilde{V}|$ of the MPR associated with $\tilde{\mathcal{R}}$. We denote by $a^\bullet$ the ending node of an arc $a \in \tilde{E}$.

LEMMA 8.5. *Let* $\{\tilde{X}_i(n), i \in \tilde{V}, n \in \mathbb{Z}\}$ *be the variables associated with* $\tilde{\mathcal{R}}$. *We have*

$$\tilde{X}(n) = B(n) \otimes \tilde{X}(n - 1), \tag{17}$$

with $B_{ij}(n) = \max_{\pi\in\Pi_{j,i}} \sum_{a\in\pi\cap\tilde{E}} \sigma_{a,\tilde{\Gamma}(a)}(n - m(a, \pi))$, *where* $\Pi_{j,i}$ *is the set of all the paths from* j *to* i *in* $\tilde{\mathcal{R}}$ *with total delay equal to* 1 *and* $m(a, \pi)$ *is the total delay on the path* π *from* $a^\bullet$ *to node* i.

Proof. The first stage of the proof consists of showing that all paths ending in node i have a total delay at least 1 provided they are long enough. Let π be a path ending in i. The length (number of nodes) of π is denoted by $l(\pi)$. Let h be the sum of the negative delays in $\tilde{\mathcal{R}}$: $h = \sum_{a\in\tilde{E}} \min(0, \tilde{\Gamma}(a))$. Assume that $l(\pi) > (-h+2)|\tilde{V}|$. Therefore, the path π must contain at least $-h + 2$ cycles. By constructivity, each

cycle has a total delay which is strictly positive. The set of cycles contained in π is denoted $C(\pi)$. Then,

$$\begin{aligned}\tilde{\Gamma}(\pi) &= \sum_{p\in\pi}\tilde{\Gamma}(p)\\ &= \sum_{p\in C(\pi)}\tilde{\Gamma}(p) + \sum_{p\in\pi\setminus C(\pi)}\tilde{\Gamma}(p) \;\geq\; |C(\pi)| + h \;\geq 2\,.\end{aligned}$$

All the paths in $\Pi_{j,i}$ have a length smaller than $(-h+2)|\tilde{V}|$. Since the graph $\tilde{\mathcal{R}}$ is finite, then $\Pi_{j,i}$ is a finite set.

Now, the equation on variables $\tilde{X}_i(n)$ in $\tilde{\mathcal{R}}$ can be written as

$$\begin{aligned}\tilde{X}_i(n) &= \max_{(j,\gamma)\in\tilde{\Delta}_i}(\tilde{X}_j(n-\gamma)+\tilde{\sigma}_{j,i,\gamma}(n))\\ &= \max_{(j,\gamma)\in\tilde{\Delta}_i,\gamma=1}(\tilde{X}_j(n-1)+\tilde{\sigma}_{j,i,1}(n)) \vee \max_{(j,\gamma)\in\tilde{\Delta}_i,\gamma\leq 0}(\tilde{X}_j(n-\gamma)+\tilde{\sigma}_{j,i,\gamma}(n)).\end{aligned}$$

In the latest equation, we replace all the variables $\tilde{X}_j(n-\gamma), \gamma\leq 0$, by their value until getting only variables of the type $\tilde{X}_j(n-\gamma), \gamma = 1$. By using the distributivity of $+$ with respect to max, we get (17). □

Using the results of the previous sections, and in particular Theorems 4.11 and 7.4 and Lemma 6.6, we obtain the following theorem.

THEOREM 8.6. *Given a recycled MPR, its associated MPR of order* 1 *and of minimal size (problem* MinSize*) has the same size as the minimal cut in the dependence graph of the MPR.*

Proof. We provide a sketch of the proof, which is an adaptation of that of Theorem 7.4. There, we used in a critical way the existence of a *finite* set $\mathcal{W}$ of possible values for the variables in a digital circuit. In an MPR, the variables take their value in $\mathbb{R}\cup\{\infty\}$, but on the other end, the function involved are all (max,+)-linear. Hence, we adapt the argument of Theorem 7.4 as follows.

We have to prove that a (max,+)-linear function from $\mathbb{R}^{|C|}$ to $\mathbb{R}^{|C|}$ ($|C|$ being the size of the minimal cut in the dependence graph $\mathcal{D}$) where the coefficients $\sigma_{ijk}(n)$ are arbitrary may not be further reduced.

Let $C = \{(i_1,n_1),\ldots,(i_{|C|},n_{|C|})\}$ be a minimal cut in $\mathcal{D}$ and let $\{X_{i_j}(n_j), j = 1,\ldots,|C|\}$ be the corresponding set of variables. Now, choose an integer k large enough such that C and $C+k$ do not share any node. There exists $|C|$ node-disjoint paths in $\mathcal{D}$ from C to $C+k$. Let $\mathcal{F}_C$ be the flow constituted by these paths. On each one of these paths, choose all the σ variables to be equal to 0. On all the arcs between C and $C+k$ which do not belong to those paths, set the σ variable to be smaller than $-\max_{j,k}|X_{i_j}(n_j)-X_{i_k}(n_k)|$. Let (i_l,n_l+k) be the successor on $(C+k)$ of (i_j,n_j) following the flow $\mathcal{F}_C$. We have $X_{i_l}(n_l+k) = X_{i_j}(n_j)$.

The set $\{X_{i_j}(n_j+k), j=1,\ldots,|C|\}$ is formed by independent variables $\forall k$ and cannot be reduced. The corresponding matrices $B(n)$ are permutation matrices (i.e., there exists a permutation σ such that $B_{\sigma(j),j}=0$ and $B_{ij}=-\infty$ for $i\neq\sigma(j)$). The remainder of the argument follows Theorem 7.4. □

8.4. Summary and complexity. A summary of the algorithm to find a solution to problem MinSize is given by the following scheme.

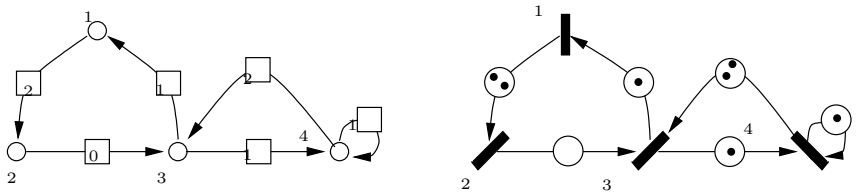


FIG. 18. *Transformation of a reduced graph* (a) *into a Petri net* (b).

$$\begin{array}{ccccccccc} & \text{reduced graph} & & \text{optimal retiming} & & & \text{forward splitting II} & & \text{Lemma 8.5} & \\ \text{MPR} & \longrightarrow & \mathcal{R} & \longrightarrow & \mathcal{R}_1 & \longrightarrow & \mathcal{R}_2 & \longrightarrow & \text{MPR} \\ \text{order } k & & & & \Gamma(\mathcal{R}_1) \in \mathbb{Z}^{E_1} & & \Gamma(\mathcal{R}_2) \leq 1 & & \text{order 1} \\ \text{dim. } |V| & & & & \Gamma_B(\mathcal{R}_1) \text{ minimal} & & |V_2| = \Gamma_B(\mathcal{R}_1) & & \text{dim. } |V_2| \end{array}$$

COROLLARY 8.7. *Let us consider a recycled MPR. An order* 1 *representation solving problem MinSize can be obtained with an algorithm of polynomial complexity to obtain graph* $\mathcal{R}_2$ *and pseudopolynomial complexity to construct matrix* $B(n)$.

Proof. This is a sketch of the proof. Let $\mathcal{R} = (V, E, \Gamma)$. The graph $\mathcal{R}_1$ is obtained using the algorithm of Proposition 4.12 whose complexity is $O(\Gamma_A(\mathcal{R})^2|V|^2)$. The graph $\mathcal{R}_3$ is obtained by applying the `Forward Splitting-II` algorithm. In this algorithm, we create at most $\Gamma_B(\mathcal{R}_2)$ nodes and $\Gamma_B(\mathcal{R}_2) + |E|$ arcs. Its complexity is at most $\Gamma_B(\mathcal{R}_2) + |E|$. The computation of an element of matrix $B(n)$ defined in Lemma 8.5 is NP-complete (reduction of knapsack with multiplicities). However, it can be obtained with pseudopolynomial complexity using dynamic programming techniques (similar to knapsack). □

Given an initial reduced graph $\mathcal{R}$, problem MinReg in section 7 and problem MinSize in section 8 are solved using the same graph $\tilde{\mathcal{R}}$, which is the retimed graph of $\mathcal{R}$ minimizing Γ_B. However, the exact solutions of problems MinReg and MinSize are obtained by performing two different types of transformations on $\tilde{\mathcal{R}}$, yielding two different graphs, say, $\mathcal{R}_{\text{reg}}$ and $\mathcal{R}_{\text{size}}$.

In general $\mathcal{R}_{\text{reg}}$ does not provide a solution to problem MinSize (it does not have a minimal number of nodes) and $\mathcal{R}_{\text{size}}$ does not provide a solution to problem MinReg (it does not have a minimal number of registers). To check this, consider, for instance, the graph of Figure 12 and apply the `Forward Splitting-II` algorithm to it. We conclude that the problems MinReg and MinSize are different, although related.

8.5. Event graphs. *Event graphs*, a subclass of *Petri nets*, are commonly used to model discrete event systems [2, 19]. The graphical formalism for event graphs is close but different from the one of reduced graphs; see Figure 18.

In the case of a timed recycled event graph, the dynamic can be represented by an MPR as in (13) with $\sigma_{i,j,\gamma}(n) = \phi_i(n) + h_{i,j},\ \phi_i(n) \geq 0, h_{i,j} \geq 0$. Since the variables $\sigma_{i,j,\gamma}(n)$ are not general, the MPR of order 1 given by Theorem 8.6 may not be minimal. For example, the removal of *implicit places* may reduce the dimension of an order 1 representation. Let us also mention that preliminary results on the problem MinSize for event graphs were proved in [4, 11, 13], precisely the existence of an order 1 representation of dimension $\Gamma_A(\mathcal{R})$ $(\geq \Gamma_B(\mathcal{R}))$.

9. Conclusion and perspectives. Let us summarize the main results obtained. We restricted our attention to recycled systems of UREs. We proved that the optimal solutions for problem MinPeb are the same for games $\mathcal{M}_1$ and $\mathcal{M}_2$; see Theorem 4.11. On the other hand, an optimal solution for game $\mathcal{M}_3$ may be strictly larger than one for the games $\mathcal{M}_1$ and $\mathcal{M}_2$; see the example of Figure 10.

There exist polynomial algorithms to solve problem MinPeb for the games $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$; see sections 4.3 and 6.5. These results have been applied to minimize the number of registers in a digital circuit and to minimize the size of a (max,+) recurrence.

To complete the picture, it would be nice to extend all the previous results to the nonrecycled case. The key results which would make everything else easy to generalize are of two types. Results related to cuts (see section 4.1) and results linking cuts and regular configurations (see section 4.2). For example, is it possible to find a minimal cut which is consecutive (generalization of Lemma 4.7)? Can we find a minimal consecutive cut which is a regular configuration (generalization of Lemma 4.9)? We are currently investigating these different issues.

Appendix A. Scheduling problems. The problem of organizing efficient computations for UREs on parallel computers has been considered by several authors. However, the investigations have often been oriented toward speeding up the execution with little or no consideration for memory requirements. We quickly describe the main results in this area as a way to put our approach into perspective.

A.1. Definition of a schedule. Assume that at time 0, the strictly negative variables $X_i(n), n < 0$, are known. Assume also that each computation of a variable is done in one time unit.

DEFINITION A.1 (schedule). *We define a* schedule *as a set of instants* $\{t_i(n), i \in V, n \in \mathbb{N}\}$ *such that* $t_i(n) \geq t_j(n-\gamma)+1$ *if* $(j,\gamma) \in \Delta_i$ *(with the convention* $t_i(n) = 0\ \forall i \in V, \forall n < 0$*).*

The instant $t_i(n)$ is necessarily larger than the length (number of arcs) of a longest path in $\mathcal{D}$ from column -1 to (i,n). A schedule is said to be *as soon as possible* (asap) if $t_i(n)$ is exactly equal to the length of the longest path from column -1 to (i,n).

A first question that has been addressed is

What is the number of processors required to carry out a computation asap?

This number is often called the *degree of parallelism* of the URE. In general, the solution is given by the size of the maximal anticliques (instead as the minimal cuts of the present paper) in the dependence graph.

Once this question is settled, and provided a sufficient number of processors is available (i.e., larger than the degree of parallelism of the system), another problem is to characterize the asap schedule. This is often called the *basic scheduling problem*. For the asap schedule, it has been proved (see [5, 6, 2]) that

$$\exists N, \forall n \geq N,\ t_i(n) = \lambda_i n + d_i(n), \tag{18}$$

where $\lambda_i \in \mathbb{R}^+$ and $d_i(n), n \geq N$, is a periodic real function. The real $\max_i \lambda_i$ is called the *cycle time* of the system. A schedule satisfying (18) is said to be *linear*. For systems of higher dimension (i.e., when $K = \mathbb{Z}^p,\ p > 1$, in (1)), there are some partial results on how to approximate the asap schedule using linear schedules; see [7, 8].

When the number of available processors is fixed and less than the degree of parallelism of the URE, finding an optimal schedule (i.e., a schedule such that the quantity $\lim_n \max_i t_i(n)/n$ is minimal) becomes NP-hard; see [15].

All the results mentioned above are more or less related to the problem of minimizing the number of processors used. On the other hand, in this paper, we considered the dual problem: How many memory locations are necessary to carry out the computation of an URE, the number of processors being unlimited?

First, we should say that, in general, a computation asap requires a lot of memory. It may not even be bounded when the reduced graph $\mathcal{R}$ is connected but not strongly connected. This makes the alternative of using a smaller memory size attractive. Second, the usual time-space trade-off tells us that some interesting results can be expected to arise when minimizing the memory size.

We have shown in the previous sections how to obtain executions using a minimal memory size. In general the schedules associated with these minimal-memory executions are not asap. A natural question to ask is whether they are very far from the asap execution or not.

For minimal-memory executions, the number of processors needed to carry out the computations will in general be greater than the degree of parallelism. A second natural question is whether it is very far from the degree of parallelism or not. These two questions are now investigated.

A.2. Number of processors and linearity of the schedule. In $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$, rules (R2), (R2b), or (R2c) do not limit a priori the number of moves of type (R3) which are feasible in one step. It corresponds to a computational model where the number of processors available is not limited. However, one can notice a posteriori that the total number of processors needed in the computation is bounded by the maximal number of pebbles $\mathcal{P}(e)$ used during the execution e. Hence, the number of processors needed to implement an execution solving problem MinPeb is kept under control.

In minimal-memory executions, in a single step, we have to compose several of the functions F_i defining the URE to compute the new variables; see section 3.1.

To take into account the "cost" of function composition, we give to each step a duration. We assume that step t lasts $l(t)$ units of time, where $l(t)$ is the length of the longest path in $\mathcal{D}$ joining a node of $\mathcal{A}(t-1)$ to a node of $\mathcal{A}(t)$ and containing no other node in $\mathcal{A}(t-1)$. Note that $l(t)$ is also the length of the longest chain of function compositions performed during step t. This is consistent with the assumption that each function computation requires one unit of time. With this convention, we can now define the schedule associated with a synchronous execution.

DEFINITION A.2. *Let $e = \{\mathcal{A}(t), t \in \mathbb{N}\}$ be an execution of game $\mathcal{M}_1$ (or $\mathcal{M}_2$ or $\mathcal{M}_3$), and let $l(t), t \in \mathbb{N}$, be the time duration of step t. Then the* schedule *of e, $\{\tau_i(n), i \in V, n \in \mathbb{N}\}$ is defined by*

$$\tau_i(n) = \sum_{t=0}^{T} l(t), \text{ where } T = \inf\{t \mid (i,n) \in \mathcal{A}(t)\}\,.$$

One verifies easily that this is indeed a schedule according to Definition A.1. Let us justify Definition A.2. Within one step, the computations are done in parallel. The duration of one step is the one of the longest computation done in the step. After each step, there is a synchronization barrier which makes the duration of the execution to be the sum of the durations of the steps.

One important point is that $\tau_i(n)$ denotes a time instant and not a step of the game. Indeed, there might be a big difference between the time instant and the step at which nodes are computed. Here is an illustration for game $\mathcal{M}_1$ (see also the discussion following Figure 4 in section 3.1). The initial pebbles remain untouched through the whole execution. An additional pebble is used to mark successively all the positive nodes. In such a case, marking a node on column n takes one step and $\Omega(n)$ units of time. Marking all the nodes up to column n requires $\Omega(n)$ steps and $\Omega(n^2)$ units of time. The schedule associated with this execution is *quadratic*.

We proved that minimal-memory executions can always be chosen to be regular (see section 4). It implies that all steps have a constant duration $l(t) = l$. Moreover, exactly one new variable is computed on each line of $\mathcal{D}$ at each step. It implies that the schedule is linear with cycle time l. Therefore, the linear schedules are dominant for our problem. We conclude that the loss in time efficiency, when compared with asap executions, is kept under control.

Among regular executions, one-pass regular executions (game $\mathcal{M}_3$) are dominant in terms of execution time, since they never "lose" time by computing the same variable twice. However, as seen in section 6, they may require more memory than general regular executions.

To summarize, when compared with asap executions, minimal-memory executions may lose a bit in terms of numbers of processors needed and execution speed but may gain a lot in terms of memory size. They provide new insights on the trade-offs between time and space in the computation of a system of UREs.

Appendix B. Sequential executions. In section 3.1, we introduced different variants of pebble games. All these games allow simultaneous moves of the pebbles. Here is another one, defined by rule $\mathcal{M}_4$, which is close to the ones defined in [21, 23]. Here, pebbles are put on the nodes one at a time when all direct predecessors already have a pebble.

B.1. Definition of a sequential game.

$\mathcal{M}_4$: Sequential execution rules.

- (R1) *(initial position).* $\mathcal{A}(0) \subset V \times \mathbb{Z}^-$, $\mathcal{A}(0) \cap (V \times \{0\}) \neq \emptyset$.
- (R2d) *(playing rule).* One step of the game consists in using rule (R3b) at most once followed by any number of moves of type (R4).
- (R3b) *(adding pebbles).* A pebble can be put on an empty node if all the direct predecessors of this node have a pebble.
- (R4) *(removing pebbles).* Remove a pebble from a node.

Let us consider the example of Figure 19. It corresponds to the same URE as in Figure 4.

Here is a possible execution. At step 0, there are pebbles on the nodes $(1, 0), (2, 0)$, and $(3, 0)$, so that (R1) is satisfied. After step t, suppose that there are three pebbles on the nodes $(1, n), (2, n)$, and $(3, n)$, respectively. At step $t + 1$, we can put a pebble on node $(1, n + 1)$ since all its predecessors (i, n) have a pebble (rule (R3b)). At step $t + 2$, we put a pebble on node $(2, n + 1)$. At step $t + 3$, we put a pebble on node $(3, n+1)$ and we remove the pebbles on nodes (i, n) since they are not needed anymore (rule (R3b) then rule (R4) three times). Then we reiterate the previous three steps.

It is impossible to use fewer pebbles. The minimal number of pebbles required to compute the graph under rule $\mathcal{M}_4$ is five, to be compared with the three pebbles needed under rules $\mathcal{M}_1$ or $\mathcal{M}_2$.

As for the computational model of the game, $\mathcal{M}_4$ corresponds to performing sequential computations. It is relevant if we want to model the computation of the

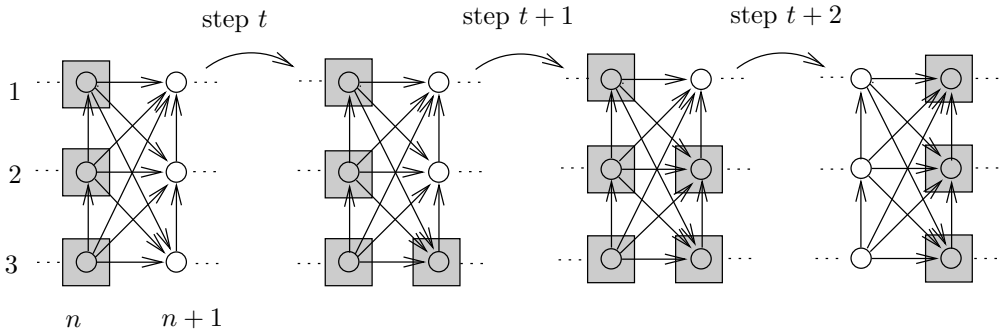


FIG. 19. *Sequential rule $\mathcal{M}_4$. Five pebbles are needed.*

URE using a sequential computer which has a single processor and makes a single computation at each step. The removal of pebbles being a "passive" operation, several such manipulations are allowed in a single step. (The data is not actually removed from the memory; it will just be overwritten the next time this memory location is used.)

In game $\mathcal{M}_4$, a single processor is used. Furthermore, any reasonable execution will compute exactly one new node at each step. Hence, the associated schedule is linear with cycle time $|V|$.

Remark B.1. Game $\mathcal{M}_4$ can be adapted to be played on a finite binary tree (for more details, see the proof of Proposition B.2). In this case, the minimal number of pebbles is known as Strahler's number. This number appears in various fields ranging from hydrology and combinatorics to molecular biology. For a nice survey paper, the reader is referred to Viennot [25].

B.2. Complexity results for $\mathcal{M}_4$. Under game $\mathcal{M}_4$, the problem of determining the minimal number of pebbles to compute a general directed acyclic graph (problem MinPeb-DAG) was proved to be NP-complete by Sethi [23]. We prove a similar result for the problem MinPeb, by showing that an instance of MinPeb-DAG can be embedded into an instance of MinPeb.

PROPOSITION B.2. *Let $\mathcal{D}$ be the dependence graph associated with a system of UREs. Solving problem* MinPeb *for game $\mathcal{M}_4$ is NP-hard. For the subclass of recycled UREs, the same problem is still NP-hard.*

Proof. First we need to define more precisely what is the pebble game $\mathcal{M}_4$ on a DAG. Let V be the set of nodes of a DAG $\mathcal{G}$. A configuration is a subset of V and an execution is a finite sequence of configurations $\{\mathcal{A}_{\mathcal{G}}(t), t = 0, \ldots, L\}$ (where L is the length of the execution). An execution is successful if all the nodes V receive a pebble during the execution. Rules (R2d), (R3b), and (R4) remain the same. Rule (R1) is modified as follows: (R1b) (initial and final position) $\mathcal{A}_{\mathcal{G}}(0) = \mathcal{A}_{\mathcal{G}}(L) = \emptyset$.

Given a DAG $\mathcal{G}$ with set of nodes V, we construct a dependence graph $\mathcal{D}$ with set of nodes $V \times \mathbb{Z}$ in the following way. If there is an arc between nodes i and j in $\mathcal{G}$, then $\forall k \in \mathbb{Z}$, we put an arc between nodes (i,k) and (j,k) and an arc between nodes (i,k) and $(j,k+1)$. We also add the recycling arcs in $\mathcal{D}$ $((i,k) \to (i,k+1)\ \forall i \in V, \forall k \in \mathbb{Z})$. An example is given in Figure 20.

Starting from an execution of the pebble game on $\mathcal{G}$, $\{\mathcal{A}_{\mathcal{G}}(t), t = 0, \ldots, L\}$, we construct an execution of the pebble game on $\mathcal{D}$, $\{\mathcal{A}(t), t \in \mathbb{N}\}$, as follows. The initial configuration is $\mathcal{A}(0) = \{(i,0), i \in V\}$. When a pebble is added on node i in $\mathcal{G}$, we add

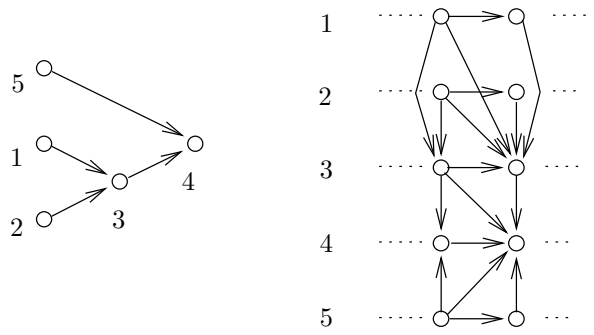


FIG. 20. *Embedding of an arbitrary DAG in a dependence graph.*

a pebble on node $(i, 1)$ in $\mathcal{D}$. When a pebble is removed from node i in $\mathcal{G}$, we remove a pebble from node $(i, 0)$ in $\mathcal{D}$. It follows that at step L, we have $\mathcal{A}(L) = \{(i, 1), i \in V\}$. Then we complete the execution $\{\mathcal{A}(t)\}$ by repeating the same steps periodically, i.e., $\mathcal{A}(t + kL) = \mathcal{A}(t) + k \ \forall t, k \in \mathbb{N}$ (see (6)). If the execution in $\mathcal{G}$ uses p pebbles, the execution in $\mathcal{D}$ requires $p + |V|$ pebbles.

Conversely, from an execution of the game on $\mathcal{D}$, we construct an execution of the game on $\mathcal{G}$ in the following way. By definition, we have $\mathcal{A}(0) \subset \{V \times \mathbb{Z}^-\}$. We set $\mathcal{A}_{\mathcal{G}}(0) = \emptyset$. When a pebble is added on node $(i, 1)$, a pebble is added on node i. When a pebble is removed from node $(i, 0)$, a pebble is removed from node i. If the execution in $\mathcal{D}$ uses P pebbles, the execution in $\mathcal{G}$ requires less than $P - |V|$ pebbles.

One can see that the above transformations map optimal executions into optimal executions (optimal in the sense of problems MinPeb and MinPeb-DAG). Hence, the minimal number of pebbles needed in $\mathcal{D}$ is the minimal number of pebbles needed in $\mathcal{G}$ plus the number of nodes of $\mathcal{G}$. Therefore, the NP-completeness of the problem for DAG (see Sethi [23]) implies that the problem for dependence graphs is at least NP-complete, i.e., NP-hard. □

Proposition B.2 contrasts with the results proved for games $\mathcal{M}_1$, $\mathcal{M}_2$, and $\mathcal{M}_3$, for which a minimal-memory execution was obtained in polynomial time in the recycled case. See section 4.3 (games $\mathcal{M}_1$ and $\mathcal{M}_2$) and section 6.5 (game $\mathcal{M}_3$).

Remark B.3. Let us consider the dependence graph of Figure 20. The execution described in the proof of Proposition B.2 solves problem MinPeb with a number of pebbles which evolves as: $5, 7, 6, 7, 6, 5, \ldots$. The number of pebbles used during an optimal execution is not always constant. Hence, if we consider a nonconnected URE, it is not true that the minimal number of pebbles needed is equal to the sum of the minimal number of pebbles needed for each connected component independently. This contrasts with the situation for games $\mathcal{M}_1, \mathcal{M}_2$, and $\mathcal{M}_3$; see section 2.4.

Acknowledgments. The authors would like to thank Jean-Claude Bermond, Alain Darte, Stéphane Gaubert, Stéphane Perennes, and Mike Robson for numerous discussions and suggestions.

REFERENCES

[1] V. ADLAKHA AND V. KULKARNI, *A classified bibliography of research on stochastic PERT networks:* 1966–1987, Information Systems and Operational Research, 27 (1989), pp. 272–296.

[2] F. Baccelli, G. Cohen, G.J. Olsder, and J.P. Quadrat, *Synchronization and Linearity*, John Wiley, New York, 1992.
[3] F. Baccelli, A. Jean-Marie, and Z. Liu, *A survey on solution methods for task graph models*, in Proceedings of the Second QMIPS Workshop, N. Götz, U. Herzog, and M. Rettelbach, eds., Univ. of Erlangen, Erlangen, Germany, 1993, pp. 163–183.
[4] H. Braker, *Algorithms and Applications in Timed Discrete Event Systems*, Ph.D. thesis, Delft Univ. of Technology, Delft, The Netherlands, 1993.
[5] P. Chretienne, *Les Réseaux de Petri Temporisés*, Ph.D. thesis, Université Paris VI, Paris, 1983.
[6] G. Cohen, D. Dubois, J.P. Quadrat, and M. Viot, *A linear system-theoretic view of discrete-event processes and its use for performance evaluation in manufacturing*, IEEE Trans. Automat. Control, AC-30 (1985), pp. 210–220.
[7] A. Darte, L. Khachiyan, and Y. Robert, *Linear scheduling is nearly optimal*, Parallel Process. Lett., 1 (1991), pp. 73–81.
[8] A. Darte and F. Vivien, *Automatic Parallelization Based on Multi-Dimensional Scheduling*, Technical report 24, Laboratoire d' Informatique du Parallélisme, Ecole Normale Supérieure, Lyon, France, 1994.
[9] L.R. Ford and D.R. Fulkerson, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1962.
[10] S. Gaubert, P. Butkovič, and R. Cuninghame-Green, *Minimal* (max,+) *realization of convex sequences*, SIAM J. Control Optim., 36 (1998), pp. 137–147.
[11] B. Gaujal, *Parallélisme et simulation de systèmes à événements discrets*, Ph.D. thesis, Université de Nice-Sophia Antipolis, France, 1994.
[12] B. Gaujal, A. Jean-Marie, and J. Mairesse, *Minimal Representation of Uniform Recurrence Equations*, Technical report RR-2568, INRIA, Sophia-Antipolis, France, 1995.
[13] B. Gaujal and A. Jean Marie, *Computational issues in stochastic recursive systems*, in Idempotency, J. Gunawardena, ed., Cambridge University Press, Cambridge, UK, 1998, pp. 209–230.
[14] M. Gondran and M. Minoux, *Graphs and Algorithms*, John Wiley, New York, 1986.
[15] C. Hanen and A. Munier, *A study of the cyclic scheduling problem on parallel processors*, Discrete Appl. Math., 57 (1995), pp. 167–192.
[16] R. Karp, R. Miller, and S. Winograd, *The organization of computations for uniform recurrence equations*, J. Assoc. Comput. Mach., 14 (1967), pp. 563–590.
[17] C. Leiserson and J. Saxe, *Retiming synchronous circuitry*, Algorithmica, 6 (1991), pp. 5–35.
[18] J. Mairesse, *Stabilité des systèmes à événements discrets stochastiques. Approche algébrique*, Ph.D. thesis, Ecole Polytechnique, Paris, 1995 (in English).
[19] T. Murata, *Petri nets: Properties, analysis and applications*, Proceedings of the IEEE, 77 (1989), pp. 541–580.
[20] G.J. Olsder, *On the characteristic equations and minimal realizations for discrete-event dynamic systems*, in Proceedings of the Seventh International Conference on Anal. and Optim. of Systems, Lecture Notes in Control and Inform. Sci. 83, Springer-Verlag, New York, 1986, pp. 189–201.
[21] N. Pippenger, *Advances in pebbling*, in Proceedings of the International Colloquium on Automata, Languages, and Programming, Lecture Notes in Comput. Sci. 140, Springer-Verlag, New York, 1982, pp. 407–417.
[22] J.H. Reif, ed., *Synthesis of Parallel Algorithms*, Morgan-Kaufmann, San Francisco, 1993.
[23] R. Sethi, *Complete register allocation problems*, SIAM J. Comput., 4 (1975), pp. 226–248.
[24] N. Shenoy and R. Rudell, *Efficient implementation of retiming*, in Proceedings of the 1994 International Conference on Computer Aided Design, San Jose, 1994.
[25] G.X. Viennot, *Trees*, in Mots, M. Lothaire, ed., Hermès, Paris, 1990, pp. 265–297.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1739–1772

APPROXIMATION ALGORITHMS FOR CURVATURE-CONSTRAINED SHORTEST PATHS*

PANKAJ K. AGARWAL† AND HONGYAN WANG†

Abstract. Let B be a point robot in the plane, whose path is constrained to have curvature of at most 1, and let Ω be a set of polygonal obstacles with n vertices. We study the collision-free, optimal path-planning problem for B. Given a parameter ε, we present an $O((n^2/\varepsilon^4)\log n)$-time algorithm for computing a collision-free, curvature-constrained path between two given positions, whose length is at most $(1+\varepsilon)$ times the length of an optimal path, provided it is robust. (Roughly speaking, a path is robust if it remains collision-free even if certain positions on the path are perturbed). Our algorithm thus runs significantly faster than the previously best known algorithm by Jacobs and Canny whose running time is $O((\frac{n+L}{\varepsilon^2})^2 + n^2(\frac{n+L}{\varepsilon^2})\log n)$, where L is the total edge length of the obstacles. More importantly, the running time of our algorithm does not depend on the size of obstacles. The path returned by this algorithm is not necessarily robust. We present an $O((n^{2.5}/\varepsilon^4)\log n)$-time algorithm that returns a robust path whose length is at most $(1+\varepsilon)$ times the length of an optimal path, provided it is robust. We also give a stronger characterization of curvature-constrained shortest paths, which, apart from being crucial for our algorithm, is interesting in its own right. Roughly speaking, we prove that, except in some special cases, a shortest path touches obstacles at points that have a visible vertex nearby.

Key words. robotics, motion planning, configuration space, approximation algorithms

AMS subject classifications. 68W25, 68W40, 68T40

PII. S0097539796307790

1. Introduction.

The *path-planning* problem involves planning a collision-free path for a robot moving amid obstacles. This is one of the main problems in robotics and has been widely studied (see, e.g., the books by Latombe [32] and by Hopcroft, Schwartz, and Sharir [26] and the survey papers by Schwartz and Sharir [47, 48] and Halperin, Kavraki, and Latombe [25]). In the simplest form of the motion planning, given a moving robot B, a set O of obstacles, and a pair of placements I and F of B, we wish to find a continuous, collision-free path for B from I to F. This problem is PSPACE-complete [11, 44], and efficient algorithms have been developed for several special cases [48]. Most of these algorithms, however, do not take into account the dynamic constraints (for instance, velocity/acceleration bounds, curvature bounds), the so-called *nonholonomic constraints*, of a real robot imposed by its physical limitations. Although there has been considerable recent work in the robotics literature (see [3, 6, 8, 10, 17, 23, 28, 30, 31, 33, 34, 36, 37, 38, 40, 45, 51, 52] and references therein) on nonholonomic motion-planning problems, relatively little theoretical work has been done on these important problems, because they are considerably harder than holonomic motion-planning problems.

In holonomic motion planning, the placement of a robot with k degrees of freedom is determined by a tuple of k (typically real) parameters, each describing one degree

*Received by the editors August 6, 1996; accepted for publication (in revised form) September 8, 2000; published electronically January 16, 2001.

http://www.siam.org/journals/sicomp/30-6/30779.html

†Department of Computer Science, Box 90129, Duke University, Durham, NC 27708-0129 (pankaj@cs.duke.edu, hongyan@cs.duke.edu). Work by the first author was supported by a National Science Foundation grant CCR-93–01259, by an Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award and matching funds from Xerox Corporation, and by a grant from the U.S.–Israeli Binational Science Foundation. Work by the second author was supported by NSF grant NSF-IRI-91-00681, Rome Labs contracts F30602-94-C-0037, ARPA/SISTO contracts N00014-91-J-1985, and N00014-92-C-0182 under subcontract KI-92-01-0182.

of freedom. The set of all placements is called the *configuration space*, and the set of placements at which the robot does not intersect any obstacles is called the *free configuration space*. There exists a path between an initial placement and a final placement if and only if the two placements lie within the same (path-) connected component of the free configuration space. This is not necessarily true if the robot has to obey nonholonomic constraints. In nonholonomic motion planning, usually a placement is not enough to describe the robot. Instead, a robot is completely described by its *state*, consisting of the k parameters and their derivatives (see [32] for a more detailed discussion), which makes the problem considerably harder.

In this paper, we study the path-planning problem for a point robot whose path is constrained to have curvature of at most 1. More formally, given a continuous differentiable path $P : I \to \mathbb{R}^2$ parameterized by arc length $s \in I$, the *average curvature* of P in the interval $[s_1, s_2] \subseteq I$ is defined by $\|P'(s_1) - P'(s_2)\|/|s_1 - s_2|$. We require that the average curvature of the robot's path is at most 1 in every interval. This restriction corresponds naturally to constraints imposed by a steering mechanism on a *car-like* robot (see [32] for a formal description), because the path traced out by the middle point between the two rear wheels has an instantaneous curvature of $\frac{1}{\lambda}\tan\phi$, where ϕ is the steering angle and λ is a parameter of the car. The maximum curvature of the path is therefore $\frac{1}{\lambda}\tan\phi_{\max}$, assuming $\phi_{\max}$ is the maximum steering angle.

Dubins [21] was perhaps the first to study the curvature-constrained shortest paths. He proved that, in the absence of obstacles, a curvature-constrained shortest path from any start position to any final position consists of at most three segments, each of which is either a straight line or an arc of a unit-radius circle. Reeds and Shepp [43] extended this obstacle-free characterization to robots that are allowed to make reversals. Using ideas from control theory, Boissonnat, Cerezo, and Leblond [7] and Sussmann and Tang [50] gave an alternative proof for both cases, and recently Sussmann [49] was able to extend the characterization for the 3-dimensional case. In the presence of obstacles, Fortune and Wilfong [22] gave a $2^{m^{O(1)}}$-time algorithm, where m is the number of bits required to specify the coordinates of all vertices of obstacles, to decide whether a feasible path exists from an initial position to a final position; their algorithm cannot find a feasible path in all the cases. Jacobs and Canny [12, 27] gave an $O((\frac{n+L}{\varepsilon^2})^2 + n^2(\frac{n+L}{\varepsilon^2})\log n)$-time algorithm (this and the next time complexities are based on [12] and are more accurate than the ones given in [27]) that finds an approximate path whose length is no more than $(1+\varepsilon)$ times the length of a shortest *ε-robust* path, where L is the total edge length of the obstacles. (Informally, a path is ε-robust if perturbations of certain positions along the path by $\pm\varepsilon/2$—in distance or in angle—do not violate the feasibility of the path; a formal definition is given in section 2.3.) The path returned by this algorithm is not necessarily robust. They also presented an $O(n^4 \log n + (\frac{n+L}{\varepsilon^2})^2)$-time algorithm that computes an $(\varepsilon/2)$-robust path whose length is no more than $(1+\varepsilon)$-times the length of an optimal ε-robust path. For the restricted case of *moderate obstacles*, i.e., when the obstacles are convex and the curvature of their boundary is also bounded by 1, Agarwal, Raghavan, and Tamaki [2] gave efficient approximation algorithms. Boissonnat and Lazard [9] gave a polynomial-time algorithm for computing the exact shortest paths for the case when the edges of the obstacles are circular arcs of unit radius and straight line segments. Wilfong [51] studied a restricted problem in which the robot must stay on one of m line segments (thought of as "lanes"), except to turn between lanes. For a scene with n obstacle vertices, his algorithm preprocesses the scene in time

$O(m^2(n^2 + \log m))$, following which queries are answered in time $O(m^2)$. There has also been work on computing curvature-constrained paths when B is allowed to make reversals [3, 30, 31, 35, 39]. Other, more general, dynamic constraints are considered in [13, 14, 15, 20, 41, 46].

2. Our model and results. Let B be a point robot. Let Ω be a set of disjoint polygonal obstacles, with a total of n vertices. We refer to edges and vertices of Ω as obstacle *features*. We assume that the edges of Ω are relatively open, so they do not contain their endpoints. A *position* X for B is a pair $(\text{LOC}(X), \text{VEC}(X))$, where $\text{LOC}(X)$ is a point in the plane, representing the location of the robot, and $\text{VEC}(X)$ is an angle between 0 and 2π, representing its orientation. Abusing the notation a little, we will use X to denote $\text{LOC}(X)$ as well if either $\text{VEC}(X)$ is not important or $\text{VEC}(X)$ is obvious from the context. We will use ℓ_X to denote the (oriented) line passing through $\text{LOC}(X)$ in direction $\text{VEC}(X)$, and C_X^+ (resp., C_X^-) to denote the counter-clockwise (resp., clockwise) oriented unit-radius circle tangent to ℓ_X at $\text{LOC}(X)$ whose center lies to the left (resp., right) of ℓ_X. We say that a position X lies on an obstacle vertex if $\text{LOC}(X) = v$ and on an obstacle edge e if $\text{LOC}(X) \in e$ and ℓ_X contains e. A *path* is an oriented curve; $\|\cdot\|$ denotes the length of a path. A path is called *legal* if its average curvature is at most 1 in every interval.

If I and F are the initial and final positions, then we regard $\text{LOC}(I)$ and $\text{LOC}(F)$ as point obstacles. A legal path is *feasible* (with respect to Ω) if it does not intersect the interior of any obstacle of Ω (see Figure 2.1). A path Π from a position X to another position Y is *optimal* if it is a feasible path with the minimum arc length, where the minimum is taken over all feasible paths from X to Y. (The minimum always exists; for a proof, see [27].)

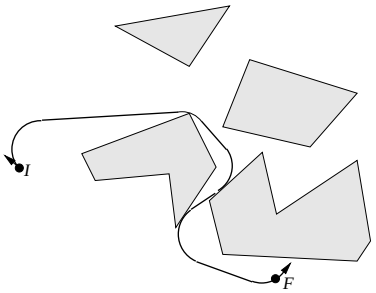


FIG. 2.1. *An example of a feasible path.*

2.1. Dubins paths and canonical paths. We call a nonempty subpath of a feasible path Π a *C-segment* (resp., *L-segment*) if it is a circular arc of unit radius (resp., line segment) and maximal. Suppose Π consists of a C-segment, an L-segment, and a C-segment; then we say that Π is of type CLC. This notion can be generalized to an arbitrarily long sequence. If the orientation of a C-segment—counter-clockwise or clockwise—is important, we use C^+ (resp., C^-) to denote a counter-clockwise (resp., clockwise) oriented C-segment, so C^+LC^- denotes a path consisting of a counter-clockwise C-segment, an L-segment, followed by a clockwise C-segment. Dubins [21] proved the following result.

THEOREM 2.1 (Dubins [21]). *In an obstacle-free environment, an optimal path between any two positions is of type CCC or CLC, or a substring thereof (see Figure 2.2).*

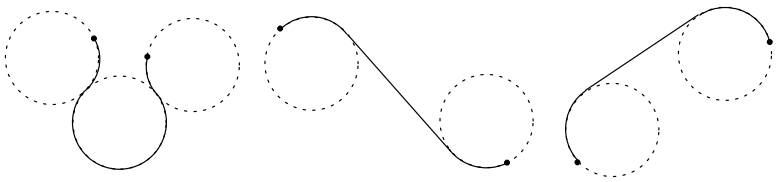


FIG. 2.2. *Examples of Dubins paths.*

We will refer to such paths as *Dubins paths*. If we fix the orientation of initial and final C-segments, then there is at most one CLC (resp., CCC) path between two positions.[1]

Therefore, there are at most four types of CLC paths, namely, C^+LC^+, C^+LC^-, C^-LC^+, and C^-LC^-, and at most two types of CCC paths, namely, $C^+C^-C^+$ and $C^-C^+C^-$. Note that not all types of path exist between two positions X and Y. For example, if the distance between the centers of C_X^+ and C_Y^- is less than 2 (i.e., the two circles intersect), then the C^+LC^- path does not exist from X to Y. Similarly, if the distance between the centers of C_X^+ and C_Y^+ is greater than 4, the $C^+C^-C^+$-type path does not exist from X to Y. Let

$$\mathcal{T} = \{C^+LC^+, C^+LC^-, C^-LC^+, C^-LC^-, C^+C^-C^+, C^-C^+C^-\}.$$

For a Dubins path Π, we define $\sigma(\Pi) \in \mathcal{T}$ to be the string that characterizes Π. If the length of a C- or an L-segment is zero, then $\sigma(\Pi)$ is not uniquely defined; in this case we choose one of them, depending on the context.

Fortune and Wilfong [22] proved that in the presence of obstacles an optimal path consists of a finite sequence of Dubins paths. We call a feasible path from a position I to a position F *canonical* if it consists of a finite sequence $\Pi_1||\cdots||\Pi_k$ of feasible paths, where each Π_i is a Dubins path from a position X_{i-1} to a position X_i, such that $X_0 = I$, $X_k = F$, and, for $0 < i < k$, $\text{LOC}(X_i) \in \partial\Omega$. Since we regard $\text{LOC}(I)$ and $\text{LOC}(F)$ as point obstacles, X_0 and X_k also lie on obstacle boundaries. Jacobs and Canny [27] proved the following property of optimal paths.

THEOREM 2.2 (Jacobs–Canny [27]). *Given a set of polygonal obstacles Ω, an initial position I, and a final position F, every optimal path from I to F is a canonical path.*

This theorem implies that an optimal path is a finite sequence of C- and L-segments, so it can be represented as a finite string over the alphabet $\{C, L\}$.

2.2. Parametrizing canonical paths. Let X be a position lying on the boundary of an obstacle. For the purpose of parametrizing canonical paths, we regard each obstacle edge pq as two oriented edges $\overrightarrow{pq}$ and $\overrightarrow{qp}$. If X lies on an oriented edge $\vec{e} = \overrightarrow{pq}$, then $\text{VEC}(X)$ is fixed, but $\text{LOC}(X)$ may lie anywhere on e. We can specify $\text{LOC}(X)$, and thus X, by a real number $x \in [0, \|e\|]$, which specifies the distance between $\text{LOC}(X)$ and p; see Figure 2.3(a). Similarly, if X lies on an obstacle vertex v, then $\text{LOC}(X)$ is fixed and $\text{VEC}(X)$ may vary. In this case, we can represent X by a real value x in an angular interval $[\theta_l, \theta_r]$, where θ_l, θ_r are the orientations of the edges adjacent to v; see Figure 2.3(b). (We could also parametrize X by a real value in a

[1] Actually, there are two CCC paths if we fix the orientations of the initial and final C-segments. But the length of the middle C-segment is less than π in one of the two paths. It is known that such a path cannot be an optimal Dubins path. Therefore, we will not consider this CCC path and assume that there is only one CCC path.

FIG. 2.3. *Parametrizing positions on obstacle boundaries.*

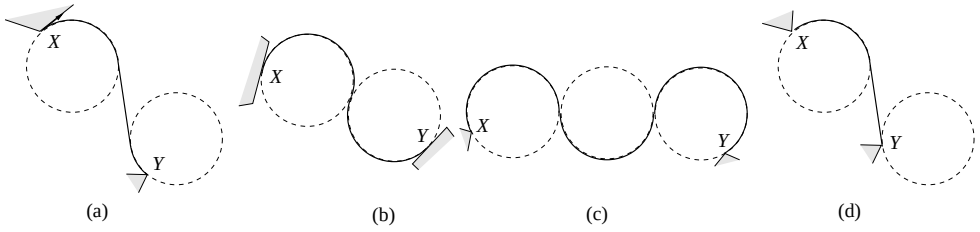


FIG. 2.4. *A few examples of singular points.*

linear interval $[0, \theta_r - \theta_l]$ or $[0, 2\pi + \theta_r - \theta_l]$ depending on whether $0 \in [\theta_l, \theta_r]$.) For two orientations $\phi_1, \phi_2 \in [\theta_l, \theta_r]$, we say that $\phi_1 \leq \phi_2$ if $\theta_l, \phi_1, \phi_2, \theta_r$ appear in that order in the counter-clockwise direction. Hence, a position X lying on an obstacle feature ϕ can be represented as a real parameter $\psi(X)$ that lies in an interval $[l_\phi, r_\phi]$; we will refer to this interval as the *feasible domain* of the feature ϕ. If $X = I$ (resp., F), then its domain is defined to be the singleton $[\text{VEC}(I), \text{VEC}(I)]$ (resp., $[\text{VEC}(F), \text{VEC}(F)]$).

Let Π be a Dubins path from a position X lying on an obstacle feature ϕ_X to another position Y lying on an obstacle feature ϕ_Y. Then Π can be represented by the 3-tuple $\psi(\Pi) = (\psi(X), \psi(Y), \sigma(\Pi))$. Let $\mathcal{I}_1, \mathcal{I}_2$ be the feasible domains of two obstacle features ϕ_1 and ϕ_2, respectively. Then $\mathcal{I}_1 \times \mathcal{I}_2 \times \mathcal{T}$ is the *configuration* space of all Dubins paths from a position on ϕ_1 to another position on ϕ_2. The configuration space of all Dubins paths from a vertex to another vertex is a subset of $\mathbb{S}^1 \times \mathbb{S}^1 \times \mathcal{T}$. By parametrizing $\psi(X), \psi(Y)$ more carefully, the configuration space can be defined as a subset of $\mathbb{R}^2 \times \mathcal{T}$, which will be useful in the following discussion. We can define the configuration space for all other cases as well. As mentioned earlier, not every point in the configuration space defines a path from X to Y.

Consider a point (τ_1, τ_2, σ) in the configuration space and vary the pair (τ_1, τ_2), keeping σ fixed; then the corresponding Dubins path deforms continuously except at certain values, at which the path either ceases to exist or the length of one of the C-segments becomes 0 or 2π, i.e., if one of the events occurs:

(i) $\psi(X)$ or $\psi(Y)$ is an endpoint of its feasible domain (e.g., τ_1 is an endpoint of $\psi(X)$ in Figure 2.4(a));
(ii) $\sigma = C^+LC^-$ and $C^+(X)$ and $C^-(Y)$ are tangent to each other (see Figure 2.4(b));
(iii) $\sigma = C^-LC^+$ and $C^-(X)$ and $C^+(Y)$ are tangent to each other;
(iv) $\sigma = CCC$ and the distance between the centers of the first and the last C-segments is 4 (see Figure 2.4(c)); and
(v) the length of a C-segment is 0 or 2π (e.g., the length of the final C-segment is 0 in Figure 2.4(d)).

A point in a configuration space (or path) satisfying any of the above five conditions

is called *singular*; see Figure 2.4 for a few examples of singular points. We call a point $\tau = (\tau_1, \tau_2, \sigma)$ in the configuration space *infeasible* if the corresponding path is either not defined or intersects the interior of an obstacle; otherwise, it is called *feasible*. The geometry of paths, configuration space of paths, and the topology of infeasible, free, and singular points are studied in detail by Jacobs and Canny [27].

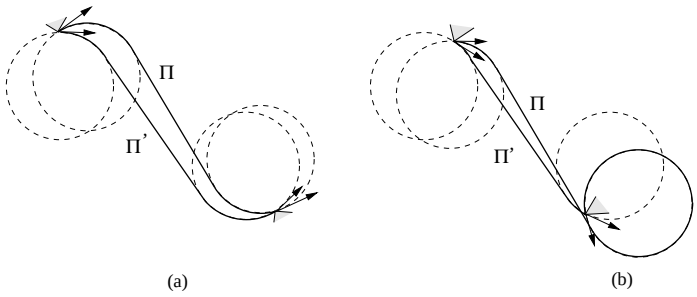


FIG. 2.5. (a) Π *and* Π' *are neighbors;* (b) Π *and* Π' *are not neighbors.*

2.3. Robust paths. Fix two obstacle features ϕ_1, ϕ_2, and let $[l_i, r_i]$ be the feasible domain of ϕ_i. Let $\tau = (\tau_1, \tau_2, \sigma)$ and $\tau' = (\tau_1', \tau_2', \sigma)$ be two points in the configuration space of $\phi_1 \times \phi_2$, i.e., $\tau_i, \tau_i' \in [l_i, r_i]$ for $i = 1, 2$. Let $R(\tau, \tau') = \{(\xi_1, \xi_2, \sigma) \mid \min\{\tau_i, \tau_i'\} < \xi_i < \max\{\tau_i, \tau_i'\}\}$ be the (open) rectangle in the configuration space formed by τ and τ'. We call τ and τ' *neighbors* if none of the points in the rectangle $R(\tau, \tau')$ is singular or infeasible (see Figure 2.5). If τ and τ' are neighbors, then τ can be deformed continuously to any point in the square without intersecting the interior of obstacles. The analysis in section 5 (see Remark 5.3 (ii)) will show that if τ and τ' are neighbors and $|\tau_i - \tau_i'| \leq \delta$ for some $0 \leq \delta \leq 1$, then the paths corresponding to points in $R(\tau, \tau')$ lie in a tube of width $O(\sqrt{\delta})$ drawn around the path corresponding to the center of $R(\tau, \tau')$.

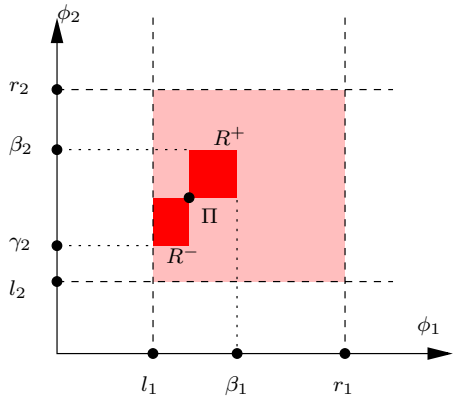


FIG. 2.6. δ^{++}*- and* δ^{--}*-robust paths.* Π *is* δ^{++}*-robust (resp.,* δ^{--}*-robust) if the square* R^+ *(resp.,* R^-*) does not contain any singular or infeasible point.*

Let Π be a Dubins path from an obstacle feature ϕ_1 to another one ϕ_2, with $\psi(\Pi) = (\tau_1, \tau_2, \sigma)$, and let $0 < \delta < 1/2$ be a real parameter. For $i = 1, 2$, let $[l_i, r_i]$ be the feasible domain of ϕ_i; set $\beta_i = \min\{r_i, \tau_i + \delta\}$ and $\gamma_i = \max\{l_i, \tau_i - \delta\}$; see

Figure 2.6. We call Π δ^{++}*-robust* if $\psi(\Pi)$ and $(\beta_1, \beta_2, \sigma)$ are neighbors and δ^{--}*-robust* if $\psi(\Pi)$ and $(\gamma_1, \gamma_2, \sigma)$ are neighbors. We define δ^{+-}*-robust,* δ^{-+}*-robust* paths in a similar manner. Intuitively, Π is δ^{++}-robust if for $0 \leq h_1, h_2 \leq \delta$, the paths $(\tau_1 + h_1, \tau_2 + h_2, \sigma)$ are homeomorphic to Π and remain in a neighborhood of Π, assuming that $\tau_i + \delta \leq r_i$. If $r_i - \tau_i < \delta$, i.e., Π lies near the boundary of the configuration space of $\phi_1 \times \phi_2$, then one has to be more careful and vary $h_i \in [\tau_i, r_i]$. The notion of "directed" robustness and the boundary conditions will be important in the subsequent discussion. (Jacobs and Canny call Π δ-robust if the points $(\tau_1 - \delta/2, \tau_2 - \delta/2, \sigma)$ and $(\tau_2 + \delta/2, \tau_2 + \delta/2, \sigma)$ are neighbors. Their definition does not take the boundary conditions into account, so the points near the boundary of feasible domains are not robust under their definition.)

Now, let Π be a canonical path consisting of Dubins paths $\Pi_1 \| \Pi_2 \| \cdots \| \Pi_k$. Jacobs and Canny call Π δ-robust if each Π_i is $(\delta/2)^{\sigma_1\sigma_2}$-robust for every pair $\sigma_1, \sigma_2 \in \{+, -\}$. (More precisely, they call Π δ-robust if each Π_i is δ-robust under their definition of δ-robustness.) A weakness of their definition is that if one of Π_i is close to a singular path, then Π_i is not robust even if no obstacle is in the neighborhood of Π_i. For example, the canonical Dubins path Π in Figure 2.7 is not robust under their definition of δ-robustness. We circumvent this problem by using a somewhat weaker definition. We call Π δ*-robust* if there exist $\sigma_i \in \{+, -\}$ for $0 \leq i \leq k$, such that Π_i is $\delta^{\sigma_{i-1}\sigma_i}$-robust.

Intuitively, Π being robust means that the tube of width $O(\sqrt{\delta})$ around Π does not intersect any obstacle except those which touch Π, and they intersect the tube only in the neighborhood of the point at which they touch Π.

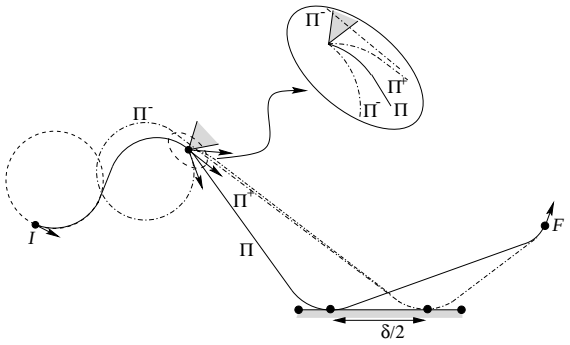


FIG. 2.7. *A robust path consisting of three canonical Dubins paths. The second canonical subpath,* $\Pi = (\tau_2, \tau_3, C^-LC^+)$*, is* δ^{++}*-robust but not* δ^{-+}*-robust because* $\Pi^- = (\tau_2 - \delta/2, \tau_3 + \delta/2, C^-LC^+)$ *and* Π *are not neighbors.*

2.4. Our results. We first obtain a stronger characterization of optimal paths (section 3.1), which besides being interesting in its own right is critical for our algorithm. Roughly speaking, we prove that, except in some very special cases, an optimal path touches an edge e of Ω only at points near a vertex v (v is not necessarily one of the endpoints of e), such that these points are visible from v. In other words, we can ignore the portions of the edges of Ω that are not near any vertex of Ω. This property enables us to develop an algorithm whose running time does not depend on the length of the boundary of Ω.

We then describe an efficient algorithm for computing a feasible path between two given positions (section 4) whose length is at most $(1 + \varepsilon)$ times that of an optimal

path, assuming that there exists an optimal path from I to F that is robust. More precisely, given an obstacle environment Ω with n vertices, two positions I and F, and a parameter ε, we present an $O((n^2/\varepsilon^4)\log n)$-time algorithm that computes a feasible path from I to F whose length is at most $(1+\varepsilon)$ times the length of an optimal ε-robust path from I to F, provided that an optimal path from I to F is robust. Compared with the Jacobs–Canny algorithm [27], our algorithm is not only considerably faster in terms of the complexity of Ω, but, more importantly, the running time is independent of L, the total edge length of Ω. The second improvement is rather significant because one cannot assume L to be small. For unconstrained optimal path planning, one can scale down the environment arbitrarily (to reduce the value of L), compute a shortest path in the scaled environment, and then scale the optimal path back. But this scaling idea does not work for curvature-constrained shortest paths, as the scaling will change the curvature of the path as well. The path returned by our algorithm is not necessarily robust. We can, however, modify the algorithm to compute an $(\varepsilon/2)$-robust path in time $O((n^{2.5}/\varepsilon^4)\log n)$, whose length is no more than $(1+\varepsilon)$ times the length of an optimal path, assuming that it is ε-robust. In fact, an $O(n^{2+\gamma}/\varepsilon^4)$-time algorithm can be obtained, using the recent range-searching data structures [1], but we will not discuss this improvement in this paper.

3. Characterization of optimal paths. In this section we characterize optimal paths in the plane in the presence of obstacles. Theorem 2.2 implies that an optimal path Π is a canonical path. However, even if we specify the sequence of obstacle edges and vertices that Π touches, there are infinitely many paths touching the same sequence of edges and vertices—an optimal path can touch a vertex at an arbitrary orientation, or it can touch an arbitrary point of an edge. Jacobs and Canny [27] observe that if one is interested only in computing $(1+\varepsilon)$-approximate paths, one can choose a finite set of orientations at which a path can touch a vertex and can also choose a finite set of points on each edge e at which a path can touch e. The number of points chosen on an edge is proportional to the length of the edge. That is why the running time of their algorithm depends on the total length of edges of Ω. We circumvent this problem by proving a stronger property of optimal paths.

A feasible C-segment is called *free* if it does not intersect the interior of Ω, *anchored* if it touches $\partial\Omega$ at two or more points (recall that we assume $\text{LOC}(I), \text{LOC}(F)$ to be point obstacles), and *semifree* if it touches $\partial\Omega$ at exactly one point. By Theorems 2.1 and 2.2, an optimal path cannot have two consecutive free C-segments. Moreover, there are only finite number of circles that touch $\partial\Omega$ at two or more points (assuming that there are no two edges parallel at distance 1), so there is only a finite number of circles that may contain anchored C-segments. We thus need a better understanding of semifree C-segments.

The following theorem states the main result of this section.

THEOREM 3.1. *If w is a semifree C-segment of an optimal path, then there is a vertex v of Ω within distance 15 from the intersection point ξ of w and $\partial\Omega$ that is visible from ξ (i.e., the segment ξv does not intersect the interior of Ω).*

3.1. Proof of Theorem 3.1. We will prove the theorem by a sequence of lemmata. We begin with a few notation and simple observations. For an oriented path Π and two points $a, b \in \Pi$, let $\Pi[a,b]$ denote the subpath of Π from a to b. For an unoriented circle C and two points $a, b \in C$, let $C[a,b]$ denote the arc of C from a to b in the clockwise direction. However, if C contains a C-segment of a path, then we assume that C is oriented the same way as the C-segment.

Recall that, by our convention, $\text{LOC}(I)$ and $\text{LOC}(F)$ are the vertices of Ω. If ξ lies within distance 15 from one of the endpoints of the obstacle edge that contains ξ, then there is nothing to prove. So assume that ξ lies in the interior of an edge $e = pq$ and $d(p, \xi), d(q, \xi) > 15$. Without loss of generality, assume that e lies on the x-axis, that w lies below the x-axis, and that w is oriented clockwise.

We divide the proof of Theorem 3.1 into a few cases. For each case, we prove the existence of a closed, simply connected region R satisfying the following properties.

- P1. R lies on or below the x-axis, R contains ξ, and either ∂R contains a vertex of Ω or the interior of R contains a point of $\partial\Omega$.
- P2. If an edge $g \in \Omega$ intersects the interior of R, it crosses ∂R in at most one point, which implies that at least one of the endpoints of g lies inside R.
- P3. $d(\xi, x) \leq 15$ for all points $x \in R$.

See Figure 3.1 for an example of R. P1–P3 (i.e., P1, P2, and P3) together imply that there is a vertex of Ω within distance 15 from ξ. In order to prove that there is also a vertex within distance 15 from ξ that is visible from ξ, we introduce R^*, the convex hull of R.

LEMMA 3.2. *If a simply connected region R satisfies* P1–P3*, then R^* also satisfies* P1–P3.

Proof. P1 is obvious. If an obstacle edge g crosses ∂R^* at two points, then, using the fact that R is simply connected, one can prove that g also crosses ∂R at two points, which contradicts property P2 of R. Hence, g crosses ∂R^* in at most one point, thereby proving P2.

For any point $x \in R^*$, the ray emanating from ξ and passing through x intersects ∂R^* at one point other than ξ itself, say, y (y may be identical to x). If $y \in R$, property P3 follows since $d(\xi, x) \leq d(\xi, y) \leq 15$. Otherwise, y lies in the interior of a line segment, both of whose endpoints, say, u and v, lie on ∂R. Property P3 also follows since $d(\xi, x) \leq d(\xi, y) \leq \max\{d(\xi, u), d(\xi, v)\} < 15$. This completes the proof of the lemma. □

LEMMA 3.3. *Let R^* be a convex region satisfying* P1–P3*. Then there exists a vertex v of Ω within distance* 15 *from ξ that is visible from ξ.*

Proof. If no obstacle edge intersects the interior of R^*, the lemma is obvious because an obstacle vertex lying on ∂R^* is visible from ξ. Otherwise, for each obstacle edge e that intersects the interior of R^*, clip it within R^*; $e \cap R^*$ is a segment. Let E denote the set of clipped segments. By property P1, $E \neq \emptyset$. We define the following partial ordering on the segments of E. We say that $e_i \prec e_j$ ($e_i, e_j \in E$) if any ray emanating from ξ and intersecting the relative interiors of both e_i and e_j intersects e_i before intersecting e_j. (That is, viewed from ξ, e_j cannot occlude any portion of e_i.) Using the fact that the segments of E are disjoint, it can be shown that $\prec$ induces a partial ordering on E; see, e.g., [18, 24]. Moreover, this partial ordering can be extended to a total ordering. By the definition of the ordering, every point on the first segment in this ordering is visible from ξ. But one of the endpoints of this segment, say, v, is a vertex of Ω (because of P2), so we have found an obstacle vertex v within distance 15 from ξ that is visible from ξ. □

The following simple lemma, which will be useful in many cases, follows from Lemma 3.2.

LEMMA 3.4. *Let α and β be two points on an optimal path Π from I to F such that α, ξ, and β appear in that order along Π, such that $\Pi[\alpha, \beta]$ lies below or on the x-axis, and such that $d(\xi, x) \leq 15$ for all $x \in \Pi[\alpha, \beta]$. Let R be the convex hull of $\Pi[\alpha, \beta]$. If α is an obstacle vertex or if it lies in the interior of an obstacle edge whose*

supporting line intersects $\Pi[\alpha, \beta]$ *at a point other than* α*, then* R *satisfies* P1–P3. *See Figure* 3.1.

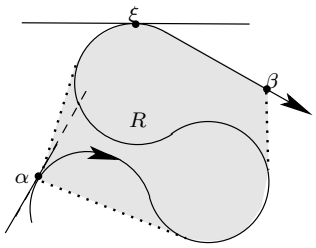


FIG. 3.1. $\Pi[\alpha, \beta]$ *and its convex hull; shaded region is* R.

Proof. Following an argument similar to the one in the proof of Lemma 3.2, we can show that R satisfies P3. Suppose an obstacle edge g intersects the interior of R; then let γ be the intersection of g with R. If both endpoints of γ lie on the boundary of ∂R, then a continuity argument implies that g crosses $\Pi[\alpha, \beta]$, which is impossible. Therefore, g crosses ∂R in at most one point, and thus R satisfies P2. If α is a vertex, R satisfies P1. Otherwise, the obstacle edge containing α lies in the interior of R near α, because the line supporting this edge intersects the interior of R. Hence, R satisfies P1 also. This completes the proof. □

Since we regard $\text{LOC}(I)$ and $\text{LOC}(F)$ as vertices of Ω and assume that ξ is not a vertex, the semifree C-segment w is not the first or the last segment of Π. Let w^- (resp., w^+) be the segment of Π immediately before (resp., after) w.

Using a perturbation argument similar to the one used in [2, 9] (see Figure 3.2), one can easily prove the following lemma, whose proof is omitted from here.

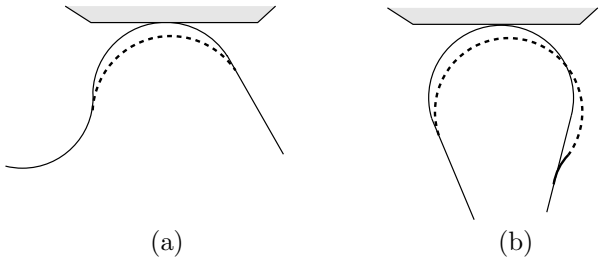


FIG. 3.2. *Length reducing perturbations:* (a) $\|w\| \leq \pi$; (b) *both* w^- *and* w^+ *are L-segments.*

LEMMA 3.5. *If* w *is neither the first segment nor the last segment of* Π*, then* (i) $\|w\| > \pi$*, and* (ii) *either* w^- *or* w^+ *is a C-segment.*

Let us assume that w^- is a C-segment. Let ξ^- (resp., ξ^+) be the point on Π that lies on $\partial\Omega$ immediately before (resp., after) ξ. Since $\Pi[\xi^-, \xi]$ is a Dubins path and w^- is a C-segment, $\Pi[\xi^-, \xi]$ consists of two or three C-segments. In either case,

$$d(\xi, x) \leq 6 \qquad \forall\, x \in \Pi[\xi^-, \xi]. \tag{3.1}$$

Since e lies on the x-axis and the endpoints of e are at distance ≥ 15 from ξ, the above inequality implies that $\Pi[\xi^-, \xi]$ lies below the x-axis. If ξ^- is a vertex or if ξ^- lies in the interior of an edge e^- and the line supporting e^- intersects $\Pi[\xi^-, \xi]$, then Theorem 3.1 follows from an application of Lemma 3.4 on the path $\Pi[\xi^-, \xi]$. In the remainder of this section, we will thus assume the following.

($\star$) ξ^- lies in the interior of an edge e^-, and the line, ℓ^-, supporting e^- does not intersect $\Pi[\xi^-, \xi]$.

Let σ be the intersection point of ℓ^- and the x-axis. Without loss of generality, assume that σ lies to the left of ξ. If ℓ^- is parallel to the x-axis, then σ lies at $x = -\infty$ and the angle $\angle\xi^-\sigma\xi = 0$.

We consider the following three cases and prove the existence of a region R satisfying P1–P3 for each case separately.

Case 1. The angle $\angle\xi^-\sigma\xi \geq \pi/6$; see Figure 3.3.

Case 2. The angle $\angle\xi^-\sigma\xi < \pi/6$.

Case 2.1. $d(\xi, x) < 8$ for all $x \in \Pi[\xi, \xi^+]$; see Figure 3.4.

Case 2.2. $d(\xi, x) \geq 8$ for some $x \in \Pi[\xi, \xi^+]$; see Figure 3.7.

LEMMA 3.6. *There exists a region R satisfying* P1–P3 *for Case* 1*, i.e., when* $\angle\xi^-\sigma\xi \geq \pi/6$.

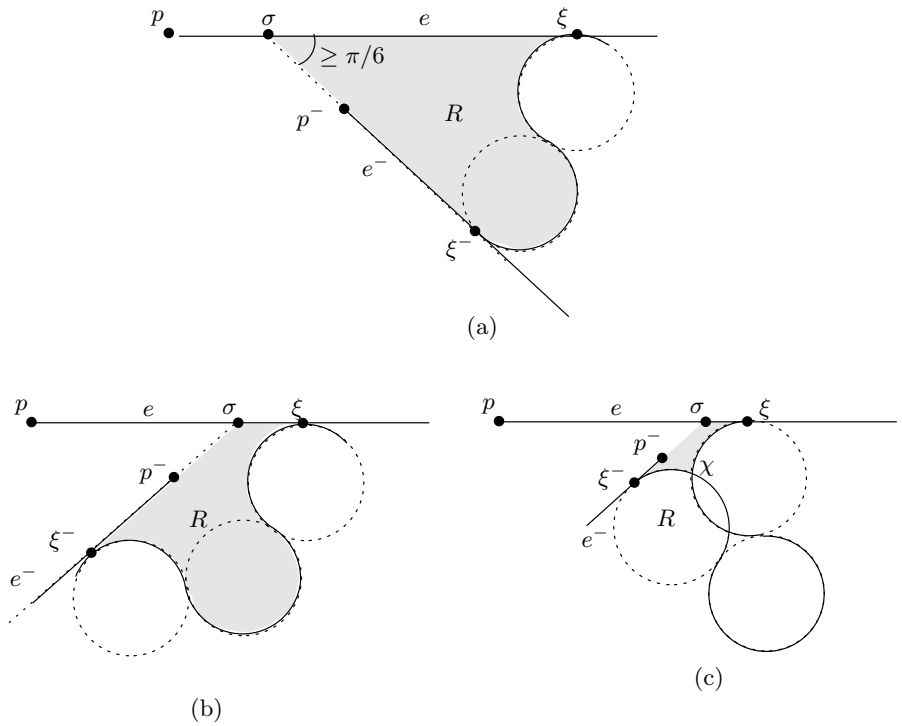


FIG. 3.3. $\angle\xi\sigma\xi^- \geq \pi/6$: (a) $\Pi[\xi^-, \xi]$ *has two C-segments;* (b), (c) $\Pi[\xi^-, \xi]$ *has three C-segments.*

Proof. If $\Pi[\xi^-, \xi]$ is simple (i.e., does not have any self-intersections), we define R to be the closed region bounded by $\Pi[\xi^-, \xi]$ and the segments $\sigma\xi$ and $\sigma\xi^-$. See Figure 3.3(a,b). Otherwise, let χ be the first self-intersection point of $\Pi[\xi^-, \xi]$, and let Γ be the simple curve obtained by discarding from $\Pi[\xi^-, \xi]$ the cycle formed by χ. We define R to be the closed region bounded by Γ and the segments $\sigma\xi$ and $\sigma\xi^-$; see the shaded region in Figure 3.3(c). Since the x-axis cannot intersect the interior of $\Pi[\xi^-, \xi]$ and, by assumption ($\star$), ℓ^- does not intersect the interior of $\Pi[\xi^-, \xi]$, R is a simply connected region. For $\angle\xi\sigma\xi^- \leq \pi/2$, using the sine law and the fact that

$d(\xi, \xi^-) \leq 6$ (see (3.1)), we obtain

$$d(\xi, \sigma) = d(\xi, \xi^-) \frac{\sin \angle \sigma\xi^-\xi}{\sin \angle \xi\sigma\xi^-} \leq \frac{d(\xi, \xi^-)}{\sin(\pi/6)} \leq \frac{6}{\sin(\pi/6)} \leq 12. \tag{3.2}$$

For $\angle \xi\sigma\xi^- > \pi/2$, $d(\xi, \sigma) < d(\xi, \xi^-) \leq 6$, since the segment $\xi\xi^-$ is the longest edge in the triangle $\Delta\xi\sigma\xi^-$. Since $d(p, \xi) > 15$, σ lies on e in both cases. The obstacle edges are disjoint, so one of the endpoints of e^-, say, p^-, lies on the segment $\sigma\xi^-$, and thus on the boundary of R, implying that R satisfies P1. R also satisfies P2 because any obstacle edge $g \in \Omega$ can cross ∂R only at the segment σp^-. Finally, (3.1) and (3.2) imply that, for any $x \in R$, $d(\xi, x) \leq \max\{d(\xi, \sigma), 6\} \leq 12$. Therefore, R satisfies P3 as well. □

LEMMA 3.7. *There exists a region R satisfying* P1–P3 *for Case* 2.1*, i.e., when $\angle \xi^-\sigma\xi < \pi/6$ and $d(\xi, x) < 8$ for all $x \in \xi^+$.*

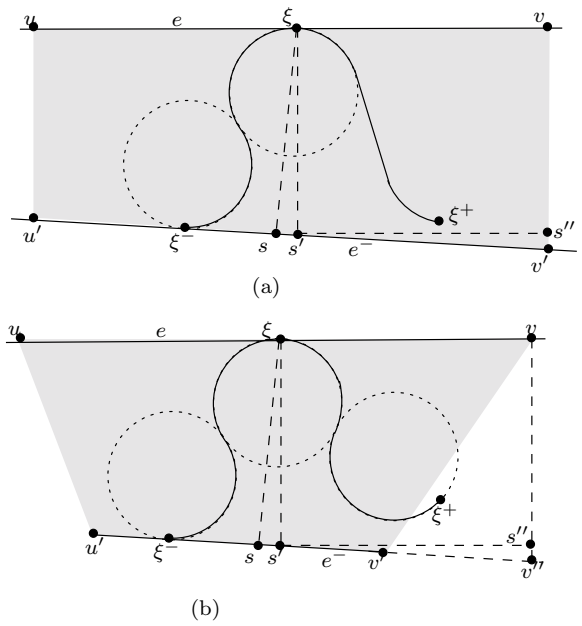


FIG. 3.4. *$\angle \xi\sigma\xi^+ < \pi/6$ and $d(\xi, \xi^+) < 8$: (a) ξ^+ lies in the interior of R; (b) v'' lies on ∂R.*

Proof. If ξ^+ is an obstacle vertex, then the claim follows from Lemma 3.4, so assume that ξ^+ lies in the interior of an obstacle edge e^+. Since $d(\xi, \xi^+) \leq 8$, ξ^+ lies in a disc D_ξ of radius 8 centered at ξ. Let u (resp., v) be the point on e at distance 8 from ξ to its left (resp., right). If the left endpoint of e^-, the edge containing the point ξ^-, lies to the left of u, let u' be the point on e^- with the same x-coordinate as u (see Figure 3.4(a)); otherwise, let u' be the left endpoint of e^- (see Figure 3.4(b)). Similarly, if the right endpoint of e^- lies to the right of v, let v' be the point on e^- with the same x-coordinate as v; otherwise, let v' be the right endpoint of e^-. We set R to be the quadrilateral $uu'v'v$.

If u' or v' is an endpoint of e^- or if any point of obstacle boundary (including ξ^+) lies in the interior of R, then R satisfies P1. Suppose neither of these conditions holds. Since uu' and vv' are vertical segments, $d(\xi, x) > 8$ for any point x on the segments uu' and vv'. Consequently, $\Pi[\xi, \xi^+]$ cannot intersect either of these two segments. We

will show in Lemma 3.8 that if the vertices of R lie in the interior of e and e^- and no obstacle intersects the interior of R, then ξ^+ cannot lie on e^- or e^+. Hence, R satisfies P1.

An obstacle edge g cannot intersect uv or $u'v'$ (as they are portions of obstacle edges), and g cannot intersect both uu' and vv' (as this would imply that g intersects $\Pi[\xi^-, \xi]$). Hence R satisfies P2.

Finally, we prove that R satisfies P3. Let v'' be the point on ℓ^- that has the same x-coordinate as v ($v'' = v'$ if the right endpoint of e^- lies to the right of v). It can be shown that, for any $x \in R$, $d(\xi, x) \leq d(\xi, v'')$. (Here we are using the assumption that ℓ and ℓ^- intersect to the left of ξ.) Let s (resp., s') be the point on ℓ^-, such that $\xi s \perp e^-$ (resp., $\xi s' \perp e$). Let s'' be the point on the segment vv'' with the same y-coordinate as that of s'. Notice that $\angle s\xi s' = \angle v''s's'' = \angle \xi\sigma\xi^- < \pi/6$, where σ is the intersection point of lines containing e and e^-. Since

$$d(v, s'') = d(\xi, s') = \frac{d(\xi, s)}{\cos \angle s\xi s'} < \frac{d(\xi, \xi^-)}{\cos \frac{\pi}{6}} \leq 6 \cdot \frac{2}{\sqrt{3}} = 4\sqrt{3}$$

and

$$d(s'', v'') = d(s', s'') \tan \angle v''s's'' < d(\xi, v) \tan \frac{\pi}{6} = 8 \cdot \frac{1}{\sqrt{3}} = \frac{8}{\sqrt{3}},$$

we have $d(v, v'') < 20/\sqrt{3}$. Therefore, for any $x \in R$,

$$d(\xi, x) \leq d(\xi, v'') = \sqrt{d(\xi, v)^2 + d(v, v'')^2} \leq \sqrt{8^2 + (20/\sqrt{3})^2} \leq 15.$$

This completes the proof of the lemma. □

We now prove the missing claim in the proof of the above lemma.

LEMMA 3.8. *Let R be the quadrilateral defined in the proof of Lemma 3.7. Suppose the vertices of R lie in the relative interiors of the edges e and e^- and the interior of R does not intersect any obstacle. If ξ^+ lies on e^- or e, then Π is not an optimal path.*

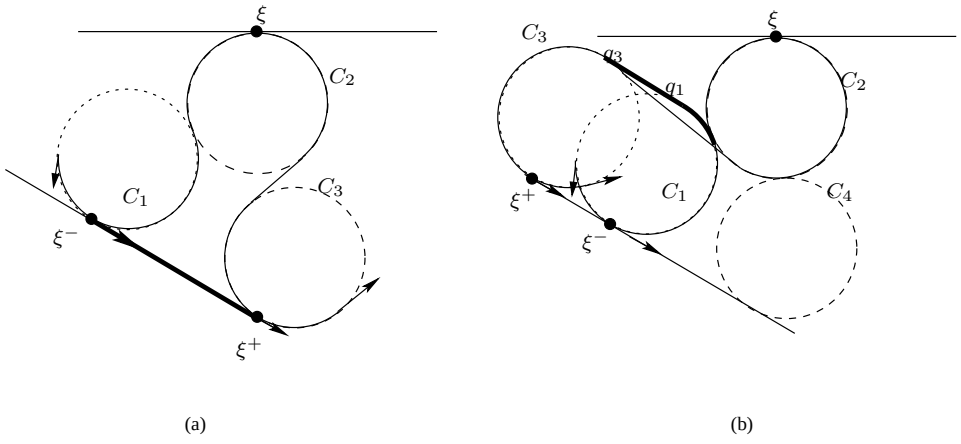


FIG. 3.5. *Shortcutting $\Pi[\xi^-, \xi^+]$ when w' is counter-clockwise oriented and ξ^+ lies to the left of ξ^-.*

Proof. We assume that ξ^+ lies on e^- and $\Pi[\xi^-, \xi]$ consists of two C-segments. We will show that we can shortcut $\Pi[\xi^-, \xi^+]$ and obtain a shorter feasible path from

I to F. A similar shortcutting procedure works if $\xi^+ \in e$ or $\Pi[\xi^-, \xi]$ consists of three C-segments. Let w' be the C-segment containing ξ^+, and let C_1, C_2, C_3 be the circles containing the C-segments w^-, w, and w'.

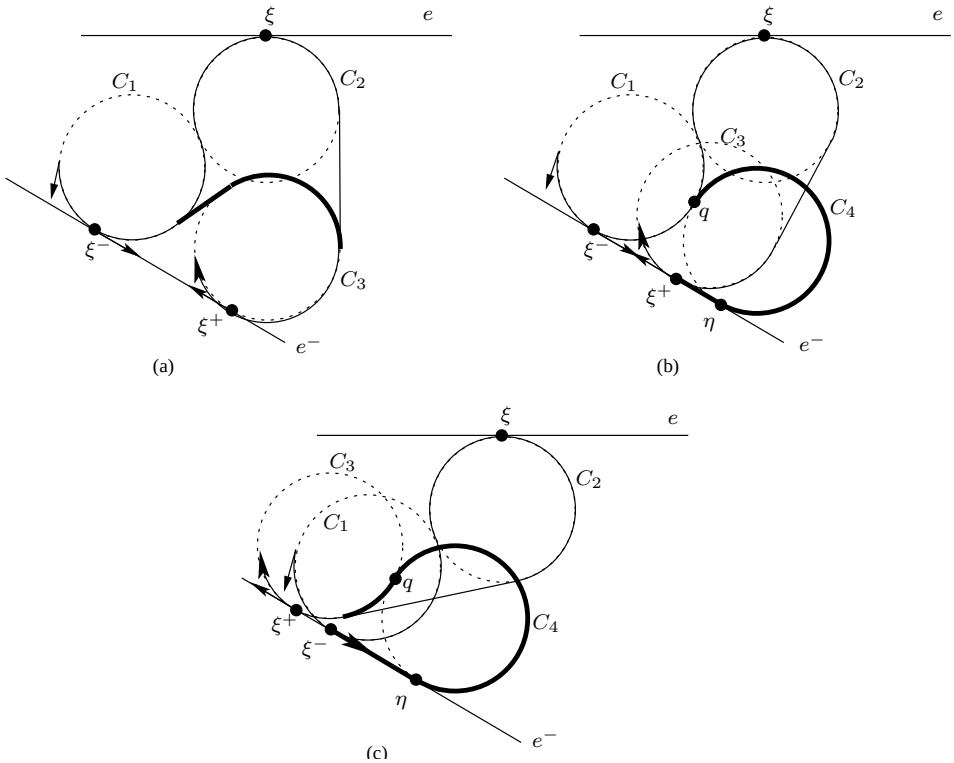


FIG. 3.6. *Shortcutting* $\Pi[\xi^-, \xi^+]$ *when* w' *is clockwise oriented; thick arcs denote the shortcutting*: (a) C_1 *and* C_3 *do not intersect,* (b) ξ^+ *lies to the right of* ξ^-*, and* (c) ξ^+ *lies to the left of* ξ^-.

Since the interior of R does not intersect any obstacle, we can prove that the interiors of circles C_1, C_2, and C_3 do not intersect any obstacle. By our assumptions, C_1 is counter-clockwise oriented and C_2 is clockwise oriented. First let us assume that w', and thus C_3, is counter-clockwise oriented. If $\xi^+ \in e^-$ lies to the right of ξ^-, we can shortcut $\Pi[\xi^-, \xi^+]$ by the segment $\overrightarrow{\xi^-\xi^+}$ (see Figure 3.5(a)), so assume that ξ^+ lies to the left of ξ^- (see Figure 3.5(b)). Since $\Pi[\xi, \xi^+]$ is a Dubins path and the C-segment w is clockwise oriented, it is a C^-LC^+-type path. This implies that the L-segment w^+ is an inner tangent of C_2 and C_3, and that C_2 and C_3 are disjoint. Let ξ' be the common endpoint of w^- and w. Using the fact that the common endpoint of w and w^+ lies on $C_2[\xi, \xi']$, the other circle (other than C_1; e.g., C_4 in Figure 3.5) tangent to both C_2 and the edge e^- lies to the right of C_1. Let $q_1 \neq \xi^-$ and $q_3 \neq \xi^+$ be the point on C_1 and C_3, respectively, so that the segment q_1q_3 is an outer common tangent of C_1 and C_3. We shortcut $\Pi[\xi^-, \xi^+]$ by the path $\Lambda = C_1[\xi^-, q_1] \parallel \overrightarrow{q_1q_3} \parallel C_3[q_3, \xi^+]$. It can be proved that $|\Lambda| \leq |\Pi[\xi^-, \xi^+]|$.

Next, let us assume that the C-segment w' is clockwise oriented. In this case we shortcut $\Pi[\xi^-, \xi^+]$ by a C^+LC^-, a C^+C^-L, or an LC^+C^- path, depending on the position of ξ^+ on e^-. If C_1 and C_3 do not intersect, we shortcut from C_1 to C_3 by

the inner tangent of C_1 and C_3 as shown in Figure 3.6(a). The new path is obviously shorter.

Suppose C_1 and C_3 intersect and ξ^+ lies to the right of ξ^-. Let C_4 be the clockwise oriented circle tangent to e^- and C_1, and that lies to the right of C_1; see Figure 3.6(b). Let $q = C_1 \cap C_4$ and $\eta = C_4 \cap e^-$. We shortcut $\Pi[\xi^-, \xi^+]$ by the C^+C^-L-type path $\Lambda = C_1[\xi^-, q] \parallel C_4[q, \eta] \parallel \overrightarrow{\eta\xi^+}$. On the other hand, if ξ^+ lies to the left of ξ^-, then we set C_4 to be the counter-clockwise oriented circle tangent to C_3 and e^-, and that lies to the right of C_1; see Figure 3.6(c). Let $q = C_3 \cap C_4$ and $\eta = C_4 \cap e^-$. We shortcut $\Pi[\xi^-, \xi^+]$ by the LC^+C^--type path $\Lambda = \overrightarrow{\xi^-\eta} \parallel C_4[\eta, q] \parallel C_3[q, \xi^+]$. We can show that in both cases, Λ is feasible and $|\Lambda| \le |\Pi[\xi^-, \xi^+]|$. □

LEMMA 3.9. *There exists a region R satisfying* P1–P3 *for Case* 2.2*, i.e., when* $\angle\xi\sigma\xi^- < \pi/6$ *and* $d(\xi, x) \ge 8$ *for some* $x \in \Pi[\xi, \xi^+]$.

Proof. Let $\eta \in \Pi[\xi, \xi^+]$ be a point for which $d(\xi, \eta) > 8$. If w^+, the segment of Π following w, is a C-segment, then, by Theorem 2.1, $\Pi[\xi, \xi^+]$ consists of at most three C-segments and $d(\xi, x) \le 6$ for all $x \in \Pi[\xi, \xi^+]$, which contradicts the assumption that $d(\xi, \eta) > 8$. Hence w^+ is an L-segment, $\Pi[\xi, \xi^+]$ is of CLC type, and $\|w^+\| \ge d(\xi, \eta) - 4 > 4$.

Let C_1 and C_2 be the circles containing w^- and w, respectively, and let h be the ray supporting w^+ and starting at the common point of w^+ and w. There are two cases to consider:

(i) h does not intersect C_1,
(ii) h intersects C_1.

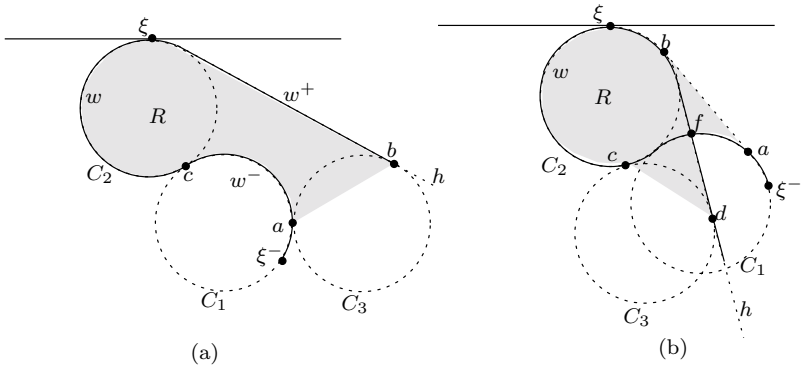


FIG. 3.7. $\angle\xi\sigma\xi^- < \pi/6$ *and* $d(\xi, \xi^+) \ge 8$: (a) h *does not intersect* C_1; (b) h *intersects* C_1.

Case (i). See Figure 3.7(a). Let C_3 be the other (unit-radius) circle tangent to both h and C_1. Let a (resp., b) be the intersection point of C_3 with C_1 (resp., h). Since $|w^+| > 4$, b lies on the segment w^+.

Let c be the common point of C_1 and C_2. Since w^- is clockwise oriented, we assume C_1 to be clockwise oriented. We consider the two cases: (a) ξ^- does not lie on $C_1[a, c]$, and (b) $\xi \in C_1[a, ac$. If $\xi^- \notin C_1[a, c]$, i.e., $a \in \Pi[\xi^-, \xi]$, define R to be the closed region bounded by $\Pi[a, b]$ and the segment ab; see the shaded region in Figure 3.7(a). If the arc $C_3[a, b]$ does not cross any obstacle edge, we can shorten Π by replacing $\Pi[a, b]$ with $C_3[a, b]$, contradicting the optimality of Π. Hence, an edge of Ω intersects $C_3[a, b]$ and thus also intersects the interior of R, thereby implying that R satisfies P1. Since an edge of Ω can cross ∂R only at ab, P2 is obvious. Finally, every point on ∂R is within distance 15 from ξ, so P3 also follows.

If $\xi^- \in C_1[a,c]$, define R to be the closed region bounded by $\Pi[\xi^-, b]$ and the segment $\xi^- b$. Since ξ^- is an interior point of the obstacle edge e^-, e^- also intersects the interior of R, proving P1. Since an edge of Ω can cross ∂R only at segment $\xi^- b$, P2 holds. P3 can also be shown.

Case (ii). See Figure 3.7(b). Let f be the first intersection point of ray h with C_1. Let ab be the segment tangent to C_1 and C_2 at a and b, respectively, such that b is on the path $\Pi[\xi, \xi^+]$. (It is easy to see that such a b exists.) Let C_3 be the unit-radius (clockwise oriented) circle tangent to C_2 and the ray h at c and d, respectively. Since $|w^+| > 4$, both d and f lie on the segment w^+.

If $a \notin \Pi[\xi^-, \xi]$, then the line ℓ^- intersects $\Pi[\xi^-, \xi]$, which contradicts the assumption ($\star$). So assume that $a \in \Pi[\xi^-, \xi]$. Define R to be the region bounded by the segment ab, the circular arc $C_2[c,b]$ the segments cd and df, and the circular arc $C_1[a,f]$; see the shaded region in Figure 3.7(b). An obstacle edge crosses either the segment ab or the arc $C_3[c,d]$, thus intersecting the interior of R, because otherwise the path obtained by concatenating the segment ab and the circular arcs $C_2[c,b]$ and $C_3[c,d]$ is shorter than the path $\Pi[a,d]$, contradicting the optimality of Π. This means that R satisfies P1. An obstacle edge can cross ∂R only at segments cd and ab, and none of the edges can cross both the segments (because then it would cross Π). P2 follows. Finally, every point on ∂R is within distance 15 from ξ, so P3 also follows. ∎

These lemmas together complete the proof of Theorem 3.1.

3.2. Anchored C-segments. Recall that a C-segment is called anchored if it touches $\partial\Omega$ at two (or more) points. We prove a property of anchored C-segments, which will be useful later. We call an anchored C-segment, touching $\partial\Omega$ at two points $p_1 \in e_1$ and $p_2 \in e_2$, *shallow* if we can find two obstacle vertices v_1 and v_2 (not necessarily distinct) such that v_i is visible from p_i and $d(p_i, v_i) \le 15$ for $i = 1, 2$. Otherwise, it is called *deep*.

LEMMA 3.10. *If w is a feasible deep anchored C-segment, then no obstacle edge intersects the interior of the circle containing w.*

Proof. Let w be an anchored C-segment touching $\partial\Omega$ at two points $p_1 \in e_1$ and $p_2 \in e_2$. Since w is a deep anchored C-segment, one of p_i, say p_1, has to be at least distance 15 away from the endpoints of e_i, because otherwise w is shallow. Without loss of generality, let $e = e_1$, $e^- = e_2$, $\xi = p_1$, $\xi^- = p_2$, where e, e^-, ξ, ξ^- are as defined in the last section.

It can be shown that the angle between e and e^- is $< \pi/6$. Otherwise, following the proof of Lemma 3.6, we can find a vertex v_i, such that v_i is visible to p_i and $d(v_i, p_i) \le 15$, contradicting the assumption that w is a deep anchored C-segment.

Let C be the circle containing w. We construct a quadrilateral $R = uu'v'v$, as in the proof of Lemma 3.7, except that we set $d(\xi, u) = d(\xi, v) = 2$, since the distance between ξ and any point on C is ≤ 2. Any obstacle edge crossing both uu' and vv' of R also crosses the C-segment w, but that is impossible because w is feasible. Hence, R satisfies P2. Obviously, R satisfies P3. If an obstacle edge intersects the interior of C, it also intersects R, making R satisfy P1. If so, by Lemma 3.3, we can find for each $i = 1, 2$ an obstacle vertex v_i that is visible from p_i and $d(v_i, p_i) \le 15$, contradicting that w is a deep anchored C-segment. Thus no obstacle edge intersects the interior of C. This proves the lemma. ∎

For a pair of nonparallel edges, there is at most one deep anchored circle, but a pair of parallel edges at distance 2 have infinitely many deeply anchored C-segments. In the next lemma, we prove that we do not have to consider parallel edges for deeply

anchored C-segments.

LEMMA 3.11. *An optimal path does not contain a deep C-segment that touches two parallel obstacle edges at their interior points.*

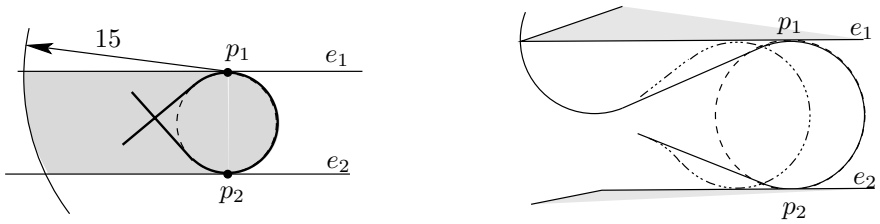


FIG. 3.8. (a) *Shaded region is R;* (b) *perturbation scheme for a deeply anchored C-segment.*

Proof. Suppose an optimal path Π contains a deep C-segment w anchored at two parallel edges e_1 and e_2. Let p_i be the intersection point of w and e_i. Since w is deep, no obstacle vertex within distance 15 is visible from one of p_i's, say, p_1. Let R be the region lying inside the disk of radius 15 centered at p_1 and the strip formed by the lines supporting e_1, e_2 and lying to the left of $w[p_1, p_2]$ (shaded region in Figure 3.8(a)). Following the same argument as in the previous lemma, we can argue that R does not intersect the obstacle boundary. If the segment preceding or following w is a C-segment, then there is an obstacle within distance 8 from ξ that lies inside the region R, and therefore both the segments are L-segments. Using a perturbation shown in Figure 3.8(b), we can now prove that Π is not optimal. This completes the proof of the lemma. □

LEMMA 3.12. *There are only $O(n)$ circles that can contain a feasible, deep anchored C-segment.*

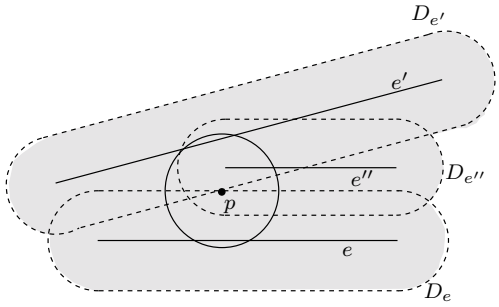


FIG. 3.9. *The straight line edges of D_e and $D_{e'}$ intersect at p. The unit radius circle centered at p is tangent to both e and e', and is crossed by e'', whose race-track $D_{e''}$ contains p in its interior.*

Proof. For each edge $e \in \Omega$, let D_e be the Minkowski sum of e and the unit-radius disk centered at the origin.[2] D_e is a race-track bounded by two semicircles of unit radius and two translated copies of e; see Figure 3.9. An intersection point p of the straight-line edges of D_e and $D_{e'}$ corresponds to the center of a unit-radius circle tangent to e and e' at their interior points. (Notice that there may be an infinite number of intersection points if e and e' are parallel and unit distance apart.) A point p lies inside D_e if and only if e intersects the interior of the unit-radius circle centered

[2]The *Minkowski sum* of two sets A and B is defined as $A \oplus B = \{a + b \mid a \in A, b \in B\}$.

at p. Set $\mathcal{D} = \bigcup_{e\in\Omega} D_e$. A unit-radius circle is tangent to two obstacle edges and does not intersect any obstacle edge in its interior only if its center lies on the intersection point σ of the boundaries of two racetracks and does not lie in the interior of any other racetrack. If the two boundary edges forming the intersection point overlap, i.e., the two corresponding obstacle edges are parallel, then by Lemma 3.11 such a circle cannot contain a deep anchored C-segment, so the intersection point σ has to be a vertex of $\mathcal{D}$. Thus the number of circles that can contain a feasible, deep anchored C-segment is at most the number of vertices of $\partial\mathcal{D}$, which, by a result of Kedem et al. [29], is $O(n)$; so the lemma follows. □

4. Computing near optimal paths. In this section we present an efficient algorithm for computing a feasible path whose length is no more than $(1+\varepsilon)$ times the length of an optimal path for any $\varepsilon > 0$, provided that the optimal path is ε-robust. As in [27], we construct a weighted directed graph $G = (V, E)$, where V is a set of discretized positions. These positions are obtained by discretizing the set of orientations at which a path touches a vertex and the set of points at which a path touches an edge. Jacobs and Canny [27] choose points uniformly spaced along each edge. We, on the other hand, use Theorem 3.1 and Lemma 3.12 to choose points more carefully, as described in section 4.1. There is an edge $(X, Y) \in E$ from a position X to another position Y if there exists a feasible Dubins path from X to Y. If there is more than one such path, we choose the one with the minimum arc length. The weight of an edge is the arc-length of the chosen Dubins path. We claim that by choosing the proper parameter δ for discretization for any optimal path Π from X to Y that is robust, there is a path from X to Y along the edges in G whose length is at most $(1+\varepsilon)$ times the length of Π. The choice of δ and the proof of this claim are given in section 5. Therefore, the problem reduces to computing a shortest path in G, which can be done in time $O(|V|^2)$, using Dijkstra's shortest-path algorithm.

4.1. Computing the node set. In this subsection we describe the node set V and an efficient algorithm for computing it. We set $V = \{I, F\} \cup V_1 \cup V_2 \cup V_3$, where each subset V_i corresponds to a specific type of positions. The first set V_1 corresponds to positions located at the vertices of Ω. More precisely, for each vertex v of Ω with feasible domain $[\alpha_v, \beta_v]$, we add the positions $(v, \alpha_v + i\delta)$ for $0 \le i < \lceil(\beta_v - \alpha_v)/\delta\rceil$ and (v, β_v) to V_1. V_1 can be constructed in $O(n/\delta)$ time in a straightforward manner.

The second set V_2 corresponds to deeply anchored C-segments. For a pair of nonparallel edges e_1, e_2, if the unit-radius circle C tangent to e_1 and e_2 does not intersect any other edge of Ω, we add four positions $(p_1, \theta_1), (p_1, \theta_1 + \pi), (p_2, \theta_2)$, and $(p_2, \theta_2 + \pi)$ to V_2, where p_i is the point at which C is tangent to e_i, and θ_i is the angle between e_i and the x-axis. Using an algorithm of Kedem et al. [29], the set of unit-radius circles tangent to two edges and not intersecting any other edge can be computed in $O(n \log^2 n)$ time, and so can be the set V_2.

The third set V_3 corresponds to semifree C-segments and shallowly anchored C-segments. For each edge $e \in \Omega$, we first mark a portion $\hat{e}$ of it, as described below, and then choose those points on e that are at distance $i\delta$ from its left endpoint for any integer $i \ge 0$, such that the semi-open interval $[i\delta, (i+1)\delta)$ intersects $\hat{e}$. Let S^e denote the set of points selected on e. Assuming that the angle between e and the x-axis is θ, for each point $p \in S^e$, we add two positions (p, θ) and $(p, \theta + \pi)$ to V_3.

We now describe which portion of each edge is marked. We mark the edges in two stages. First, for each edge e, we mark the portion that lies within distance 30 from any of its endpoints. Next, for each vertex $v \in \Omega$, let D_v be the disk of radius 15 centered at v. Let E_v be the set of edges that contain at least one unmarked point

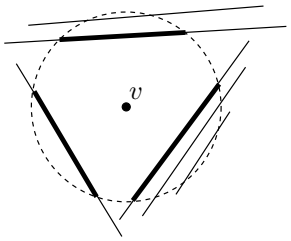


FIG. 4.1. *Segments in E_v; fat edges denote the portion of E_v E_v-visible from v.*

inside D_v, i.e., $e \in E_v$ if $e \cap D_v$ contains at least one point whose distance to both endpoints of e is more than 30. A point is E_v-visible from v if it is visible from v with respect to the edge set E_v (i.e., we ignore the edges in the set $E \setminus E_v$). For each edge $e \in E_v$, we mark the portion of $e \cap D_v$ that is E_v-visible from v. We repeat this step for all vertices of Ω.

LEMMA 4.1. *Let p be a point on an edge $e \in \Omega$ so that there is a vertex v of Ω visible from p and so that $d(p, v) \leq 15$. Then there is a point $q \in S^e$ within distance δ from p.*

Proof. It suffices to show that every such point p lies on $\hat{e}$ because, by construction, for every point $q \in \hat{e}$, there is a point $q' \in S^e$ such that $d(q, q') \leq \delta$. Since $d(p, v) \leq 15$ and v is visible from p, the above algorithm would have been marked p, implying that $p \in \hat{e}$. ∎

LEMMA 4.2. *Suppose Π is an optimal path from I to F, and let $\langle X_1, \ldots, X_k \rangle$ be the sequence of positions on Π such that $\text{LOC}(X_i) \in \partial\Omega$ for every $1 \leq i \leq k$. For each $1 \leq i \leq k$, there exist two (not necessarily distinct) nodes $Y_i^-, Y_i^+ \in V$ such that*

(i) *if $\text{LOC}(X_i)$ is a vertex v of Ω, then $\text{LOC}(Y_i^-), \text{LOC}(Y_i^+) = v$, $\text{VEC}(Y_i^-) \leq \text{VEC}(X_i) \leq \text{VEC}(Y_i^+)$, and $|\text{VEC}(Y_i^-) - \text{VEC}(X_i)|, |\text{VEC}(Y_i^+) - \text{VEC}(X_i)| \leq \delta$; or*

(ii) *if $\text{LOC}(X_i)$ is an interior point of an obstacle edge e, then $\text{LOC}(Y_i^-), \text{LOC}(Y_i^+) \in e$, Y_i^-, X, and Y_i^+ appear in that order along e,*

$$d(\text{LOC}(Y_i^-), \text{LOC}(X_i)), d(\text{LOC}(Y_i^+), \text{LOC}(X_i)) \leq \delta,$$

and $\text{VEC}(Y_i^-) = \text{VEC}(Y_i^+) = \text{VEC}(X_i)$.

Proof. If $\text{LOC}(X_i)$ is a vertex, the lemma follows from the definition of V_1. If $p = \text{LOC}(X_i)$ lies in the interior of an obstacle edge e, then p lies on a semifree C-segment, a shallow anchored C-segment, or a deep anchored C-segment. In the last case, X_i is a node in V_2. In the first two cases, there is a vertex $v \in \Omega$ so that v is visible from p and that $d(p, v) \leq 15$. By Lemma 4.1 and the definition of V_3, there exist two nodes $Y_i^-, Y_i^+ \in V_3$ so that $d(\text{LOC}(Y_i^-), \text{LOC}(X_i)), d(\text{LOC}(Y_i^+), \text{LOC}(X_i) \leq \delta$ and $\text{VEC}(Y_i^-) = \text{VEC}(Y_i^+) = \text{VEC}(X_i)$. This completes the proof of the lemma. ∎

Next, we prove that the size of V is $O(n/\delta)$. Since $|V_1| = O(n/\delta)$ and $|V_2| = O(n)$, it suffices to bound the size of V_3.

LEMMA 4.3. $\sum_e |S^e| = O(n/\delta)$.

Proof. For each edge $e \in \Omega$, let $\hat{e}$ denote the marked portions of e, $\#\hat{e}$ the number of connected components of $\hat{e}$, and $\|\hat{e}\|$ the total length of $\hat{e}$. For each connected component γ of $\hat{e}$, the algorithm chooses $2 + \|\gamma\|/\delta$ points. Therefore, $|S^e| \leq 2\#\hat{e} + \|\hat{e}\|/\delta$.

The total measure of points marked in the first stage is at most $60n$. Recall that for each vertex v, $e \in E_v$ if $e \cap D_v$ contains at least one point whose distance to both

endpoints of e is more than 30. Thus none of the endpoints of E_v lie inside D_v. This implies that the set of points in $(\bigcup E_v) \cap D_v$ that are E_v-visible from v form a set of disjoint chords of D_v (see Figure 4.1), and therefore its measure is bounded by the length of the perimeter of D_v, which is 30π. Therefore,

$$\sum_{e \in \Omega} \|\hat{e}\| \leq 60n + 30\pi n = O(n).$$

Next, for each connected component $\gamma \in \hat{e}$, if γ is the first or the last connected component of $\hat{e}$, then we charge γ to e itself. Otherwise, charge γ to any of the vertices that marked it in the second stage. We will prove in Lemma 4.4 that each vertex is charged by at most ten connected components, so

$$\sum_{e \in \Omega} \#\hat{e} = 2n + 10n = O(n).$$

This completes the proof. ∎

LEMMA 4.4. *Each obstacle vertex is charged by at most ten connected components.*

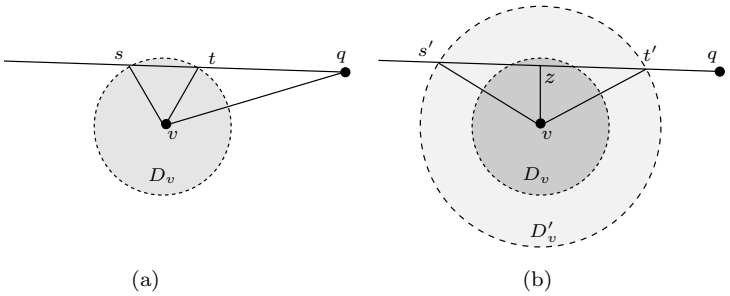


FIG. 4.2. $d(s,t) < 15$ *and* $\angle s'vt' > \pi/2$.

Proof. Since the edges of E_v are disjoint and do not contain any of its endpoints inside D_v and the visibility is taken with respect to the edge set E_v, v marks at most one connected component of each edge in E_v; see Figure 4.1. That is, for each edge $e \in E_v$, either v marks the entire chord $e \cap D_v$, or it does not mark any point of e. Hence, the number of connected components charged to v is bounded by the number of edges in E_v that are E_v-visible from v.

Partition the set of edges that are E_v-visible from v into two subsets Γ_1 and Γ_2. An edge e is in Γ_1 if the length of the chord $e \cap D_v$ is at least 15, and in Γ_2 otherwise. Each edge $e \in \Gamma_1 \cup \Gamma_2$ splits the circle ∂D_v into two circular arcs; we refer to the shorter one as the *cap* induced by e. Since the edges in $\Gamma_1 \cup \Gamma_2$ are E_v-visible from v, the caps induced by them are pairwise disjoint.

Each cap induced by an edge of Γ_1 spans an angle of at least $\pi/3$ because the length of the chord $e \cap D_v$ is at least 15 and the radius of D_v is 15. Since the caps are disjoint, $|\Gamma_1| \leq 6$.

Next, let e be an edge of Γ_2. Let p (resp., q) denote the left (resp., right) endpoint of e, and let s (resp., t) denote the left (resp., right) endpoint of $e \cap D_v$; see Figure 4.2(a). By definition, $d(s,t) < 15$, and therefore $\angle stv, \angle tsv > \pi/3$ and $d(t,q) > 15$. Since $e \in E_v$, $e \cap D_v = st$ contains a point u such that $d(u,p), d(u,q) > 30$. Hence,

$d(s,q) = d(s,u) + d(u,q) > 30$, and

$$\begin{aligned} d(v,q) &= \sqrt{d(v,s)^2 + d(s,q)^2 - 2\,d(s,q)\,d(v,s)\cos(\angle vst)} \\ &= \sqrt{d(v,s)^2 + d(s,q)^2 - 2\,d(s,q)\,d(v,s)\frac{d(s,t)}{2d(v,s)}} \\ &= \sqrt{d(v,s)^2 + d(s,q)(d(s,q) - d(s,t))} \\ &= \sqrt{d(v,s)^2 + d(s,q)d(q,t)} \\ &> \sqrt{15^2 + 30 \cdot 15} = 15\sqrt{3}. \end{aligned}$$

Similarly, one can show that $d(v,p) > 15\sqrt{3}$.

Draw a disk D'_v of radius $15\sqrt{3}$ centered at v. Since $d(v,p), d(v,q) > 15\sqrt{3}$, the endpoints of every edge in Γ_2 lie outside D'_v. Moreover, each edge of Γ_2 is E_v-visible from v, the caps of $\partial D'_v$ induced by the edges of Γ_2 are also pairwise disjoint. For an edge e, let s', t' be the endpoints of $e \cap D'_v$, and let z be the midpoint of the segment $s't'$. See Figure 4.2(b). Since $d(v,z) \le 15$ and $d(v,t') = 15\sqrt{3}$,

$$\angle s'vt' = 2\angle zvt' = 2 \cdot \cos^{-1}\frac{d(v,z)}{d(v,t')} \ge 2 \cdot \cos^{-1}\frac{1}{\sqrt{3}} > \pi/2.$$

Hence, the cap of $\partial D'_v$ induced by $e \in \Gamma_2$ spans an angle of at least $\pi/2$, which implies $|\Gamma_2| \le 4$. This completes the proof of the lemma. □

We now show that the node set V_3 can be computed in time $O(n^2 \log n + n/\delta)$. For every vertex v, the set E_v can be computed in $O(n)$ time in a straightforward manner. Let $E'_v = \{e \cap D_v \mid e \in E_v\}$. An edge e' of $E'v$ is marked if it is E_v-visible from v. Recall that either every point on e' is E_v-visible from v, or no point on e is E_v-visible from v. This set can be computed in $O(n \log n)$ time by performing an angular sweep around v. Let $\rho(\theta)$ be the ray emanating from v in direction θ. Let $\theta_1, \ldots, \theta_k$ be the orientations such that $\rho(\theta)$ passes through an endpoint of an edge in E'_v. We sweep the plane with the ray $\rho(\theta)$ by varying θ from 0 to 2π. For each θ, we maintain the edges of E'_v intersecting $\rho(\theta)$, sorted in the order they intersect $\rho(\theta)$. Since the segments of E'_v are pairwise disjoint, the ordering changes only at θ_i's. Let $e_i \in E'_v$ be the first edge in this ordering in the interval $[\theta_i, \theta_{i+1})$. We mark e_i. At each θ_i, we can update the ordering in time $O(\log n)$. Repeating this process for all the vertices of Ω, V_3 can be computed in time $O(n^2 \log n + n/\delta)$. Hence, we conclude the following lemma.

LEMMA 4.5. *$|V| = O(n/\delta)$ and V can be computed in time $O(n^2 \log n + n/\delta)$.*

4.2. Computing the edge set. We now describe how to compute the edge set E. For each pair of positions $X, Y \in V$, we compute all $O(1)$ Dubins paths from X to Y, check which of them are feasible, and select the one with the minimum arc length. The only nontrivial step is to determine whether a given Dubins path is feasible. We will consider CCC and CLC paths separately.

Testing CCC paths. CCC paths can be further classified into two subcategories: $C^+C^-C^+$ and $C^-C^+C^-$. We consider only $C^+C^-C^+$ paths; $C^-C^+C^-$ paths can be handled in a similar manner.

For each position X, let C_X denote the clockwise oriented circle passing through X, and let ϕ_X^+ (resp., ϕ_X^-) be the intersection point of C_X and $\partial\Omega$ immediately after (resp., before) $\text{LOC}(X)$, so the interiors of the arcs $C_X[\text{LOC}(X), \phi_X^+]$ and $C_X[\phi_X^-, \text{LOC}(X)]$ do not intersect $\partial\Omega$ (see Figure 4.3(a)); ϕ_X^+ and ϕ_X^- can be computed in $O(n)$ time.

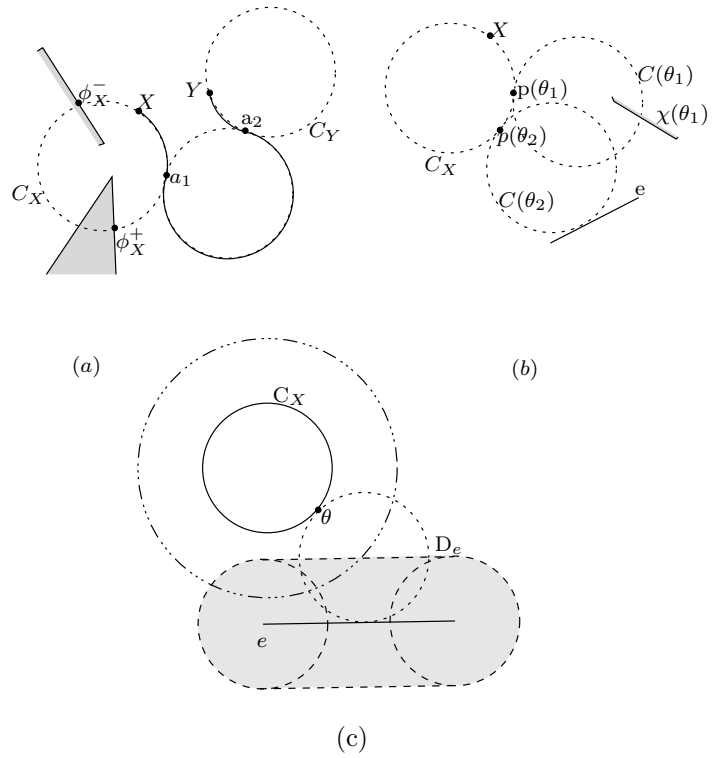


FIG. 4.3. (a) $C^+C^-C^+$ *path;* (b) $p(\theta), C(\theta)$, *and a critical orientation* θ_2; (c) *the center of* C_θ *is an intersection point of* ∂D_e *and* C'_X.

Let $w_1w_2w_3$ be a $C^+C^-C^+$ path from a position X to another position Y, with a_i being the common endpoint of w_i and w_{i+1} for $i = 1, 2$. Obviously, w_1 (resp., w_3) does not intersect Ω if and only if $a_1 \in C_X[\text{LOC}(X), \phi_X^+]$ (resp., $a_2 \in C_Y[\phi_Y^-, \text{LOC}(Y)]$); see Figure 4.3(a). After computing ϕ_X^+ and ϕ_X^- for all $O(n/\delta)$ positions in time $O(n^2/\delta)$, given a $C^+C^-C^+$ path, we can check in $O(1)$ time whether its first and last C-segments intersect Ω. Next, we describe how to test whether the middle C-segment of a $C^+C^-C^+$ path intersects Ω.

Fix a position X. We construct a linear-size data structure, in $O(n \log n)$ time, which, given a position Y, can determine in $O(\log n)$ time whether the middle C-segment of the $C^+C^-C^+$ path from X to Y intersects any obstacle edge.

For an orientation θ, $0 \le \theta < 2\pi$, let $p(\theta)$ be the point on C_X such that the length of the arc $C_X[\text{LOC}(X), p(\theta)]$ is θ. Let $C(\theta)$ be the counter-clockwise-directed circle tangent to C_X at $p(\theta)$. Set $\chi(\theta)$ to be the first edge of Ω intersected by $C(\theta)$, as one walks along it (in the counter-clockwise direction) starting from $p(\theta)$. If there is no such edge, then $\chi(\theta)$ is undefined (see Figure 4.3(b)).

We call an orientation θ *critical* (with respect to X) if $C(\theta)$ is either tangent to an obstacle edge or it passes through an obstacle vertex. Let $\langle \theta_1, \theta_2, \ldots, \theta_k \rangle$ be the sequence of critical angles sorted in the increasing order. Using a simple continuity argument and the fact that the obstacle edges have pairwise disjoint interiors, we can prove the following.

LEMMA 4.6. *For all orientations* θ *in any interval* (θ_i, θ_{i+1}), *the set of obstacle edges that intersect* $C(\theta)$ *and the order in which they intersect* $C(\theta)$ *remain the same.*

This lemma implies that the value of $\chi(\theta)$ remains the same for all orientations within each interval $[\theta_i, \theta_{i+1})$.

LEMMA 4.7. *For a fixed position X, there are $O(n)$ critical orientations, and they can be computed in $O(n)$ time.*

Proof. For an edge $e \in \Omega$, let D_e be the Minkowski sum of e and the unit-radius disk. Let C'_X be the circle concentric with C_X and of radius 2. $C(\theta)$ is tangent to e or passes through an endpoint of e if and only if $C(\theta)$ is centered at an intersection point of C'_X and ∂D_e (see Figure 4.3(c)). There are at most 4 intersection points between C'_X and ∂D_e. Thus an edge e can contribute at most $O(1)$ critical orientations, resulting in $O(n)$ critical orientations for all edges. These critical orientations can easily be computed in $O(n)$ time. □

By sweeping the circle $C(\theta)$ for $\theta \in [0, 2\pi)$, we can compute the values of χ for all $O(n)$ intervals (θ_i, θ_{i+1}) in $O(n \log n)$ time as follows. For each θ, we maintain the set of edges intersecting $C(\theta)$, sorted in the order in which they intersect $C(\theta)$. By Lemma 4.6, this ordering changes only at critical orientations. At each critical orientation, we can update the ordering in $O(\log n)$ time, thus spending a total of $O(n \log n)$ time. We record the value of χ for each interval $[\theta_i, \theta_{i+1})$.

Now, given a $C^+C^-C^+$ Dubins path $w_1w_2w_3$, we first compute the orientation θ of a_1, the common endpoint of w_1 and w_2. Using the above data structure, we can determine $e = \chi(\theta)$ in $O(\log n)$ time by a binary search. Finally, we check in $O(1)$ time whether w_2 intersects the edge e (That is, we check whether a_2, the common endpoint of w_2 and w_3, lies before the intersection point of $C(\theta)$ and e, in which case w_2 does not intersect any edge of Ω.) This completes the description of the data structure.

We can thus determine in time $O((n^2/\delta^2) \log n)$ all pairs $X, Y \in V$ for which there is a feasible CCC-path from X to Y.

Testing CLC paths. There are four types of CLC paths, namely, C^+LC^+, C^+LC^-, C^-LC^+, and C^-LC^-. Consider C^+LC^+ paths. Let $w_1w_2w_3$ be a C^+LC^+ path. After $O(n^2/\delta)$ preprocessing as above, we can easily determine whether w_1 or w_3 intersects Ω. As for w_2, we construct a similar data structure. Fix a position X. For a given θ, we now define $\ell(\theta)$ to be the ray tangent to C_X and emanating from $p(\theta)$, and we define $\chi(\theta)$ to be the first edge of Ω intersected by $\ell(\theta)$. An orientation θ is *critical* if $\ell(\theta)$ passes through a vertex of Ω. We can again construct a linear-size data structure in $O(n \log n)$ time that, given a position Y, can determine in $O(\log n)$ time whether the line segment of the C^+LC^+ path from X to Y intersects Ω.

Hence, we can compute in $O((n^2/\delta^2) \log n)$ time all pairs $X, Y \in V$ that admit a feasible C^+LC^+ path from X to Y. Repeating this procedure for other types of CLC paths, we can compute in $O((n^2/\delta^2) \log n)$ time all the pairs $X, Y \in V$ for which there is a feasible CLC path from X to Y.

After having computed the vertices and edges of G, we can compute a shortest path in G, using any standard shortest-path algorithm [19]. Putting everything together, we obtain the following result.

THEOREM 4.8. *The graph G, as described above, can be constructed in time $O((n^2/\delta^2) \log n)$, and a shortest path from I to F in G can be computed in an additional $O(n^2/\delta^2)$ time.*

5. Error analysis. In this section we prove that our algorithm computes an $(1 + \varepsilon)$-approximation to an optimal path provided that we choose $\delta = c\varepsilon^2$, where c is a sufficiently small constant independent of ε, and the optimal path is δ-robust. We first bound the change in the length of a Dubins path, whose end-positions lie on

obstacle boundaries, as we perturb the parameters of its end-positions, and then we bound the length of the path computed by the above algorithm.

5.1. Error induced by a single Dubins path. To give an error bound for our approximation algorithm, we need to answer the following question: given two Dubins paths of the same type whose end-positions differ by a small amount, what is the difference in length of these two paths? Let X and X' be two positions on an obstacle feature (vertex or edge) φ. We define $\Delta(X', X)$ to be the distance between X and X' in the configuration space. If φ is a vertex, then $\textsc{loc}(X') = \textsc{loc}(X)$ and we define $\Delta(X', X) = |\textsc{vec}(X') - \textsc{vec}(X)|$. If φ is an edge, i.e., $\textsc{vec}(X') = \textsc{vec}(X)$, then we define $\Delta(X', X) = \|\textsc{loc}(X') - \textsc{loc}(X)\|$. For two paths Π and Π', let $\Delta(\Pi', \Pi)$ be the difference in length of these two paths, i.e., $\Delta(\Pi', \Pi) = |\|\Pi'\| - \|\Pi\||$.

LEMMA 5.1. *Let φ_1 and φ_2 be two obstacle features, and let Π be a CLC path from a position I on φ_1 to a position F on φ_2. Let I' and F' be positions on φ_1 and φ_2, respectively, such that $\Delta(I', I) = \delta_I$ and $\Delta(F', F) = \delta_F$ for any reals $0 \le \delta_I, \delta_F \le 1$. Let Π' be the path from I' to F' of the same type as Π. If Π and Π' are neighbors, then*

$$\Delta(\Pi', \Pi) = O(\delta_I + \delta_F).$$

Proof. We will prove that if $\delta_F = 0$ (i.e., $F' = F$), then $\Delta(\Pi', \Pi) = O(\delta_I)$. By reversing the direction of Π, this also implies that $\Delta(\Pi', \Pi) = O(\delta_F)$ if $\delta_I = 0$. If both $\delta_I, \delta_F > 0$, then let Π'' be the CLC path from I' to F of the same type as Π; the existence of Π'' follows from the fact that Π and Π' are neighbors. Then

$$\Delta(\Pi', \Pi) \le \Delta(\Pi'', \Pi) + \Delta(\Pi', \Pi'') = O(\delta_I + \delta_F),$$

as claimed. We now assume $F' = F$, and set $\delta = \delta_I$.

We will first prove the claim for the case in which φ_1 is an obstacle vertex u, i.e., $I = (u, \theta_I)$ and $|\textsc{vec}(I') - \textsc{vec}(I)| = \delta$. Let C_1 (resp., C_2) be the unit-radius circle containing the initial (resp., final) C-segment of Π, and let o_i (for $i = 1, 2$) denote the centers of C_i. Let C_1' be the circle containing the initial C-segment of Π', and let o_1' be the center of C_1'. If Π is a C^+LC^+ or C^-LC^- type path, then

$$\|\Pi\| = |\textsc{vec}(I) - \textsc{vec}(F)| + d(o_1, o_2),$$

in which case,

$$\Delta(\Pi', \Pi) \le |\textsc{vec}(I') - \textsc{vec}(I)| + |d(o_1, o_2) - d(o_1', o_2)| \le \delta + d(o_1, o_1').$$

By applying the cosine law to $\Delta o_1 u o_1'$ (see Figure 5.1(b)),

$$d(o_1, o_1') = \sqrt{1 + 1 - 2\cos\delta} = 2\sin(\delta/2) \le \delta. \tag{5.1}$$

Therefore, $\Delta(\Pi', \Pi) \le 2\delta$.

In the following we prove the lemma for the case when Π is a C^+LC^- type path; the other case, when Π is a C^-LC^+ type path, is symmetric. Let p (resp., q) be the initial (resp., final) point of the L-segment of Π, i.e., it is the point at which the common tangent of C_1^+ and C_2^- touches C_1 (resp., C_2). Let s be the point on C_1 such that $so_1 \perp o_1o_2$ and $\angle so_1p < \pi/2$, and let S be the position on C_1 corresponding to s, i.e., $\textsc{loc}(S) = s$ and $\textsc{vec}(S)$ is the orientation of the tangent to C_1 (assuming that C_1 is clockwise oriented) at s. We define a similar point t and position T on

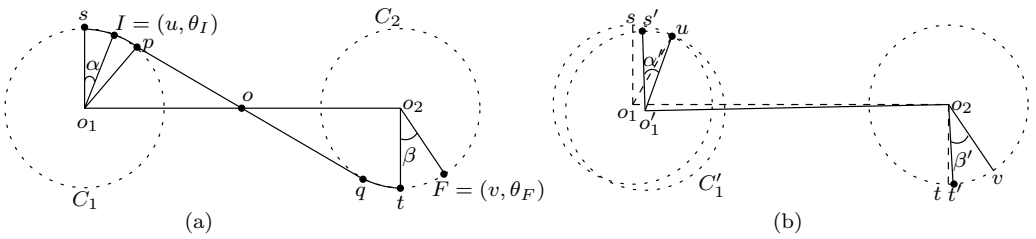


FIG. 5.1. *Bounding the difference in path length for CLC paths.*

C_2; see Figure 5.1(a). Instead of considering Π, we consider the C^+LC^- path from S to T and let ρ be the length of this path. Let $\alpha = \angle so_1u \in [-\angle po_1s, 2\pi - \angle po_1s)$ and $\beta = \angle to_2v \in [-\angle qo_2t, 2\pi - \angle qo_2t)$, measured in the counter-clockwise direction. Then

$$\|\Pi\| = \rho + \alpha + \beta;$$

see Figure 5.1(a). We define ρ', α', and β' corresponding to Π'. Then $\|\Pi'\| = \rho' + \alpha' + \beta'$, and

$$\Delta(\Pi', \Pi) \leq |\rho' - \rho| + |\alpha' - \alpha| + |\beta' - \beta|. \tag{5.2}$$

We first bound $|\alpha' - \alpha|$. Applying the sine law to $\triangle o_1o_2o'_1$,

$$\frac{\sin \angle o_1o_2o'_1}{d(o_1, o'_1)} = \frac{\sin \angle o'_1o_1o_2}{d(o_1, o_2)}.$$

Therefore $\sin \angle o_1o_2o'_1 \leq d(o_1, o'_1)/d(o_1, o_2)$. Using (5.1), the fact that $d(o_1, o_2) \geq 2$, and the inequality $\sin^{-1} x \leq 2x$ for any $0 \leq x \leq 1$, we obtain

$$\angle o_1o_2o'_1 \leq \sin^{-1} \frac{\delta}{2} \leq \delta.$$

Let w be the intersection point of lines supporting the segments so_1 and $s'o'_1$. Since $so_1 \perp o_1o_2$ and $s'o'_1 \perp o'_1o_2$, $\angle o_1wo'_1 = \angle o_1o_2o'_1 \leq \delta$; see Figure 5.2. Moreover,

$$\alpha' + \angle o_1uo'_1 = \alpha + \angle o_1wo'_1,$$

and therefore,

$$|\alpha' - \alpha| \leq \angle o_1uo'_1 + \angle o_1wo'_1 \leq 2\delta. \tag{5.3}$$

Similarly, we can show that $|\beta' - \beta| = 2\delta$.

Next, we bound $|\rho' - \rho|$. Notice that

$$\rho = 2\left(d(o, p) + \|C_1[s, p]\|\right).$$

Let $\mu = d(o_1, o) = d(o_1, o_2)/2$; then $\angle so_1p = \angle poo_1 = \sin^{-1}(1/\mu)$, and we have

$$\rho = 2\left(\sin^{-1}\left(\frac{1}{\mu}\right) + \sqrt{\mu^2 - 1}\right).$$

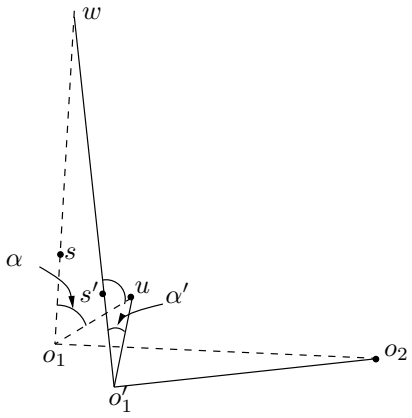


FIG. 5.2. *Bounding* $|\alpha' - \alpha|$.

Similarly, if we let $\nu = d(o_1', o_2)/2$, then

$$\rho' = 2\left(\sin^{-1}\left(\frac{1}{\nu}\right) + \sqrt{\nu^2 - 1}\right).$$

As in (5.1), $d(o_1', o_1) \leq \delta$, which implies that $|\nu - \mu| \leq \delta/2 \leq \delta$. Consider the function

$$f(x) = \sin^{-1}\left(\frac{1}{x}\right) + \sqrt{x^2 - 1}.$$

Then $|\rho' - \rho| = 2|f(\nu) - f(\mu)|$. Since both $f(x)$ and its derivative are monotonically increasing,

$$|\rho' - \rho| \leq 2(f(\mu + \delta) - f(\mu))$$

and $|\rho' - \rho|$ maximizes as μ tends to infinity. Using Taylor expansion, one can write

$$f(x) = x + \sum_{i=1}^{\infty} \frac{c_i}{x^{2i-1}},$$

where $|c_i|$'s are monotonically decreasing with i. Thus

$$|\rho' - \rho| \leq \lim_{\mu \to \infty} 2\left(\mu + \delta + \sum_{i=1}^{\infty} \frac{c_i}{(\mu + \delta)^{2i-1}} - \left(\mu + \sum_{i=1}^{\infty} \frac{c_i}{\mu^{2i-1}}\right)\right) = 2\delta. \tag{5.4}$$

Combining together (5.3) and (5.4), we obtain that

$$\Delta(\Pi', \Pi) \leq 6\delta = O(\delta).$$

If I is located in the interior of an obstacle edge, then $\|\text{LOC}(I') - \text{LOC}(I)\| = \delta$. Let o_1' be the same as defined before. Notice that $d(o_1, o_1') = \delta$. If Π is of C^+LC^+ or C^-LC^- type,

$$\Delta(\Pi', \Pi) = |d(o_1, o_2) - d(o_1', o_2)| < d(o_1, o_1') = \delta.$$

If Π is a C^+LC^- or C^-LC^+ path, we also bound the difference in path lengths by

$$\Delta(\Pi', \Pi) \leq |\rho' - \rho| + |\alpha' - \alpha| + |\beta' - \beta|,$$

where the notations are the same as defined before. It will be easy to see that $|\alpha' - \alpha| = \angle o_1 o_2 o_1'$. If we look at the triangle $\triangle o_1 o_2 o_1'$, using the facts that $d(o_1, o_1') = \delta$ and $d(o_1, o_2) \geq 2$, we can derive that $\angle o_1 o_2 o_1' \leq \delta$ as we did earlier for the case in which $\text{LOC}(I)$ is an obstacle vertex. Therefore, $|\alpha' - \alpha| \leq \delta$. Similarly, $|\beta' - \beta| \leq \delta$. Again using the fact that $d(o_1', o_1) = \delta$, $|\rho' - \rho| \leq 2\delta$ can be shown in a way similar to the one given above. This completes the proof of the lemma. □

LEMMA 5.2. *Let φ_1 and φ_2 be two obstacle features, and let Π be a CCC path from a position I on φ_1 to a position F on φ_2. Let I' and F' be positions on φ_1 and φ_2, respectively, such that $\Delta(I', I) = \delta_I$ and $\Delta(F', F) = \delta_F$ for any reals $0 \leq \delta_I, \delta_F \leq 1$. Let Π' be the path from I' to F' of the same type as Π. If Π and Π' are neighbors, then*

$$\Delta(\Pi', \Pi) = O(\sqrt{\delta_I} + \sqrt{\delta_F}).$$

Proof. As in the proof of Lemma 5.1, we need only to prove the lemma for the case in which $\delta_I > 0$ and $\delta_F = 0$. Let $\delta = \delta_I$. We assume that I is located at an obstacle vertex; thus $\text{LOC}(I') = \text{LOC}(I)$ and $|\text{VEC}(I') - \text{VEC}(I)| = \delta$. We prove the lemma for the case when Π is a $C^-C^+C^-$ path; the other case, when Π is a $C^+C^-C^+$ path, is symmetric.

Let p (resp., q) be the initial (resp., final) location of the path Π. Let o_i be the center of the unit circle containing the ith C-segment of Π. Consider the triangle $\Delta o_1 o_2 o_3$; see Figure 5.3(a). Without loss of generality, assume that $p, q \notin \triangle o_1 o_2 o_3$; other cases can be handled similarly. Let b_i be the angle of the triangle at o_i. Then

$$\begin{aligned}
\|\Pi\| &= b_1 + \angle p o_1 o_3 + 2\pi - b_2 + b_3 + \angle o_1 o_3 q \\
&= (b_1 + b_3) + 2\pi - b_2 + \angle p o_1 o_3 + \angle o_1 o_3 q \\
&= \pi - b_2 + 2\pi - b_2 + \angle p o_1 o_3 + \angle o_1 o_3 q \\
&= 3\pi - 2b_2 + \angle p o_1 o_3 + \angle o_1 o_3 q.
\end{aligned}$$

Let o_1', o_2', and o_3 be the centers of circles containing the C-segments of Π'. Notice that $\angle o_1' p o_1 = \delta$; see Figure 5.3(b). Let o_3' be the intersection point of the line supporting the segment $o_1' p$ and the line supporting the segment $o_2' o_3$; assume that o_1' is situated so that o_3 lies between o_2' and o_3', and consider the triangle $\Delta o_1' o_2' o_3'$. Let r_i be the angle of the triangle at o_i' and $\gamma = \angle o_2' o_3 o_1 - r_3$ (see Figure 5.3). Then

$$\begin{aligned}
\|\Pi'\| &= r_1 + 2\pi - r_2 + \angle o_2' o_3 o_1 + \angle o_1 o_3 q \\
&= r_1 + 2\pi - r_2 + r_3 + \gamma + \angle o_1 o_3 q \\
&= r_1 + 2\pi - r_2 + r_3 + \angle o_1' p o_1 + \angle p o_1 o_3 + \angle o_1 o_3 q \\
&= (r_1 + r_3) + 2\pi - r_2 + \delta + \angle p o_1 o_3 + \angle o_1 o_3 q \\
&= 3\pi - 2r_2 + \delta + \angle p o_1 o_3 + \angle o_1 o_3 q.
\end{aligned}$$

Thus

$$\Delta(\Pi', \Pi) = |\delta + 2b_2 - 2r_2| \leq \delta + 2|b_2 - r_2|.$$

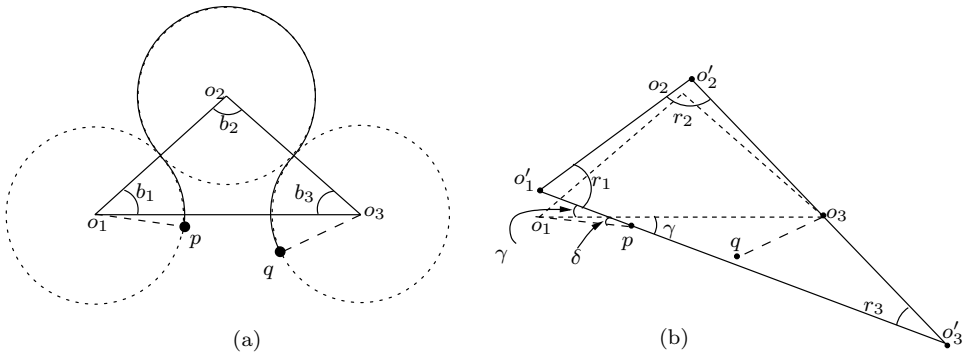


FIG. 5.3. *Bounding the difference in path length for CCC paths.*

Let $\rho = d(o_1, o_3)$ and $\rho' = d(o_1', o_3)$. Applying the cosine law to $\Delta o_1 o_2 o_3$ and $\Delta o_1' o_2' o_3$, and using the fact that $d(o_1, o_2) = d(o_2, o_3) = d(o_1', o_2') = d(o_2', o_3) = 2$, we obtain

$$b_2 = \cos^{-1}\left(\frac{2^2 + 2^2 - \rho^2}{2 \cdot 2 \cdot 2}\right) = \cos^{-1}\left(1 - \frac{\rho^2}{8}\right),$$
$$r_2 = \cos^{-1}\left(\frac{2^2 + 2^2 - \rho'^2}{2 \cdot 2 \cdot 2}\right) = \cos^{-1}\left(1 - \frac{\rho'^2}{8}\right).$$

Define a function

$$f(x) = \cos^{-1}\left(1 - \frac{x^2}{8}\right).$$

Then $|b_2 - r_2| = |f(\rho) - f(\rho')|$. Since $f(x)$ is monotonically increasing for $x \geq 0$,

$$|b_2 - r_2| \leq f(\rho) - f(\rho - \delta).$$

Moreover,

$$\frac{df}{dx} = 1 \Big/ \sqrt{4 - \frac{x^2}{4}}$$

is also monotonically increasing for $x \geq 0$ and $\rho \leq 4$; therefore, $|b_2 - r_2|$ maximizes at $\rho = 4$. Thus

$$\begin{aligned}
|b_2 - r_2| &\leq \cos^{-1}\left(1 - \frac{4^2}{8}\right) - \cos^{-1}\left(1 - \frac{(4-\delta)^2}{8}\right) \\
&\leq \pi - \cos^{-1}(-1 + \delta) \\
&= \pi - 2\cos^{-1}\left(\sqrt{\frac{\delta}{2}}\right) \\
&\leq \pi - 2\left(\frac{\pi}{2} - \sin^{-1}\sqrt{\frac{\delta}{2}}\right) \\
&= 2\sin^{-1}\sqrt{\frac{\delta}{2}} \leq 2\sqrt{2\delta}.
\end{aligned}$$

The last inequality follows froms the fact that $\sin^{-1}(x/2) \le x$ if $x \le 1$. Since $\delta \le 1$, the above inequalities hold.

For $\delta \le 1$, $\delta < \sqrt{\delta}$, and therefore,

$$\Delta(\Pi', \Pi) \le \delta + 2|b_2 - r_2| \le \sqrt{\delta} + 4\sqrt{2}\sqrt{\delta} = (4\sqrt{2}+1)\sqrt{\delta},$$

as desired. ∎

Remark 5.3. (i) Notice that $\Delta(\Pi, \Pi') = O(\sqrt{\delta})$ only if the distance between the centers o_1 and o_3 is almost 4. If $d(o_1, o_3) \le 4 - c$ for some constant $c > 0$, then $\Delta(\Pi, \Pi') \approx \delta/\sqrt{c}$.

(ii) Let φ_I, φ_F be two obstacle features, and for $i = 1, 2$, let Π_1 be a path from a position I_i on φ_I to a position F_i on φ_F. If $\Delta(I_1, I_2), \Delta(F_1, F_2) \le \delta$ for some $\delta < 1$ and Π_1 and Π_2 are neighbors, then the analysis in this section implies Π_2 lies in a tube of width $O(\sqrt{\delta})$ around Π_1.

5.2. Goodness of our approximation.

LEMMA 5.4. *Let $\delta = c\varepsilon^2$, where c is a sufficiently small constant. If there exists an optimal path Π from I to F that is δ-robust, then there exists a path from I to F in the graph G computed in section 4 whose length is at most $(1+\varepsilon)\|\Pi\|$.*

Proof. Let $\Pi = \Pi_1 \| \dots \| \Pi_k$, where each Π_i is a Dubins path of type $\sigma_i \in \mathcal{T}$ from a position X_{i-1} to a position X_i, such that $X_0 = I$, $X_k = F$, and $\text{LOC}(X_i) \in \partial\Omega$ for $0 < i < k$. Since Π is δ-robust, there exist $\chi_0, \dots, \chi_k \in \{+, -\}$ so that Π_i is $\delta^{\chi_{i-1}\chi_i}$-robust. Then, by Lemma 4.2 and the robustness of Π_i, there exist graph nodes $Y_i^{\chi_i}$ for $0 \le i \le k$ such that $\Delta(X_i, Y_i^{\chi_i}) \le \delta$ and such that the Dubins path from $Y_i^{\chi_i}$ to $Y_{i+1}^{\chi_{i+1}}$ of type σ_i is feasible. Therefore, there is an edge from $Y_i^{\chi_i}$ to $Y_i^{\chi_{i+1}}$ in G. Let Π_i' be the path corresponding to this edge in G. $\Pi' = \Pi_1' \| \dots \| \Pi_k'$ is the desired path from I to F in G. To prove the lemma, it suffices to show that $\|\Pi_i'\| \le (1+\varepsilon)\|\Pi_i\|$ for $1 \le i \le k$, provided we choose $\delta = c\varepsilon^2$ small enough.

If Π_i is a CLC path, by Lemma 5.1, $\Delta(\Pi_i', \Pi_i) \le O(\delta)$. Since Π_i is an ε-robust path, its length is at least ε. Therefore,

$$\|\Pi_i'\| \le \|\Pi_i\| + O(\delta) \le \|\Pi_i\| + \varepsilon^2 \le (1+\varepsilon)\|\Pi_i\|,$$

provided the constant c is chosen sufficiently small. If Π_i is a CCC path, by Lemma 5.2, $\Delta(\Pi_i', \Pi_i) \le O(\sqrt{\delta})$. But the length of a CCC path is at least π; therefore,

$$\|\Pi_i'\| \le \|\Pi_i\| + O(\sqrt{\delta}) \le \|\Pi_i\| + \varepsilon \le \left(1 + \frac{\varepsilon}{\pi}\right)\|\Pi_i\| \le (1+\varepsilon)\|\Pi_i\|,$$

provided c is chosen small enough. This completes the proof of the lemma. ∎

Plugging $\delta = O(\varepsilon^2)$ in Theorem 4.8, we obtain the following result.

THEOREM 5.5. *Given a polygonal obstacle environment Ω, an initial position I, a final position F, and a parameter ε, so that there exists an optimal path from I to F that is ε-robust. We can compute in time $O((n^2/\varepsilon^4)\log n)$ a feasible path from I to F whose arc length is at most $(1+\varepsilon)$ times the length of an optimal path.*

Remark 5.6. Recall that the running time of the algorithm is $O((n^2/\varepsilon^4)\log n)$ because we choose $\delta = O(\varepsilon^2)$, and the graph G has $O((n/\delta))$ vertices and in the worst-case every pair of vertices is connected by an edge. If the distance between the centers of initial and final circles of CCC type paths for most pairs of vertices is not close to 4, one can show that it suffices to add edges between $O((n^2/\delta)\log(1/\delta))$ pairs of vertices, and that these pairs can be computed in time $O((n^2/\delta^2)\log(1/\delta))$. In this case the time complexity improves to $O((n^2/\varepsilon^2)\log n \log(1/\varepsilon))$.

6. Computing near optimal robust paths. The path computed by the above algorithm is not necessarily robust because some of the edges in G may not correspond to robust paths. We can compute a graph $G' = (V, E')$, where E' is the set of edges corresponding to $(\varepsilon/2)$-robust paths. An easy argument shows that if δ is chosen correctly, there is an $(\varepsilon/2)$-robust path in G whose length is at most $(1+\varepsilon)$ times the length of an optimal path from I to F if the optimal path is ε-robust.

Next we show that E' can be computed in $O((n^{2.5}/\varepsilon^4)\log n)$ time. For each pair of positions $X, Y \in V$, we compute all $O(1)$ Dubins paths from X to Y, check which of them are $(\varepsilon/2)$-robust, and select the one with the minimum arc length. Recall that a canonical Dubins path is δ-robust if it is $\delta^{\chi\chi'}$-robust for some $\chi, \chi' \in \{-, +\}$. We will show how to determine which of the Dubins paths are $(\varepsilon/2)^{++}$-robust.

Let $\Pi = (\tau_1, \tau_2, \sigma)$ be a canonical Dubins path from an obstacle feature ϕ_1 to another obstacle feature ϕ_2. Let $[\alpha_i, \beta_i]$ denote the feasible domain of ϕ_i for $i = 1, 2$, and let $r_i = \min\{\tau_i + \varepsilon/2, \beta_i\}$. Set $R_\Pi = \{(\tau_1', \tau_2', \sigma) \mid \tau_i \le \tau_i' \le r_i\}$ to be the rectangle in the configuration space. Π is $(\varepsilon/2)^{++}$-robust if R_Π does not contain a singular or infeasible point in its interior. We can check in $O(1)$ time whether R_Π contains a singular point, so it suffices to check whether R_Π contains any infeasible point. Let γ_Π be the union of paths (regarding each path as a set of points) corresponding to the points in R_Π, i.e., the area swept by the paths $(\tau_1', \tau_2', \sigma)$, as we vary $\tau_i' \in [\tau_i, r_i]$. Roughly speaking, γ_Π is the union of at most three regions, each of which is the area swept by a line segment or a circular arc as it moves along an algebraic arc of constant degree. Therefore, the boundary of γ_Π consists of $O(1)$ x-monotone algebraic arcs, each of $O(1)$ degree, and they can be computed in $O(1)$ time. R_Π does not contain any infeasible point if and only if γ_Π does not intersect the interior of Ω. Since the endpositions of Π lie on the obstacle boundary, γ_Π intersects the interior of any obstacle if and only if any of the obstacle edges intersect the interior of γ_Π. We thus have the following intersection-detection problem at hand: Let $\Gamma = \{\gamma_1, \ldots, \gamma_m\}$ be a set of m regions, each of whose boundary consists of $O(1)$ algebraic arcs of constant degrees, and let S be a set of n disjoint line segments in the plane. Report all regions in Γ whose interiors do not intersect any segment of S. We present an $O((m\sqrt{n} + n)\log n)$-time algorithm to report such a subset. Since $m = O(n^2/\varepsilon^4)$ in our case, we conclude that we can compute all $(\varepsilon/2)$-robust paths in time $O((n^{2.5}/\varepsilon^4)\log n)$.

We now present an algorithm for the intersection-detection problem just described. It suffices to describe an $O(n \log n)$-time algorithm for the case when $m = \sqrt{n}$, for otherwise we can partition Γ into $\lceil m/\sqrt{n} \rceil$ subsets, $\Gamma_1, \ldots, \Gamma_s$, each of size at most $\sqrt{n}$, and solve the intersection-detection problem for each Γ_i and S separately. The total running time is obviously $O((m\sqrt{n} + n)\log n)$.

Let E be the set of x-monotone arcs bounding the regions in Γ. A segment $e \in S$ intersects the interior of a region $\gamma \in \Gamma$ if at least one of the following two conditions is satisfied. (i) An endpoint of e lies in the interior of γ, or (ii) e intersects the boundary of γ. It is possible to check both of these conditions for all regions in Γ in $O(n \log n)$ time, using a single sweep-line algorithm. But for the sake of clarity, we explain how to check each of the two conditions separately. The first condition can be checked in $O(n \log n)$ time by a variant of the batched point-location algorithm by Preparata [42]. It basically sweeps a vertical line from left to right and maintains the subset of regions that intersect the sweep-line. Whenever the sweep-line encounters an endpoint p of S, it reports and deletes all the regions of Γ whose interior contains p. These steps can be implemented efficiently using interval trees or segment trees [5]. We omit the rather easy and standard details from here. The total running time of

the algorithm is $O((m^2 + n + k)\log(m + n))$, where k is the number of regions in Γ that contain an endpoint of S. Since $m = \sqrt{n}$, and each region of Γ is a semialgebraic region of constant description complexity, $k = O(m) = O(\sqrt{n})$. Hence, the total time spent is $O(n \log n)$.

Next, we explain how to detect condition (ii). This can be done by modifying the Bentley–Ottman [4] algorithm for segment-intersection reporting, as follows. We sweep a vertical line from left to right and store the arcs of $E \cup S$ intersecting the sweep-line in a height balanced tree T, sorted in y-direction. The algorithm maintains the invariant that none of the arcs of E intersecting the sweep-line intersects any segment of S to the left of the sweep line. A region γ is deleted from Γ as soon as we detect an intersection between $\partial\gamma$ and S; all the boundary arcs of γ are deleted from E as well. The sweep-line stops at the endpoints of $E \cup S$ and the intersection points of arcs in E. At the left (resp., right) endpoint of an arc $e \in E$, we insert e into T (resp., delete e from T). We do the same at the endpoints of S. At an intersection point σ of two arcs $\rho_1, \rho_2 \in E$, we swap the order of ρ_1 and ρ_2 in T. Whenever one of the adjacent element of an active arc $\rho \in E$ changes (because of insertion, deletion, or swapping of two arcs), we check whether the new adjacent element is a segment $e \in S$. If e and ρ intersect, we delete ρ from E and T. Let γ be the region bounded by ρ. We report and delete γ from Γ, and we delete all the edges bounding γ. Note that deletion of these arcs from T may change the adjacent elements of other arcs stored in T, so we have to check for their intersections, but this time can be charged to the arcs deleted. Since each arc of Γ is deleted only once and there are $m = \sqrt{n}$ arcs, the total time spent in this step is $O(\sqrt{n}\log(m + n))$. The sweep-line stops in at most $O(m^2 + n)$ points; hence the overall time spent is also $O(n \log n)$. Putting all the steps together, we obtain the following theorem.

THEOREM 6.1. *Given a polygonal obstacle environment Ω, an initial position I, a final position F so that there exists an optimal path from I to F that is robust, and a parameter ε, we can compute in time $O((n^{2.5}/\varepsilon^4)\log n)$ a feasible $(\varepsilon/2)$-robust path from I to F whose arc length is at most $(1 + \varepsilon)$ times the length of an optimal path.*

7. Conclusion. In this paper we presented an efficient and simple approximation algorithm for computing a curvature-constrained shortest path. The main ingredients of our algorithm are a stronger characterization of curvature-constrained shortest paths, by exploiting their geometry, and a fast and simple algorithm for constructing the graph. We conclude this paper by suggesting a few open problems.

(i) Can one improve the running time of our algorithm to almost linear? Since we are interested only in computing approximate shortest paths, it may be sufficient to construct a small subset of the edges of G. Such techniques have been used to compute approximate unconstrained shortest paths amid obstacles, e.g., [16].

(ii) How fast can one compute an approximate curvature-constrained shortest path in 3-space, especially in view of the recent result by Sussmann [49]?

(iii) Can one develop simple and efficient algorithms for more general constraints, exploiting the geometry of paths?

Acknowledgments. The authors thank John Reif for valuable discussions and Sylvian Lazard and an anonymous referee for several useful comments and for pointing out a few errors in an earlier version of the paper.

REFERENCES

[1] P. K. Agarwal and J. Matoušek, *On range searching with semialgebraic sets*, Discrete Comput. Geom., 11 (1994), pp. 393–418.

[2] P. K. Agarwal, P. Raghavan, and H. Tamaki, *Motion planning for a steering-constrained robot through moderate obstacles*, in Proceedings of the 27th ACM Symposium on Theory of Computing, Las Vegas, NV, 1995, pp. 343–352.

[3] J. Barraquand and J. C. Latombe, *Nonholonomic multibody mobile robots: Controllability and motion planning in the presence of obstacles*, Algorithmica, 10 (1993), pp. 121–155.

[4] J. Bentley and T. Ottmann, *Algorithms for reporting and counting intersections*, IEEE Trans. Comput., 28 (1979), pp. 643–647.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computational Geometry: Algorithms and Applications*, Springer-Verlag, Berlin, 1998.

[6] J. D. Boissonnat and X. N. Bui, *Accessibility Region for a Car that Only Moves Forwards Along Optimal Paths*, Tech. report 2181, INRIA, Sophia Antipolis, France, 1994.

[7] J. D. Boissonnat, A. Cerezo, and J. Leblond, *Shortest paths of bounded curvature in the plane*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 2315–2320.

[8] J. D. Boissonnat, A. Cerezo, and J. Leblond, *A Note on Shortest Paths in the Plane Subject to a Constraint on the Derivative of the Curvature*, Tech. report 2160, INRIA, Sophia Antipolis, France, 1994.

[9] J. D. Boissonnat and S. Lazard, *A polynomial-time algorithm for computing a shortest path of bounded curvature amidst moderate obstacles*, in Proceedings of the 12th ACM Symposium on Computational Geometry, Philadelphia, PA, 1996, pp. 242–251.

[10] X. N. Bui, J. D. Boissonnat, P. Soueres, and J. P. Laumond, *Shortest path synthesis for Dubins non-holonomic robot*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 2–7.

[11] J. Canny, *Some algebraic and geometric configurations in PSPACE*, in Proceedings of the 20th ACM Symposium on Theory of Computing, Chicago, IL, 1988, pp. 460–467.

[12] J. Canny, private communication.

[13] J. Canny, B. Donald, J. Reif, and P. Xavier, *On the complexity of kinodynamic planning*, in Proceedings of the 29th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1988, pp. 306–316.

[14] J. Canny, A. Rege, and J. Reif, *An exact algorithm for kinodynamic planning in the plane*, Discrete Comput. Geom., 6 (1991), pp. 461–484.

[15] J. Canny and J. Reif, *New lower bound techniques for robot motion planning*, in Proceedings of the 28th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1987, pp. 49–60.

[16] K. L. Clarkson, *Approximation algorithms for shortest path motion planning*, in Proceedings of the 19th ACM Symposium on Theory of Computing, ACM, New York, 1987, pp. 56–65.

[17] E. J. Cockayne and G. W. C. Hall, *Plane motion of a particle subject to curvature constraints,* SIAM J. Control, 43 (1975), pp. 197–220.

[18] R. Cole and M. Sharir, *Visibility problems for polyhedral terrains*, J. Symbolic Comput., 7 (1989), pp. 11–30.

[19] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.

[20] B. Donald and P. Xavier, *Near-optimal kinodynamic planning for robots with coupled dynamics bounds*, in Proceedings of the IEEE International Symposium on Intelligent Controls, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 354–359.

[21] L. E. Dubins, *On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents*, Amer. J. Math., 79 (1957), pp. 497–516.

[22] S. Fortune and G. Wilfong, *Planning constrained motion*, Ann. Math. Artificial Intelligence, 3 (1991), pp. 21–82.

[23] T. Fraichard, *Smooth trajectory planning for a car in a structured world*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 2–7.

[24] L. Guibas, M. Overmars, and M. Sharir, *Ray Shooting, Implicit Point Location, and Related Queries in Arrangements of Segments*, Tech. report 433, Department of Computer Science, New York University, New York, 1989.

[25] D. Halperin, L. E. Kavraki, and J.-C. Latombe, *Robotics*, in Handbook of Discrete and Computational Geometry, J. E. Goodman and J. O'Rourke, eds., CRC Press, Boca Raton,

FL, 1997, pp. 755–778.
[26] J. Hopcroft, J. Schwartz, and M. Sharir, eds., *Planning, Geometry, and Complexity of Robot Motion*, Ablex Publishing, Norwood, NJ, 1984.
[27] P. Jacobs and J. Canny, *Planning smooth paths for mobile robots*, in Nonholonomic Motion Planning, Z. Li and J.F. Canny, eds., Kluwer Academic, Norwell, MA, 1992, pp. 271–342.
[28] P. Jacobs, J. P. Laumond, and M. Taix, *Efficient motion planners for nonholonomic mobile robots*, in Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, IEEE Computer Society, Los Alamitos, CA, 1991, pp. 1229–1235.
[29] K. Kedem, R. Livne, J. Pach, and M. Sharir, *On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles*, Discrete Comput. Geom., 1 (1986), pp. 59–71.
[30] V. P. Kostov and E. V. Degtiariova-Kostova, *Suboptimal Paths in the Problem of a Planar Motion with Bounded Derivative of the Curvature*, Tech. report, INRIA, Cedex, France, 1993.
[31] V. P. Kostov and E. V. Degtiariova-Kostova, *The Planar Motion with Bounded Derivative of the Curvature and its Suboptimal Paths*, Tech. Report, INRIA, Cedex, France, 1994.
[32] J. C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Norwell, MA, 1991.
[33] J. C. Latombe, *A fast path-planner for a car-like indoor mobile robot*, in Proceedings of the 9th National Conference on Artificial Intelligence, 1991, pp. 659–665.
[34] J. P. Laumond, *Finding collision free smooth trajectories for a nonholonomic mobile robot*, in Proceedings of the 10th International Joint Conference on Artificial Intelligence, 1987, pp. 1120–1123.
[35] J. P. Laumond, P. E. Jacobs, M. Taix, and R. M. Murray, *A motion planner for nonholonomic mobile robots*, IEEE Trans. Robotics and Automation, 10 (1994), pp. 577–593.
[36] J. P. Laumond and T. Simeon, *Motion Planning for a Two Degrees of Freedom Mobile Robot with Towing*, Tech. report, LAAS/CNRS, LAAS, Toulouse, France, 1989.
[37] J. P. Laumond, M. Taix, and P. Jacobs, *A motion planner for car-like robots based on a global/local approach*, in Proceedings of the IEEE/RSJ International Workshop on Intelligent Robots and Systems, IEEE Computer Society, Los Alamitos, CA, 1990, pp. 765–773.
[38] Z. Li and J. F. Canny, eds., *Nonholonomic Motion Planning*, Kluwer Academic Publishers, Norwell, MA, 1992.
[39] B. Mirtich and J. Canny, *Using skeletons for nonholonomic path planning among obstacles*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1992, pp. 2533–2540.
[40] Y. Nakamura and R. Mukherjee, *Nonholonomic path planning and automation*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 1050–1055.
[41] C. Ó'Dúnlaing, *Motion planning with inertial constraints*, Algorithmica, 2 (1987), pp. 431–475.
[42] F. Preparata, *A note on locating a set of points in a planar subdivision*, SIAM J. Comput., 8 (1979), pp. 542–545.
[43] J. A. Reeds and L. A. Shepp, *Optimal paths for a car that goes both forwards and backwards*, Pacific J. Math., 145 (1990), pp. 367–393.
[44] J. H. Reif, *Complexity of the generalized movers problem*, in Planning, Geometry, and Complexity of Robot Motion, J. Hopcroft, J. Schwartz, and M. Sharir, eds., Ablex Publishing, Norwood, NJ, 1987, pp. 267–281.
[45] J. H. Reif and M. Sharir, *Motion planning in the presence of moving obstacles*, in Proceedings of the 26th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1985, pp. 144–154.
[46] J. H. Reif and S. R. Tate, *Approximate kinodynomic planning using L_2-norm dynamic bounds*, Comput. Math. Appl., 27 (1994), pp. 29–44.
[47] J. T. Schwartz and M. Sharir, *Motion planning and related geometric algorithms in robotics*, in Proceedings of the International Congress of Mathematicians, 1986, pp. 1594–1611.
[48] J. T. Schwartz and M. Sharir, *Algorithmic motion planning in robotics*, in Handbook of Theoretical Computer Science, Vol. A, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 391–430.
[49] H. J. Sussmann, *Shortest 3-dimensional paths with a prescribed curvature bound*, in Proceedings of the 34th Conference on Decision and Control, 1995, pp. 3306–3312.
[50] H. J. Sussmann and G. Tang, *Shortest paths for the Reads–Shepp car; a worked out example of the use of geometric techniques in nonlinear optimal control*, SIAM J. Control Optim., to appear.

[51] G. Wilfong, *Motion planning for an autonomous vehicle*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1988, pp. 529–533.

[52] G. Wilfong, *Shortest paths for autonomous vehicles*, in Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society, Los Alamitos, CA, 1989, pp. 15–20.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1773–1789

ON BIPARTITE DRAWINGS AND THE LINEAR ARRANGEMENT PROBLEM*

FARHAD SHAHROKHI†, ONDREJ SÝKORA‡, LÁSZLÓ A. SZÉKELY§, AND IMRICH VRŤO¶

Abstract. The bipartite crossing number problem is studied and a connection between this problem and the linear arrangement problem is established. A lower bound and an upper bound for the optimal number of crossings are derived, where the main terms are the optimal arrangement values. Two polynomial time approximation algorithms for the bipartite crossing number are obtained. The performance guarantees are $O(\log n)$ and $O(\log^2 n)$ times the optimal, respectively, for a large class of bipartite graphs on n vertices. No polynomial time approximation algorithm which could generate a provably good solution had been known. For a tree, a formula is derived that expresses the optimal number of crossings in terms of the optimal value of the linear arrangement and the degrees, resulting in an $O(n^{1.6})$ time algorithm for computing the bipartite crossing number.

The problem of computing a maximum weight biplanar subgraph of an acyclic graph is also studied and a linear time algorithm for solving it is derived. No polynomial time algorithm for this problem was known, and the unweighted version of the problem had been known to be NP-hard, even for planar bipartite graphs of degree at most 3.

Key words. bipartite drawing, bipartite crossing number, biplanar graph, linear arrangement, approximation algorithms

AMS subject classifications. 68R10, 05C78, 05C95, 94C15

PII. S0097539797331671

1. Introduction.

The crossing number problem calls for placing the vertices of a graph in the plane and drawing the edges with Jordan curves, so that the number of edge crossings is minimized. This problem has been extensively studied in graph theory [27], combinatorial geometry [17], and theory of VLSI [12]. Moreover, this problem has also been studied by the graph drawing community, since one of the most important aesthetic objectives in graph drawing is reducing the number of crossings [18]. In this paper we study the *bipartite crossing number* problem which is an important variation of the crossing number problem. The problem of obtaining drawings of graphs on multiple layers with a small number of edge crossings frequently

*Received by the editors December 23, 1997; accepted for publication (in revised form) September 8, 2000; published electronically January 16, 2001. The research of the second and fourth author was partly done while they were visiting Department of Mathematics and Informatics of University in Passau, team of Prof. F.-J. Brandenburg and was supported in part by the Alexander von Humboldt Foundation and by the Slovak Scientific Grant Agency grant 2/7007/20. A preliminary version of this paper was published at WADS'97 [21]. That version contained slight inaccuracies like missing error terms which are corrected in the current version.

http://www.siam.org/journals/sicomp/30-6/33167.html

†Department of Computer Science, University of North Texas, P.O. Box 13886, Denton, TX 76203-3886 (farhad@cs.unt.edu). The research of the first author was supported by NSF grants CCR-9528228 and CCR-9988525.

‡Department of Computer Science, Loughborough University, Loughborough, Leicestershire LE11 3TU, UK (O.Sykora@lboro.ac.uk). The work of the second author was partly done while visiting LaBRI, Université Bordeaux I, team of Prof. A. Raspaud. He is on a leave from Institute of Mathematics, Bratislava, Slovak Republic.

§Department of Mathematics, University of South Carolina, Columbia, SC 29208 (laszlo@math.sc.edu). The research of the third author was supported in part by Hungarian NSF contracts T 016 358 and T 019 367 and by NSF contract DMS 970 1211.

¶Department of Informatics, Institute of Mathematics, Slovak Academy of Sciences, P.O. Box 56, 840 00 Bratislava, Slovak Republic (vrto@savba.sk).

arises in graph drawing and the design of VLSI [3, 4, 14, 24]. When the number of layers is two, the underlying problem is called the *bipartite drawing* or the *two-layer drawing* problem. Throughout this paper $G = (V_0, V_1, E)$ denotes a connected bipartite graph, where V_0, V_1 are the two classes of independent vertices, and E is the edge set. We will assume that $|V_0 \cup V_1| = n$ and $|E| = m$. A bipartite drawing of G consists of placing the vertices of V_0 and V_1 into distinct points on two parallel lines and then drawing each edge using a straight line segment connecting the points representing the endvertices of the edge. Let $bcr(G)$ denote the *bipartite crossing number* of G, that is, $bcr(G)$ is the minimum number of edge crossings over all bipartite drawings of G.

Computing $bcr(G)$ is NP-hard [8] even when the ordering of vertices in V_0 is fixed [4]. Integer programming methods for computing $bcr(G)$ have been studied by various researchers [10, 15, 26]. These methods, however, do not guarantee polynomial time running times. Moreover, although a polynomial time approximation algorithm with the performance guarantee of $O(\log^4 n)$ times the optimal is known for the crossing number of degree bounded graphs [13], no polynomial time approximation algorithm whose performance is guaranteed has been known for approximating $bcr(G)$. A nice result in this area is a fast polynomial time algorithm which approximates the bipartite crossing number by a factor of 3, when the positions of vertices in V_0 are fixed [4].

In this paper we explore an important relationship between the bipartite crossing number problem and the linear arrangement problem which is another well-known problem in the theory of VLSI [1, 2, 11, 23]. Let δ_G denote the minimum degree of G, a_G denote the arboricity of G, i.e., the minimum number of acyclic graphs that G can be decomposed to, and let $L(G)$ denote the optimal value for the linear arrangement problem. We show (Theorem 3.3) that $bcr(G)$ plus the sum of the square of degrees in G is $\Omega(\delta_G L(G))$. Moreover, we show (Theorem 3.6) that $bcr(G)$ is $O(a_G L(G))$. Our general method for constructing the upper bound is shown to provide for an optimal solution and an exact formula, resulting in an $O(n^{1.6})$ time algorithm for computing $bcr(G)$ when G is a tree (Theorem 5.1). A direct consequence of our results is to obtain the first polynomial time approximation algorithm for $bcr(G)$ with a performance guarantee of $O(\log n)$ from the optimal for a large class of graphs. This class contains all regular graphs, all degree bounded graphs, and all genus bounded graphs, provided that these graphs are not too sparse. We also obtain a polynomial time divide and conquer approximation algorithm in which the divide phase approximately separates the graph, and show that it has the performance guarantee of $O(\log^2 n)$ from the optimal for a variety of graphs. Both algorithms produce drawings in which the coordinates of all vertices are integers so that the sums of edge lengths are also provably near-optimal, with the same approximation factors as for the number of edge crossings. This property of our drawings makes them very appealing to the graph drawing and theory of VLSI.

We also settle the open problem of computing a largest *biplanar* subgraph of an acyclic graph in polynomial time. A bipartite graph $G = (V_0, V_1, E)$ is called a *biplanar* graph if it has a bipartite drawing in which no two edges cross each other. It is known that the problem of determining a largest biplanar subgraph is NP-hard, even when G is planar and the vertices in V_0 and V_1 have degrees at most 3, and at most 2, respectively [5]. No polynomial time algorithm for solving this problem on acyclic graphs had been known, and the proposed heuristics for computing a large biplanar subgraph of an acyclic graph such as the one in [25] would not necessarily compute a largest one. Thus, the question of whether or not a largest biplanar subgraph of an

acyclic graph can be computed in polynomial time had been an open one. Indeed, since the NP-hardness result was shown for very sparse planar graphs [5], it was suggesting that the problem may also be NP-hard for acyclic graphs. Surprisingly, in this paper we present a linear time algorithm for solving the weighted version of this problem in any acyclic graph (Theorem 6.2). The weighted version was first introduced in [15].

Section 2 contains preliminaries and the basic notations. Section 3 contains our main results and explores the relation between $bcr(G)$ and the linear arrangement problem. Section 4 contains the applications and includes several observations on the separator based lower bounds and also contains the approximation algorithms for $bcr(G)$. In particular, a lower bound of $\Omega(\delta_G n b_\beta(G))$ for $bcr(G)$ is derived, where $b_\beta(G)$, $\beta < 1/2$ is the size of the β-separator in G. This lower bound is crucial in verifying the performance guarantee of the divide and conquer algorithm. Section 5 contains our result regarding the bipartite crossing number of a tree T which is expressed exactly in terms of $L(T)$ and the degrees of the vertices, resulting in an $O(n^{1.6})$ time algorithm for computing $bcr(T)$. Finally, section 6 contains our linear time algorithm for computing a largest biplanar subgraph of an acyclic graph.

2. Preliminaries. With the exception of section 6 we consider the connected graphs only under the term graph. Let $G = (V_0, V_1, E)$, $V = V_0 \cup V_1$, and $v \in V$. We denote by d_v the degree of v, and denote by d_v^* the number of vertices of degree 1 which are adjacent to v. We denote by δ_G the minimum degree of G. A bipartite drawing of G is obtained by placing the vertices of V_0 and V_1 into distinct points on two horizontal lines y_0, y_1, respectively, and drawing each edge with one straight line segment. We will assume that y_0 is the line $y = 0$ and y_1 is the line $y = 1$. Any bipartite drawing of G is identified by a coordinate function $h : V_0 \cup V_1 \to \mathbb{R}$, where for any $v \in V_0 \cup V_1$, $h(v)$ is the x coordinate of the vertex v in the drawing. If h is a coordinate function of a bipartite drawing, then the restrictions $h_{|_{V_0}}$ and $h_{|_{V_1}}$ are injections and determine the order in which the vertices are placed on the lines y_0 and y_1, respectively. Conversely, any function $h : V_0 \cup V_1 \to \mathbb{R}$, whose restrictions to V_0 and V_1 are injections, gives rise to a bipartite drawing whose coordinate function is h. Throughout this paper we do not distinguish between a bipartite drawing and its coordinate function and refer to a drawing by referring to its coordinate function.

Let h be a bipartite drawing of $G = (V_0, V_1, E)$. For any $e \in E$, let $bcr_h(e)$ denote the number of crossings of the edge e with other edges. Let $bcr(h)$ denote the total number of edge crossings in h, i.e., $bcr(h) = \frac{1}{2}\sum_e bcr_h(e)$. The bipartite crossing number of G, denoted by $bcr(G)$, is the minimum number of edge crossings over all bipartite drawings of G. Clearly, $bcr(G) = \min_h bcr(h)$.

Given an arbitrary graph $G = (V, E)$, and a function $f : V \to \mathbb{R}$, define the *length* of f, denoted by L_f, to be

$$\sum_{uv \in E} |f(u) - f(v)|.$$

The *linear arrangement* problem is to determine a bijection $f : V \to \{1, 2, 3, \ldots, |V|\}$ of minimum length. This minimum value whose computation is NP-hard [8] will be denoted by $L(G)$. Let h be a bipartite drawing of $G = (V_0, V_1, E)$. Note that the *length* of h, denoted by L_h, is defined as

$$\sum_{uv \in E} |h(u) - h(v)|.$$

Let h be a bipartite drawing of $G = (V_0, V_1, E)$, and let $u_1, u_2, \ldots, u_{d_v}$ be the neighbors of a vertex $v \in V_1$ satisfying $h(u_1) < h(u_2) < \cdots < h(u_{d_v})$. Define the *median vertex* of v, denoted by $med(v)$, to be $u_{\lfloor \frac{d_v}{2} \rfloor}$ if $d_v \geq 2$, and to be u_1 if $d_v = 1$ [4]. We say that h has the *median property* if the following properties hold:

(i) h is an injection;
(ii) $h(V_0) = \{1, 2, \ldots, |V_0|\}$;
(iii) $h(med(v)) + \epsilon \geq h(v) > h(med(v))$, for any $v \in V_1$, where the value of ϵ is positive but "infinitesimally small";
(iv) if $med(x) = med(y)$ for $x \in V_1$ of odd degree and $y \in V_1$ of even degree, then $h(x) < h(y)$.

If a bipartite drawing h does not have the median property, then it can be converted to a drawing which has the property using a process which is called the *median construction*. This process consists of placing the vertices of V_0 in the same order in which they appear in h into the locations $(1, 0), (2, 0), \ldots, (|V_0|, 0)$, and then placing each $v \in V_1$ on a proper position so that the median property holds. Eades and Wormald [4] showed the following remarkable result.

THEOREM 2.1 (see [4]). *Let h be a bipartite drawing of $G = (V_0, V_1, E)$. If h' is obtained from h by the application of the median construction, then*

$$bcr(h') \leq 3bcr(h).$$

3. Linear arrangement and bipartite crossings. In this section we explore the relationship between $bcr(G)$ and $L(G)$.

3.1. Lower bounds. Let h be a bipartite drawing of $G = (V_0, V_1, E)$. Let $e = ab \in E$, and let u be a vertex in $V_0 \cup V_1$ so that $u \notin \{a, b\}$. We say e *covers* u in h if the line parallel to the y axis passing through u has a point in common with the edge e. Note that for any $e = ab$, $a \in V_0, b \in V_1$, neither a nor b are covered by e. However, a vertex $c \in V_1$ with $h(c) = h(a)$ is covered by e. Let $N_h(e)$ denote the number of those vertices in V_1 which are covered by e in h. We will use the following two lemmas later.

LEMMA 3.1. *Let h be a bipartite drawing of $G = (V_0, V_1, E)$.*

(i) *Assume that for any $v \in V_0$, $h(v)$ is an integer. Then there is a bijection $f^* : V_0 \cup V_1 \to \{1, 2, \ldots, n\}$ so that for any $e = ab \in E$*

$$|f^*(a) - f^*(b)| \leq N_h(e) + |h(a) - h(b)| + 1.$$

(ii) *Assume that h has the median property. Then the bijection f^* that satisfies* (i) *also satisfies*

$$L_{f^*} \leq \frac{8bcr(h)}{\delta_G} + L_h + \sum_{a \in V_0} d_a d_a^* + m.$$

Proof. To prove (i), we construct f^* by moving all vertices in V to integer locations. Formally, let $w_1, w_2, \ldots, w_n$ be the order of vertices of $V_0 \cup V_1$ such that $h(w_1) \leq h(w_2) \leq \cdots \leq h(w_n)$. Define $f^*(w_i) = i$, $1 \leq i \leq n$; then (i) easily follows.

For (ii), let $e = ab \in E$, $a \in V_0, b \in V_1$. Consider the case $h(a) > h(b)$. Let $v \in V_1$ be covered by e in h. If $d_v = 1$, then v generates one crossing on e, since v and $med(v)$ are separated by the line segment e in h. On the other hand, if $d_v \geq 2$, then at least $\lfloor d_v/2 \rfloor \geq d_v/4$ of vertices adjacent to v are separated from v in h by the straight line

segment e, since h has the median property. We conclude that any $v \in V_1$ which is covered by e generates at least $\delta_G/4$ crossings on e. Thus, $bcr_h(e) \geq N_h(e)\frac{\delta_G}{4}$.

Now consider the case that $h(a) < h(b)$, and let $v \in V_1$ be a vertex covered by e. Then v generates at least $d_v - \lfloor \frac{d_v}{2} \rfloor \geq d_v/2$ crossings on e provided that v is not a vertex of degree 1 which is adjacent only to a. Consequently, in this case, $bcr_h(e) \geq (N_h(e) - d_a^*)\delta_G/2$.

We conclude that in either case, $bcr_h(e) \geq \frac{1}{4}(N_h(e) - d_a^*)\delta_G$, and hence $N_h(e) \leq \frac{4bcr(e)}{\delta_G} + d_a^*$. Consequently, using (i),

$$|f^*(a) - f^*(b)| \leq \frac{4bcr(e)}{\delta_G} + d_a^* + |h(a) - h(b)| + 1.$$

To finish the proof of (ii) take the sum over all $e = ab \in E$. □

LEMMA 3.2. *Let h be a bipartite drawing of $G = (V_0, V_1, E)$. If h has the median property, then*

$$L_h \leq \epsilon + \sum_{\substack{uv \in E, u \in V_0, v \in V_1 \\ d_v \geq 2}} |h(u) - h(v)|,$$

with an arbitrary small $\epsilon > 0$.

Proof. To prove the claim, let $uv \in E$ with $v \in V_1$ so that $d_v = 1$. Since h has the median property, $med(v) = u$, and thus v is placed arbitrarily close to u in h. Therefore we may assume that $|h(v) - h(u)| \leq \frac{\epsilon}{V_1}$. Thus, the sum of the contributions of all edges which are incident to vertices of degree one in V_1 to L_h is at most $|V_1|\frac{\epsilon}{|V_1|} = \epsilon$, and the claim follows. □

We now prove the main result of this section.

THEOREM 3.3. *Let $G = (V_0, V_1, E)$; then*

$$bcr(G) + \frac{1}{12}\sum_{v \in V} d_v^2 \geq \frac{1}{36}\delta_G L(G).$$

Proof. Let h be a bipartite drawing of G. We will construct an appropriate bijection $f^* : V_0 \cup V_1 \to \{1, 2, \ldots, n\}$. Let h' be a drawing which is obtained by applying the median construction to h. Let $v \in V_1$ with $d_v \geq 2$, and let $u_1, u_2, \ldots, u_{d_v}$ be its neighbors with $h'(u_1) < h'(u_2) < \cdots < h'(u_{d_v})$. Let i be an integer, $1 \leq i \leq \lfloor d_v/2 \rfloor$, and let u be a vertex in V_0 so that $h'(u_i) < h'(u) < h'(u_{d_v - i + 1})$. Observe that u generates d_u crossings on the edges $u_i v$ and $u_{d_v - i + 1} v$ if it is not adjacent to v. Similarly, u generates $d_u - 1$ crossings on the edges $u_i v$ and $u_{d_v - i + 1} v$ if it is adjacent to v. Thus,

$$\begin{aligned} bcr_{h'}(u_i v) + bcr_{h'}(u_{d_v - i + 1} v) &\geq (h'(u_{d_v - i + 1}) - h'(u_i) - 1)\delta_G - d_v \\ &= (h'(u_{d_v - i + 1}) - h'(v) + h'(v) - h'(u_i) - 1)\delta_G - d_v. \end{aligned} \tag{3.1}$$

Note that h' has the median property; thus for $i = 1, 2, \ldots, \lfloor d_v/2 \rfloor$,

$$h'(u_i) < h'(v) < h'(u_{d_v - i + 1})$$

and hence (3.1) implies

$$\begin{aligned} bcr_{h'}(u_i v) + bcr_{h'}(u_{d_v - i + 1} v) \geq (&|h'(v) - h'(u_{d_v - i + 1})| + |h'(v) \\ &- h'(u_i)| - 1)\delta_G - d_v. \end{aligned} \tag{3.2}$$

Using (3.2) observe that for any $v \in V_1$ with $d_v \geq 2$,

$$\sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} (bcr_{h'}(u_i v) + bcr_{h'}(u_{d_v - i+1} v))$$

$$\geq \delta_G \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} (|h'(v) - h'(u_i)| + |h'(v) - h'(u_{d_v-i+1})|) - \delta_G \left\lfloor \frac{d_v}{2} \right\rfloor - \left\lfloor \frac{d_v}{2} \right\rfloor d_v. \tag{3.3}$$

It follows using (3.3) that if $d_v \geq 2$ is even, then

$$\sum_{i=1}^{d_v} bcr_{h'}(u_i v) = \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} (bcr_{h'}(u_i v) + bcr_{h'}(u_{d_v-i+1} v))$$

$$\geq \delta_G \sum_{i=1}^{\lfloor \frac{d_v}{2} \rfloor} (|h'(v) - h'(u_i)| + |h'(v) - h'(u_{d_v-i+1})|) - \delta_G \left\lfloor \frac{d_v}{2} \right\rfloor - \left\lfloor \frac{d_v}{2} \right\rfloor d_v$$

$$= \delta_G \sum_{i=1}^{d_v} |h'(v) - h'(u_i)| - \delta_G \left\lfloor \frac{d_v}{2} \right\rfloor - \left\lfloor \frac{d_v}{2} \right\rfloor d_v. \tag{3.4}$$

Moreover, if $d_v \geq 2$ is odd, then

$$bcr_{h'}(u_{\lfloor \frac{d_v}{2} \rfloor} v) + bcr_{h'}(u_{\lceil \frac{d_v}{2} \rceil} v) \leq \sum_{i=1}^{d_v} bcr_{h'}(u_i v)$$

and

$$bcr_{h'}(u_{\lfloor \frac{d_v}{2} \rfloor} v) + bcr_{h'}(u_{\lceil \frac{d_v}{2} \rceil} v) \geq (h'(u_{\lceil \frac{d_v}{2} \rceil}) - h'(u_{\lfloor \frac{d_v}{2} \rfloor}) - 1)\delta_G,$$

where the upper bound is obvious, and the lower bound holds since no vertex adjacent to v is between $u_{\lceil \frac{d_v}{2} \rceil}$ and $u_{\lfloor \frac{d_v}{2} \rfloor}$. Consequently, if $d_v \geq 2$ is odd, then

$$\sum_{i=1}^{d_v} bcr_{h'}(u_i v) \geq bcr_{h'}(u_{\lfloor \frac{d_v}{2} \rfloor} v) + bcr_{h'}(u_{\lceil \frac{d_v}{2} \rceil} v)$$

$$\geq \delta_G |h'(v) - h'(u_{\lceil \frac{d_v}{2} \rceil})| - \delta_G,$$

where the last line is obtained by observing that $h'(u_{\lceil \frac{d_v}{2} \rceil}) > h'(v) > h'(med(v)) = h'(u_{\lfloor \frac{d_v}{2} \rfloor})$. Combining this with (3.3), for odd d_v, we obtain,

$$2\sum_{i=1}^{d_v} bcr_{h'}(u_i v) \geq \delta_G \sum_{i=1}^{d_v} |h'(v) - h'(u_i)| - \delta_G - \delta_G \left\lfloor \frac{d_v}{2} \right\rfloor - \left\lfloor \frac{d_v}{2} \right\rfloor d_v. \tag{3.5}$$

We note that since (3.5) is weaker than (3.4), it must also hold when d_v is even, and conclude by summing (3.5) over all $v \in V_1$ with $d_v \geq 2$, that

$$4bcr(h') \geq \delta_G \sum_{\substack{uv \in E, v \in V_1 \\ d_v \geq 2}} |h'(v) - h'(u)|$$

$$- \delta_G |V_1| - \delta_G \sum_{v \in V_1} \left\lfloor \frac{d_v}{2} \right\rfloor - \sum_{v \in V_1} \left\lfloor \frac{d_v}{2} \right\rfloor d_v$$

$$\geq \delta_G \sum_{\substack{uv \in E, v \in V_1 \\ d_v \geq 2}} |h'(v) - h'(u)| - 2\sum_{v \in V_1} d_v^2.$$

Using Lemma 3.2, the above inequality gives

$$4bcr(h') \geq \delta_G L_{h'} - \epsilon - 2\sum_{v\in V_1} d_v^2. \tag{3.6}$$

Since h' has the median property, the bijection f^* in Part (ii) of Lemma 3.1 satisfies

$$\delta_G L_{h'} \geq \delta_G L_{f^*} - 8bcr(h') - \delta_G m - \delta_G \sum_{v\in V_0} d_v d_v^*.$$

Observe that if $\delta_G \geq 2$, then $\sum_{v\in V_0} d_v d_v^* = 0$. Hence, the above inequality can be rewritten as

$$\delta_G L_{h'} \geq \delta_G L_{f^*} - 8bcr(h') - \delta_G m - \sum_{v\in V_0} d_v d_v^*.$$

Usage of the above inequality in (3.6) gives

$$12bcr(h') \geq \delta_G L_{f^*} - \delta_G m - \epsilon - \sum_{v\in V_0} d_v d_v^* - 2\sum_{v\in V_1} d_v^2. \tag{3.7}$$

Observing that $L_{f^*} \geq L(G)$, $bcr(h') \leq 3bcr(h)$, $\delta_G m + \epsilon = \epsilon + \sum_{v\in V_0} d_v \delta_G \leq \sum_{v\in V} d_v^2$, and $\sum_{v\in V_0} d_v d_v^* + 2\sum_{v\in V_1} d_v^2 \leq 2\sum_{v\in V} d_v^2$, we obtain

$$36bcr(h) + 3\sum_{v\in V} d_v^2 \geq \delta_G L(G),$$

which finishes the proof. □

Next, we investigate the cases for which the error term $\sum_{v\in V} d_v^2$ can be eliminated from Theorem 3.3.

COROLLARY 3.4. *Let $G = (V_0, V_1, E)$ so that $m \geq (1+\gamma)n$ and $\sum_{v\in V} (d_v - d_v^*)^2 \geq \alpha \sum_{v\in V} d_v^2$, where γ and α are positive constants. Then*

$$bcr(G) \geq C_{\alpha,\gamma}\delta_G L(G), \quad \textit{where} \quad C_{\alpha,\gamma} = \frac{1}{36}\cdot\frac{1}{1+\frac{8+4\gamma}{3\alpha}}.$$

Proof. To prove the result we will first show that for any bipartite drawing h of G,

$$bcr(h) \geq \frac{\sum_{v\in V}(d_v - d_v^*)^2}{16} - m. \tag{3.8}$$

For now assume that (3.8) holds. Since $bcr(G) \geq m - n + 1$ [14] and $n \leq \frac{\gamma}{1+\gamma}m$, we conclude that $m \leq (\gamma+1)bcr(G)$. Combining this inequality with (3.8) gives $(2+\gamma)bcr(G) \geq \frac{1}{16}\sum_{v\in V}(d_v - d_v^*)^2 \geq \frac{\alpha}{16}\sum_{v\in V} d_v^2$, and thus

$$\frac{16(2+\gamma)}{\alpha} bcr(G) \geq \sum_{v\in V} d_v^2,$$

and the claim follows from Theorem 3.3.

To prove (3.8), let h be any bipartite drawing of G, and let $v \in V_0$ so that $d_v - d_v^* \geq 2$. Let $u_1, u_2, \ldots, u_{d_v - d_v^*}$ be the set of vertices of degree at least 2 which are adjacent to v, and assume with no loss of generality that $h(u_1) < h(u_2) < \cdots <$

$h(u_{d_v-d_v^*})$. Let i be an integer, $1 \le i \le \lfloor \frac{d_v-d_v^*}{2} \rfloor$, and note that any vertex u_j, $d_v - d_v^* - i + 1 > j > i$, generates at least one crossing on the edges $u_i v$ and $u_{d_v-i+1}v$. Thus $bcr(vu_i) + bcr(vu_{d_v-d_v^*-i+1}) \ge d_v - d_v^* - 2i$, $1 \le i \le \lfloor \frac{d_v-d_v^*}{2} \rfloor$, and therefore

$$\begin{aligned}\sum_{i=1}^{\lfloor \frac{d_v-d_v^*}{2} \rfloor} bcr_h(u_i v) + bcr_h(u_{d_v-i-d_v^*+1}v) &\ge \sum_{i=1}^{\lfloor \frac{d_v-d_v^*}{2} \rfloor} d_v - d_v^* - 2i \\ &\ge (d_v - d_v^*)\frac{d_v - d_v^* - 1}{2} - \frac{d_v - d_v^*}{2} \cdot \frac{d_v - d_v^* + 2}{2} \\ &\ge \frac{1}{4}(d_v - d_v^*)^2 - d_v. \qquad (3.9)\end{aligned}$$

It follows by summing (3.9) over all $v \in V_1$ that

$$2bcr(h) \ge \frac{\sum_{v\in V_1} (d_v - d_v^*)^2}{4} - 2m.$$

Similarly, we can show that $2bcr(h) \ge (\sum_{v\in V_0}(d_v - d_v^*)^2/4) - 2m$, and hence the claim follows. ∎

3.2. An upper bound. We now derive an upper bound on $bcr(G)$. We need the following obvious lemma.

LEMMA 3.5. *Let h be a bipartite drawing of $G = (V_0, V_1, E)$. Let $e = uv$ and $\bar{e} = ab$ be two edges which cross in h, $u, a \in V_0, v, b \in V_1$. If $|h(v) - h(u)| \ge |h(a) - h(b)|$, then either a or b is covered by e in h. Moreover, if a is covered by e, then*

$$|h(b) - h(u)| \le |h(v) - h(u)|,$$

whereas if b is covered by e, then

$$|h(a) - h(v)| \le |h(v) - h(u)|.$$

Let H be a subgraph of G. We denote by V_H and E_H, respectively, the vertex set and the edge set of H. The arboricity of G, denoted by a_G, is $\max_H \lceil \frac{|E_H|}{|V_H|-1} \rceil$, where the maximum is taken over all subgraphs H, with $|V_H| \ge 2$. Note that $\delta_G/2 \le a_G \le \Delta_G$, where Δ_G denotes the maximum degree of G. A well-known theorem of Nash-Williams [16] asserts that a_G is the minimum number of edge disjoint acyclic subgraphs to which edges of G can be decomposed.

THEOREM 3.6. *Let $G = (V_0, V_1, E)$, and let $f : V_0 \cup V_1 \to \{1, 2, \ldots, n\}$ be a bijection; then*

$$bcr(f) \le 5a_G L_f.$$

In particular,

$$bcr(G) \le 5a_G L(G).$$

Proof. The mapping f is the coordinate function of a bipartite drawing.

Let $e = uv \in E$, $u \in V_0$, $v \in V_1$, and let I_e denote the set of those edges crossing e in the drawing f so that for any $ab \in I_e$,

$$|f(a) - f(b)| \le |f(v) - f(u)|.$$

Observe that if any edge $e' \notin I_e$ crosses e, then $e \in I_{e'}$. Hence, in this case the crossing of e and e' contributes one to $|I_{e'}|$. We conclude that

$$bcr(f) \leq \sum_{e \in E} |I_e|,$$

and will show that $|I_e| \leq a_G(4|f(u) - f(v)| + 1)$. For $e = uv \in E$, with $u \in V_0, v \in V_1$, let V_0^e be the set of all those vertices y of V_0 so that $|f(y) - f(v)| \leq |f(u) - f(v)|$. Similarly, let V_1^e be the set of all those vertices y of V_1 so that $|f(y) - f(u)| \leq |f(u) - f(v)|$. Note that $|V_i^e| \leq 2|f(u) - f(v)| + 1$, $i = 0, 1$, since the coordinates of all vertices are integers. Therefore, we have $|V_0^e \cup V_1^e| \leq 4|f(u) - f(v)| + 2$. Let $\bar{e} = ab \in I_e$, $a \in V_0, b \in V_1$, and observe that by Lemma 3.5, $a \in V_0^e$ and $b \in V_1^e$. Consequently, $|I_e| \leq |E_H|$, where H is the induced subgraph of G on the vertex set $V_0^e \cup V_1^e$. Clearly,

$$|I_e| \leq |E_H| \leq a_G(4|f(u) - f(v)| + 2 - 1) = a_G(4|f(u) - f(v)| + 1)$$

by the definition of a_G, and thus, observing that $L_f \geq m$, we obtain

$$bcr(f) \leq \sum_{e \in E} I_e \leq 5 a_G L_f.$$

To complete the proof we take f to be the optimal solution to the linear arrangement problem, that is, $L_f = L(G)$, and note that $bcr(G) \leq bcr(f)$. ∎

4. Applications. It is easy to provide examples of graphs G for which $bcr(G) = \Theta(\delta_G L(G))$.

PROPOSITION 4.1. *Let $G = (V_0, V_1, E)$ so that $m \geq (1 + \gamma)n$ for a fixed γ. Then*

$$bcr(G) = \Theta(\delta_G L(G)),$$

provided that G satisfies any of the following conditions:

(i) *$\delta_G \geq 2$ and $\delta_G = \Theta(a_G)$,*

(ii) *G is connected and has bounded degrees.*

Proof. If (i) holds, the claim easily follows from Corollary 3.4 and Theorem 3.6. Assume that (ii) holds and k is a constant upper bound on the degrees. Note that $d_v - d_v^* \geq 1$ for any $v \in V$, since G is connected and is not a star, and thus $\sum_{v \in V}(d_v - d_v^*)^2 \geq n$. (Note that the star is excluded by the density condition $m \geq (1 + \gamma)n$.) Now let $\alpha = \frac{1}{k^2}$, then, $n \geq \frac{1}{k^2} \sum_{v \in V} d_v^2$. Hence this graph satisfies the conditions of Corollary 3.4; moreover, it is easy to see that $a_G \leq k = O(1)$, and we conclude using Theorem 3.6 that $bcr(G) = \Theta(L(G))$. ∎

4.1. Bipartite crossings, separators, genus, and page number. The appearance of a_G in the upper bound of Theorem 3.6 relates $bcr(G)$ to other important topological properties of G such as genus of G, denoted by g_G, and page number of G denoted by p_G [27].

PROPOSITION 4.2. *Let $G = (V_0, V_1, E)$, and assume that $\delta_G \geq 2$ and $m \geq (1 + \gamma)n$, for a fixed $\gamma > 0$. Then $bcr(G) = \Theta(L(G))$, provided that either $p_G = O(1)$, or $g_G = O(1)$.*

Proof. Assume that $a_G = O(1)$ is added to the conditions for G; then $\delta_G = O(1)$, and one can conclude using Proposition 4.1 that $bcr(G) = \Theta(L(G))$. To prove the claim, one has to observe that if either $p_G = O(1)$, or $g_G = O(1)$, then $a_G = O(1)$. ∎

Let $0 < \beta \leq \frac{1}{2}$ be a constant and denote by $b_\beta(G)$ size of a smallest β-separator of G. That is,

$$b_\beta(G) = \min_{\beta n \leq |A| \leq (1-\beta)n} |(A, \bar{A})|,$$

where $(A, \bar{A})$ denotes a cut which partitions V into A and $\bar{A}$. Leighton [12] proved that for any degree bounded graph G, $cr(G) + n = \Omega(b^2_{\frac{1}{3}}(G))$, where $cr(G)$ is the planar crossing number of G. Another very interesting consequence of Theorem 3.3 is providing a stronger version of Leighton's result for $bcr(G)$.

THEOREM 4.3. *Let $G = (V_0, V_1, E)$; then, for any constant $0 < \beta < \frac{1}{2}$,*

$$bcr(G) + \sum_{v \in V} d_v^2 = \Omega(\delta_G n b_\beta(G)).$$

Proof. The claim follows from the lower bound in Theorem 3.3 and the well-known observation that $L(G) \geq (1 - 2\beta) n b_\beta(G)$. (See, for instance, [9].) □

Remark. After submission of this paper we derived a weaker version of Theorem 4.3 using Menger's theorem [22].

4.2. Approximation algorithms. Let $G = (V_0, V_1, E)$. The *bipartite arrangement problem* is to find a bipartite drawing h of G with smallest L_h, so that for any $v \in V_0 \cup V_1$, $h(v)$ is an integer. We denote this minimum value by $\bar{L}(G)$. Note that coordinate function h for a bipartite drawing need not to be an injection, since it may be that $h(a) = h(b)$, for some $a \in V_0$, and some $b \in V_1$. Thus, in general $\bar{L}(G) \neq L(G)$. Our approximation algorithms construct bipartite drawings in which all vertices have integer coordinates, so that the number of edge crossings, and at the same time, the length of the drawings are small. We need the following lemma.

LEMMA 4.4. *Let $G = (V_0, V_1, E)$; then*

$$\bar{L}(G) \geq \frac{L(G) - 1}{4}.$$

Proof. Let h be a feasible solution to the bipartite arrangement problem. Let $e = ab \in E$, and note that $N_h(e) \leq |h(a) - h(b)|$, since any vertex in $V_0 \cup V_1$ has an integer x coordinate. Let f^* be the bijection in (i) of Lemma 3.1; then $|f^*(a) - f^*(b)| \leq 2|h(a) - h(b)| + 1$, and hence it follows by taking the sum over all edges, that $L_{f^*} \leq 2L_h + m$. To prove the lemma, we claim that there are at least $\frac{m-1}{2}$ edges $e = ab$, so that $h(a) \neq h(b)$, and consequently $L_h \geq \frac{m-1}{2}$, which implies the result. To prove our claim, note that there are at most $\frac{n}{2}$ edges ab, so that $h(a) = h(b)$, and hence there are at least $m - \frac{n}{2} \geq \frac{m-1}{2}$ edges ab, with $h(a) \neq h(b)$, since G is connected and therefore has at least $n - 1$ edges. □

Even et al. [6] presented polynomial time approximation algorithms with the performance guarantees of $O(\log n \log\log n)$ times the optimal for several NP-hard problems, including the linear arrangement problem. Rao and Richa improved the quality of the approximation to $O(\log n)$ for the linear arrangement problem [19]. Combining the result in [19] with ours, we obtain the following.

THEOREM 4.5. *Let $G = (V_0, V_1, E)$, and let $f : V_0 \cup V_1 \to \{1, 2, \ldots, n\}$ be a bijection which is a $O(\log n)$ times optimal approximate solution to the linear arrangement problem. Then $L_f = O(\bar{L}(G) \log n)$. Moreover, if G meets any of the conditions in Proposition 4.1, then $bcr(f) = O(bcr(G) \log n)$.*

Proof. Note that $L_f = O(L(G) \log n)$ and thus the claim regarding L_f follows from Lemma 4.4. To finish the proof, note that Theorem 3.6 gives $bcr(f) =$

$O(a_G L(G) \log n)$, and the claim regarding $bcr(f)$ is verified by the application of Proposition 4.1. □

A nice algorithmic consequence of the lower bound presented in Theorem 4.3 is that a standard divide and conquer algorithm gives a good approximation for $bcr(G)$ in polynomial time. The divide stage of the algorithm uses a pseudoapproximation algorithm for computing a graph separator [7, 13]. Such a pseudoapproximation algorithm generates a 0.3-separator whose value is $O(\log n)$ times the optimal value of a 1/3-separator [7, 13].

THEOREM 4.6. *Let A be a polynomial time pseudoapproximation algorithm for computing a 0.3-separator with a performance guarantee of $O(\log n)$ times the optimal value of a 1/3-separator. Consider a divide and conquer algorithm which* (a) *recursively separates the graph G in to subgraphs G_1 and G_2, using A,* (b) *obtains the bipartite drawings of G_1 and G_2, and then* (c) *inserts the edges of the separator between these two drawings to obtain a drawing for G. This algorithm generates, in polynomial time, a bipartite drawing h of G with integer coordinates, so that $L_h = O(\bar{L}(G) \log^2 n)$. Moreover, if G meets the conditions in Proposition 4.1, then $bcr(h) = O(bcr(G) \log^2 n)$.*

Proof. Let $\bar{b}(G)$ denote the number of those edges having one endpoint in the vertex set of G_1 and the other in the vertex set of G_2. Let h_1 and h_2 be the bipartite drawings that are obtained for G_1 and G_2, respectively. Let h denote the drawing of G which is obtained by placing h_1 to the left of h_2 and then inserting the edges between G_1, and G_2. The claim for L_h can be verified using arguments similar to those in [9] and employing Lemma 4.1. To verify the claim regarding $bcr(h)$, note that

$$bcr(h) \leq bcr(h_1) + bcr(h_2) + \bar{b}^2(G) + \bar{b}(G)m.$$

Now observing that $m \leq a_G n$, $\bar{b}(G) = O(\log n b_{\frac{1}{3}}(G))$, and that $n b_{\frac{1}{3}}(G) \leq 3L(G)$, we obtain

$$bcr(h) \leq bcr(h_1) + bcr(h_2) + O(a_G L(G) \log n)$$

which implies

$$bcr(h) = O(a_G L(G) \log^2 n).$$

Note that by Proposition 4.1, $bcr(G) = \Theta(a_G L(G))$, and the claim follows. □

Remark. Note that since a_G can be computed in polynomial time, the class of graphs with $a_G \leq c\delta_G$ is recognizable in polynomial time, when c is a given constant. Hence, those graphs which meet the required conditions in Proposition 4.1 can be recognized in polynomial time. Also, note that many important graphs such as those introduced in Proposition 4.2 meet the conditions, and hence for these graphs the performances of both algorithms are guaranteed. Finally note that the lower bound of $\Omega(n\delta_G b_{\frac{1}{3}}(G))$ for $bcr(G)$ has been crucial to verify the suboptimality of the solution in Theorem 4.6.

5. Bipartite crossings in trees. We note that if a_G is small, then the gap between the upper bound in Theorem 3.6 and the lower bound in Theorem 3.3 is small, and hence, it is natural to investigate the case $a_G = 1$, that is, when G is acyclic. In fact, in this case the method in the proof of Theorem 3.6 provides for an optimal bipartite drawing.

THEOREM 5.1. *Let T be a tree on the vertex set $V = V_0 \cup V_1$, where V_0 and V_1 are the partite sets, and $|V| = n$. Let f^* be a bijection utilizing an optimal solution to the linear arrangement problem. Then*

$$bcr(f^*) = bcr(T) = L(T) - n + 1 - \sum_{v \in T} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil . \tag{5.1}$$

Proof. We prove the theorem by induction on n. The result is true for $n = 1, 2$. Let $n \geq 3$. Assume that the theorem is true for all l-vertex trees, $l < n$, and let T be a tree on n vertices. We first show that the right-hand side (RHS) of (5.1) is a lower bound on $bcr(T)$. We then show that $bcr(f^*)$ is equal to the RHS of (5.1). Consider an optimal bipartite drawing h of T. It is not difficult to see that one of the leftmost (rightmost) vertices is a leaf. Denote the left leaf by v_0, the right leaf by v_k, and let $P = v_0 v_1 \dots v_k$ be the path between v_0 and v_k. Note that P will cross any edge in T which is not incident to v_i, $0 \leq i \leq k$. It follows that path P will generate at least

$$c_P = n - 1 - k - \sum_{i=1}^{k-1} (d_{v_i} - 2) \tag{5.2}$$

crossings, where c_P counts exactly the number of edges in T which are not incident to any vertex on P. By deleting the edges of P we obtain trees T_i, on the vertex sets $V^i = V_0^i \cup V_1^i$, rooted in $v_i, i = 1, 2, \dots, k-1$. Consider the optimal bipartite drawings of T_i, $i = 1, 2, \dots, k-1$, and place them consecutively such that the edges in the drawing of T_i do not cross the edges in the drawing of T_j, for $i \neq j$. Then draw the path P without self crossings such that v_0 (v_k) is placed to the left (right) of the drawing of $T_1(T_{k-1})$. Clearly the number of crossings in this new drawing is $\sum_{i=1}^{k-1} bcr(T_i) + c_P$; therefore we conclude that

$$bcr(h) = \sum_{i=1}^{k-1} bcr(T_i) + c_P = \left(\sum_{i=1}^{k-1} bcr(T_i) \right) + n - 1 - k - \sum_{i=1}^{k-1} (d_{v_i} - 2),$$

for otherwise h is not an optimal drawing. For any $v \in V$, let d_v^i denote the degree of v in T_i; applying the inductive hypothesis to T_i, $i = 1, 2, \dots, k-1$, we obtain

$$\begin{aligned} bcr(T) &= \sum_{i=1}^{k-1} \left(L(T_i) - |V^i| + 1 - \sum_{v \in V^i} \left\lfloor \frac{d_v^i}{2} \right\rfloor \left\lceil \frac{d_v^i - 2}{2} \right\rceil \right) \\ &\quad + n - 1 - k - \sum_{i=1}^{k-1} (d_{v_i} - 2) \\ &= \sum_{i=1}^{k-1} \left(L(T_i) - \sum_{v \in V^i} \left(\left\lfloor \frac{d_{v_i}}{2} \right\rfloor \left\lceil \frac{d_{v_i} - 2}{2} \right\rceil + d_{v_i} - 2 \right) \right). \end{aligned} \tag{5.3}$$

Now observe that for $v \in V^i$, $d_v^i = d_v$ if $v \neq v_i$; otherwise $d_v^i = d_v - 2$, $i = 1, 2, \dots, k-1$. Consequently,

$$\begin{aligned} \sum_{v \in V^i} \left\lfloor \frac{d_v^i}{2} \right\rfloor \left\lceil \frac{d_v^i - 2}{2} \right\rceil + d_{v_i} - 2 &= \left\lfloor \frac{d_{v_i} - 2}{2} \right\rfloor \left\lceil \frac{d_{v_i} - 4}{2} \right\rceil + d_{v_i} - 2 \\ &\quad + \sum_{v \in V^i - v_i} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil \end{aligned}$$

$$= \sum_{v \in V^i} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil,$$

where the last line is obtained by observing that

$$\left\lfloor \frac{d_{v_i} - 2}{2} \right\rfloor \left\lceil \frac{d_{v_i} - 4}{2} \right\rceil + d_{v_i} - 2 = \left\lfloor \frac{d_{v_i}}{2} \right\rfloor \left\lceil \frac{d_{v_i} - 2}{2} \right\rceil.$$

Thus it follows using (5.3) that

$$bcr(h) = \sum_{i=1}^{k-1} L(T_i) - \sum_{v \in V} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil. \tag{5.4}$$

Now consider the optimal linear arrangements of the trees T_i, specified by the bijections f_i, $0 \le i \le k$. Place $V^0, V^1, \ldots, V^k$ consecutively on a line so that the vertices in each V^i are placed in the order specified by f_i, $0 \le i \le k$. Let $g : V \to \{1, 2, \ldots, n\}$ denote the bijection associated with this arrangement; then $L_g = \sum_{i=1}^{k-1} L(T_i) + n - 1$. Using this fact (5.4) implies

$$bcr(T) \ge L(T) - n + 1 - \sum_{v \in T} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil,$$

since $L_g \ge L(T)$.

To finish the proof we will show that $bcr(f^*)$ is equal to the RHS of (5.1). Consider an optimal linear arrangement f^* of the tree T. It is not difficult to see that ${f^*}^{-1}(1)$ and ${f^*}^{-1}(n)$ are leaves [1, 20]. Let $P = v_0 v_1 \ldots v_k$ be the path between $v_0 = {f^*}^{-1}(1)$ and $v_k = {f^*}^{-1}(n)$ in T, and let T_i, $1 \le i \le k-1$ be the trees defined in the first part of the proof. Let g be the bijection described earlier; then $L_g = \sum_{i=1}^{k-1} L(T_i) + n - 1$, and thus we conclude that

$$L_{f^*} = L(T) = \sum_{i=1}^{k-1} L(T_i) + n - 1, \tag{5.5}$$

and note that the above equation implies that P does not cross itself, in the arrangement associated with f^*. It follows that P does not cross itself in the bipartite drawing f^*. Let f_i^* be the restriction of f^* to V^i, $i = 1, 2, \ldots, k-1$. Note that $bcr(f^*) = \sum_{i=1}^{k-1} bcr(f_i^*) + c_P$. By applying the induction hypothesis to f_i^*, $i = 1, 2, \ldots, k-1$, we obtain

$$bcr(f^*) = \sum_{i=1}^{k-1} \left(L(T_i) - |V_i| + 1 - \sum_{v \in V_i} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil \right) + c_P. \tag{5.6}$$

Substituting c_P its value from (5.2), and repeating the same steps used in deriving (5.4), we obtain

$$bcr(f^*) = \sum_{i=1}^{k-1} L(T_i) - \sum_{v \in V} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil. \tag{5.7}$$

To complete the proof, we use (5.5) in (5.7) and obtain

$$bcr(f^*) = L(T) - n + 1 - \sum_{v \in T} \left\lfloor \frac{d_v}{2} \right\rfloor \left\lceil \frac{d_v - 2}{2} \right\rceil. \qquad \square$$

Remark. Since the optimal linear arrangement of an n-vertex tree can be found in $O(n^{1.6})$ time [1], Theorem 5.1 implies that computing $bcr(T)$ can also be done in $O(n^{1.6})$ time.

6. Largest biplanar subgraphs in acyclic graphs. Throughout this section $T = (V, E)$ denotes a tree, where $V = V_0 \cup V_1$. Let w_{ij} denote the weight assigned to an edge $ij \in E$. Let $w(B)$ denote the sum of weights for all edges in a subgraph B of T. In this section we present a linear time algorithm to compute a biplanar subgraph of T of largest weight.

A tree is called a *caterpillar* if it consists of a path to which some vertices of degree 1 (leaves) are attached. The path is called the *backbone* of the caterpillar. There are four categories of vertices in a caterpillar. First, consider those caterpillars which are not stars. They have a backbone on at least two vertices, and any vertex on the backbone is of degree at least 2. An endvertex of the backbone is called an *endbone*, any internal vertex of the backbone is called a *midbone*, and any leaf attached to an endbone is called an *endleaf*. Any leaf attached to a midbone is called a *midleaf*.

For a star with at least three vertices, the nonleaf vertex is considered an endbone, the backbone consists of this single endbone, and all leaves are endleaves. If a star has two vertices, then we treat these vertices as endbones. We would not concern ourselves with a star on only one vertex, as it does not play any role in the nontrivial instances of our problem.

Let $T = (V, E)$ and $r \in V$. We may view r as the root of T, and for any $x \in V,\ x \neq r$, define the *parent* of x to be the vertex adjacent to x on the unique path between x and r. For any $x \in V$, the set of *children* of x, denoted by N_x, are those vertices of T whose parent is x. For any $x \in V,\ x \neq r$, let T_x denote the component of T containing x, which is obtained by removing the parent of x from T. We define T_r to be T.

It is well known and easy to show that any graph is biplanar iff all of its connected components are caterpillars. It is further easy to check that any acyclic graph is a caterpillar iff it does not contain a double claw (a star on four vertices whose edges are subdivided) as a subgraph. Hence, any graph is biplanar iff it does not contain a double claw as a subgraph. Let B be a biplanar subgraph of $T = (V, E)$. We say that a caterpillar is in B if this caterpillar is a connected component of B. We say that B *spans* a vertex $a \in V$ if there is an edge ab in B.

For any $x \in V$, let B_x denote the set of all biplanar subgraphs of T_x and $w^*(T_x)$ denote the maximum weight of any biplanar subgraph in B_x. Our goal is to determine $w^*(T_r)$. To achieve this goal, for any $x \in V$, we define five additional related optimization problems as follows:

$$w^1(T_x) = \max_{B \in B_x} \{w(B) : \ \ x \text{ is endleaf of a caterpillar in } B\},$$
$$w^2(T_x) = \max_{B \in B_x} \{w(B) : \ x \text{ is midleaf of a caterpillar in } B\},$$
$$w^3(T_x) = \max_{B \in B_x} \{w(B) : \ x \text{ is endbone of a caterpillar in } B\},$$
$$w^4(T_x) = \max_{B \in B_x} \{w(B) : \ x \text{ is midbone of a caterpillar in } B\},$$
$$w^5(T_x) = \max_{B \in B_x} \{w(B) : \ B \text{ does not span } x\}.$$

It is obvious that for any $x \in V$, $w^*(T_x) = \max_{1 \leq i \leq 5} w^i(T_x)$, and therefore solving all five problems for T_x determines $w^*(T_x)$. For any leaf $v \in V$, we define $w^1(T_v) =$

$w^5(T_v) = 0$, $w^*(T_v) = 0$, and $w^i(T_v) = -\infty$ for $i = 2, 3, 4$. To avoid long displayed equations, for any $x \in V$ and any $u \in N_x$, we denote $w_{xu} + w^5(T_u)$ by $g(u)$ and denote $\max\{w_{ux} + w^5(T_u), w^*(T_u)\}$ by $f(u)$.

LEMMA 6.1.

$$w^1(T_x) = \max_{y \in N_x} \left\{ \left(\sum_{y' \in N_x \setminus \{y\}} w^*(T_{y'}) \right) + w_{xy} + \max_{i=1,3} w^i(T_y) \right\}, \tag{6.1}$$

$$w^2(T_x) = \max_{y \in N_x} \left\{ w_{xy} + w^4(T_y) + \sum_{y' \in N_x \setminus \{y\}} w^*(T_{y'}) \right\}, \tag{6.2}$$

$$\begin{aligned} w^3(T_x) = \max \Bigg\{ & \max_{\substack{y_1, y_2 \in N_x \\ y_1 \neq y_2}} \Bigg\{ w_{xy_1} + \max_{i=1,3} w^i(T_{y_1}) + g(y_2) \\ & + \sum_{y' \in N_x \setminus \{y_1, y_2\}} f(y') \Bigg\}, \max_{y \in N_x} \left\{ g(y) + \sum_{y' \in N_x \setminus \{y\}} f(y') \right\} \Bigg\}, \end{aligned} \tag{6.3}$$

$$\begin{aligned} w^4(T_x) = & \max_{\substack{y_1, y_2 \in N_x \\ y_1 \neq y_2}} \Bigg\{ w_{xy_1} + w_{xy_2} + \max_{i=1,3} w^i(T_{y_1}) + \max_{i=1,3} w^i(T_{y_2}) \\ & + \sum_{y' \in N_x \setminus \{y_1, y_2\}} f(y') \Bigg\}, \end{aligned} \tag{6.4}$$

$$w^5(T_x) = \sum_{y \in N_x} w^*(T_y). \tag{6.5}$$

Proof (sketch). The basic idea for the recurrence relations is to describe how an optimal solution for T_x can be constructed from the optimal solutions for the trees rooted in N_x. Indeed, (6.1), (6.2), and (6.5) are not difficult to see. For (6.3), let x be an endbone of a caterpillar C in $B \in B_x$, $w(B) = w^3(T_x)$. First, consider the case that C is not a star. Since x is an endbone of C, it has at least two neighbors in C, and all but one of its neighbors are leaves in C. Let y_1 be the nonleaf neighbor of x in C and y_2 be an endleaf in C. Then y_1 is either an endbone or an endleaf in $C \setminus \{x\}$, for otherwise B will contain a double claw. This justifies the first two terms in the first inner curly bracket. To justify the third and the fourth terms note that the leaf y_2 will contribute $g(y_2)$, and any $y' \in N_x \setminus \{y_1, y_2\}$ will contribute $f(y')$, respectively, to $w(B)$. To justify the terms in the second inner bracket, consider the case that C is a star. Then C must contain at least one vertex of degree 1, y, which will contribute $g(y)$ to $w(B)$. Furthermore, any $y' \in N_x \setminus y$ is a leaf of C only if $w_{xy'} + w^5(T_{y'}) > w^*(T_{y'})$; otherwise B must contain a maximum biplanar subgraph of T_y which has the weight $w^*(T_y)$. Thus, any $y' \in N_x \setminus y$ will contribute $f(y')$ to $w(B)$. For (6.4), let x be a midbone of a caterpillar C in $B \in B_x$, $w(B) = w^4(T_x)$. Then x has two neighbors y_1 and y_2 which are located on the backbone of C. By deleting x from C, we obtain exactly two caterpillars C_1 and C_2 so that y_i is either an endbone or an endleaf for C_i, $i = 1, 2$, for otherwise B will contain a double claw. Now follow arguments similar to those for deriving (6.3). □

THEOREM 6.2. *For an edge-weighted acyclic graph $T = (V, E)$, a biplanar subgraph of largest weight can be computed in $O(|V|)$ time.*

Proof (sketch). With no loss of generality assume that T is connected, for otherwise we apply our arguments to the connected components of T. We select a root r

for T, perform a post order traversal, and show that for any $x \in V$, $w^i(T_x)$, $1 \leq i \leq 5$, and $w^*(T_x)$ can be computed in $O(|N_x|)$ time if all these quantities are already known for the children of x. This is obvious for $w^*(T_x)$, and it is obvious for $w^5(T_x)$ using (6.5). For $w^1(x)$ and $w^2(x)$ we use (6.1) and (6.2), and note that the expressions in the curly braces can be computed in $O(N_x)$ time if a maximizing y^* is known. Therefore the issue is to find a maximizing vertex $y^* \in N_x$ in $O(N_x)$ time. It is easy to see that for (6.1), y^* must maximize $w_{xy} + \max_{i=1,3} w^i(T_y) - w^*(T_y)$, and for (6.2), $y^* \in N_x$ must maximize $w_{xy} + w^4(T_y) - w^*(T_y)$; these can be computed in $O(|N_x|)$ time.

For computing $w^3(x)$ using (6.3), first note that $y_1^*, y_2^* \in N_x$ can be found in $O(|N_x|)$ time which maximizes $w_{xy_1} + \max_{i=1,3} w^i(T_{y_1}) + g(y_2) + \sum_{y' \in N_x \setminus \{y_1, y_2\}} f(y') = w_{xy_1} + \max_{i=1,3} w^i(T_{y_1}) - f(y_1) + g(y_2) - f(y_2) + \sum_{y' \in N_x} f(y')$. To do so, select distinct vertices $y_1^*, y_2^* \in N_x$ in $O(|N_x)$ time which maximize $w_{xy_1} + \max_{i=1,3} w^i(T_{y_1}) - f(y_1) + g(y_2) - f(y_2)$. This can be done by observing that for any two sequences of numbers, $a_1, a_2, \ldots, a_l$ and $b_1, b_2, \ldots, b_l$, the value of $\max_{i \neq j} a_i + b_j$ can be computed in $O(l)$ time. Finally, note that $y^* \in N_x$ can be computed in $O(|N_x|)$ time to maximize $g(y) + \sum_{y' \in N_x \setminus \{y\}} f(y') = g(y) - f(y) + \sum_{y' \in N_x} f(y')$. To do so, select $y^* \in N_x$ which maximizes $g(y) - f(y)$.

For $w^4(x)$ employ (6.4), and use arguments similar to those employed for computing $w^3(x)$.

It is easy to maintain for every x not just the values $w^i(T_x)$, $w^*(T_x)$, but also the edge sets which realize these values; therefore, we can store the edge set of a maximum-weight biplanar subgraph as well. □

REFERENCES

[1] F. R. K. Chung, *On optimal linear arrangements of trees*, Comput. Math. Appl., 10 (1984), pp. 43–60.

[2] J. Díaz, *Graph layout problems*, in Mathematical Foundations of Computer Science, Lecture Notes in Comput. Sci. 629, Springer-Verlag, Berlin, 1992, pp. 14–23.

[3] J. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis, *Algorithms for drawing graphs: An annotated bibliography*, Comput. Geom., 4 (1994), pp. 235–282.

[4] P. Eades and N. Wormald, *Edge crossings in drawings of bipartite graphs*, Algorithmica, 11 (1994), pp. 379–403.

[5] P. Eades and S. Whitesides, *Drawing graphs in* 2 *layers*, Theoret. Comput. Sci., 131 (1994), pp. 361–374.

[6] G. Even, J. Naor, S. Rao, and B. Schieber, *Divide-and-conquer approximation algorithms via spreading matrices*, in Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, 1995, pp. 62–71.

[7] G. Even, J. S. Naor, S. Rao, and B. Schieber, *Fast approximate graph partition algorithms*, in Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1997, pp. 639–648.

[8] M. R. Garey and D. S. Johnson, *Crossing number is NP-complete*, SIAM J. Algebraic Discrete Methods, 4 (1983), pp. 312–316.

[9] M. Hansen, *Approximate algorithms for geometric embeddings in the plane with applications to parallel processing problems*, in the 30th Annual IEEE Symposium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, 1989, pp. 604–609.

[10] M. Jünger and P. Mutzel, 2 *layer straight line crossing minimization: Performance of exact and heuristic algorithms* , J. Graph Algorithms Appl., 1 (1997), pp. 1–25.

[11] M. Juvan and B. Mohar, *Optimal linear labelings and eigenvalues of graphs*, Discrete Appl. Math., 36 (1992), pp. 153–168.

[12] F. T. Leighton, *Complexity Issues in VLSI*, MIT Press, Cambridge, MA, 1983.

[13] F. T. Leighton and S. Rao, *An approximate max flow min cut theorem for multicommodity flow problem with applications to approximation algorithm*, in the 29th Annual IEEE Sym-

posium on Foundations of Computer Science, IEEE Computer Society Press, Los Alamitos, 1988, pp. 422–431.

[14] M. May and K. Szkatula, *On the bipartite crossing number*, Control Cybernet., 17 (1988), pp. 85–98.

[15] P. Mutzel, *An alternative method to crossing minimization on hierarchical graphs*, in Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1190, Springer-Verlag, Berlin, 1997, pp. 318–333.

[16] C. St. J. A. Nash-Williams, *Edge disjoint spanning trees of finite graphs*, J. London Math. Soc., 36 (1961), pp. 445–450.

[17] J. Pach and P. K. Agarwal, *Combinatorial Geometry*, John Wiley, New York, 1995.

[18] H. Purchase, *Which aesthetic has the greatest effect on human understanding?*, in Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1353, Springer-Verlag, Berlin, 1998, pp. 248–261.

[19] S. Rao and A. Richa, *New approximation techniques for some ordering problems*, in Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SIAM, Philadelphia, 1998, pp. 211–218.

[20] M. A. Seidvasser, *The optimal numbering of the vertices of a tree*, Diskret. Analiz, 19 (1970), pp. 56–74.

[21] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrťo, *On bipartite crossings, largest biplanar subgraphs, and the linear arrangement problem*, in Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 1272, Springer-Verlag, Berlin, 1998, pp. 55–68.

[22] F. Shahrokhi, O. Sýkora, L. A. Székely, and I. Vrťo, *A new lower bound for the bipartite crossing number with algorithmic applications*, in Symposium on Graph Drawing, Lecture Notes in Comput. Sci. 1353, Springer-Verlag, Berlin, 1998, pp. 37–46. Extended version appeared in Theoret. Comput. Sci., 245 (2000), pp. 281–294.

[23] Y. Shiloach, *A minimum linear arrangement algorithm for undirected trees*, SIAM J. Comput., 8 (1979), pp. 15–32.

[24] K. Sugiyama, S. Tagawa, and M. Toda, *Methods for visual understanding of hierarchical systems structures*, IEEE Trans. Systems Man Cybernet., 11 (1981), pp. 109–125.

[25] N. Tomii, Y. Kambayashi, and Y. Shuzo, *On planarization algorithms of the* 2*-level graphs*, IECEJ, EC77-38 (1977), pp. 1–12.

[26] V. Valls, R. Marti, and P. Lino, *A branch and bound algorithm for minimizing the number of crossings arcs in bipartite graphs*, European J. Oper. Res., 90 (1996), pp. 303–319.

[27] A. T. White and L. W. Beineke, *Topological graph theory*, in Selected Topics in Graph Theory, L. W. Beineke and R. J. Wilson, eds., Academic Press, New York, 1978, pp. 15–50.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1790–1801

EDGE-DISJOINT PATHS IN EXPANDER GRAPHS*

ALAN M. FRIEZE†

Abstract. Given a graph $G = (V, E)$ and a set of κ pairs of vertices in V, we are interested in finding, for each pair (a_i, b_i), a path connecting a_i to b_i such that the set of κ paths so found is edge-disjoint. For arbitrary graphs the problem is $\mathcal{NP}$-complete, although it is in $\mathcal{P}$ if κ is fixed.

We present a polynomial time randomized algorithm for finding edge-disjoint paths in an r-regular expander graph G. We show that if G has sufficiently strong expansion properties and r is sufficiently large, then *all* sets of $\kappa = \Omega(n/\log n)$ pairs of vertices can be joined. This is within a constant factor of best possible.

Key words. edge-disjoint paths, expander graphs, randomized algorithms

AMS subject classifications. 68W20, 05C85

PII. S0097539700366103

1. Introduction.

Given a graph $G = (V, E)$ with n vertices and a set of κ pairs of vertices in V, we are interested in finding, for each pair (a_i, b_i) a path connecting a_i to b_i such that the set of κ paths so found is edge-disjoint. This problem has numerous algorithmic applications—in VLSI, for example, or in the context of virtual circuit routing.

For arbitrary graphs the related decision problem is $\mathcal{NP}$-complete, even for planar graphs, although Robertson and Seymour [19], as part of their monumental work on graph minors, have shown that the problem is in $\mathcal{P}$ if κ is fixed. The algorithm of Robertson and Seymour is not considered to be practical and so there has been a lot of work on developing algorithms in various cases, such as planar graphs and grids, where there are restrictions on the placement of the endpoints; see, for example, Frank [5], Wagner and Weihe [21], or Zhou, Tamura, and Nishizeki [22]. Approximation algorithms have been developed for some classes of planar graphs by Kleinberg and Tardos [13].

Another class of graphs which has yielded positive results is expanders. Peleg and Upfal [18] presented a polynomial time algorithm for the case where G is a (sufficiently strong) bounded degree expander graph, and $\kappa \le n^{\epsilon}$ for a small constant ϵ that depends on the expansion property of the graph. This result has been improved and extended by Broder, Frieze, and Upfal [2, 3], Frieze [6], Leighton and Rao [14], and Leighton, Rao, and Srinivasan [15, 16]. In these papers G has to be a (sufficiently strong) bounded degree expander and κ can grow as fast as $n/(\log n)^{\theta}$, where θ depends only on the expansion properties of the input graph, but is at least 2. Also, Kleinberg and Rubinfeld [12] have developed an on-line approximation algorithm.

Let D be the median distance between pairs of vertices in G. Clearly it is not possible to connect more than $O(m/D)$ pairs of vertices by edge-disjoint paths, for all choices of pairs, since some choice would require more edges than all the edges available. In the case of an r-regular expander, this absolute upper bound on κ is $O(n/\log n)$ (assuming r is independent of n). In this paper, we show that if G has

*Received by the editors February 1, 2000; accepted for publication (in revised form) October 6, 2000; published electronically February 23, 2001. This author was supported in part by NSF grant CCR9818411.

http://www.siam.org/journals/sicomp/30-6/36610.html

†Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213 (alan@random.math.cmu.edu).

sufficiently strong *edge* expansion properties and r is sufficiently large, then *all* sets of $\kappa = \Omega(n/\log n)$ pairs of vertices can be joined. This, therefore, is within a constant factor of the optimum. The precise definition of "sufficiently strong" is given after the theorem.

THEOREM 1.1. *Let $G = (V, E)$ be an n-vertex, r-regular graph. Suppose that G is a sufficiently strong edge expander. Then there exist $\epsilon_1, \epsilon_2 > 0$ such that G has the following property: For all sets of pairs of vertices $\{(a_i, b_i) \mid i = 1, \dots, \kappa\}$ satisfying*

(i) $\kappa = \lceil \epsilon_1 rn/\log n \rceil$.

(ii) *For each vertex v, $|\{i : a_i = v\}| + |\{i : b_i = v\}| \le \epsilon_2 r$.*

There exist edge-disjoint paths in G, each of length $O(\log n)$, joining a_i to b_i, for each $i = 1, 2, \dots, \kappa$. Furthermore, there is a polynomial time randomized algorithm for constructing these paths.

ϵ_1, ϵ_2 depend only on certain expansion parameters α, β, γ defined below. They do not depend on n or r. (For example, conditions (1.3) *with $\epsilon = \alpha$,* (2.1), (3.1), *and* (4.1) *suffice.)*

REMARK 1. *The algorithm is a simplification of an algorithm of* [8] *for finding edge-disjoint paths in random regular graphs. There we have a slightly stronger result in that we can take $\kappa = \lceil \epsilon_1 rn/\log_r n \rceil$, i.e., an increase by a factor $\log r$. This is due to the fact that a random r-regular graph has strong vertex expansion. If G has sufficiently strong vertex expansion properties, then we can also take $\kappa = \lceil \epsilon_1 rn/\log_r n \rceil$; see Remark* 2 *below.*

1.1. Preliminaries. We define expanders in terms of edge expansion (a weaker property than vertex expansion).

Let $G = (V, E)$ be a graph and let $n = |V|$. For $S \subset V$ let $\mathrm{out}(S) = \mathrm{out}_G(S)$ be the number of edges with one endpoint in S and one endpoint in $V \setminus S$, that is,

$$\mathrm{out}(S) = \left| \left\{ \{u, v\} \mid \{u, v\} \in E, u \in S, v \notin S \right\} \right|.$$

Similarly,

$$\mathrm{in}(S) = \left| \left\{ \{u, v\} \mid \{u, v\} \in E, u, v \in S \right\} \right|.$$

A graph $G = (V, E)$ is a θ-expander if for every set $S \subset V$, $|S| \le n/2$, we have $\mathrm{out}(S) \ge \theta |S|$.

An r-regular graph $G = (V, E)$ is called an (α, β, γ)-expander if for every set $S \subset V$

$$\mathrm{out}(S) \ge \begin{cases} (1-\alpha) r|S| & \text{if } |S| \le \gamma n, \\ \beta r |S| & \text{if } \gamma n < |S| \le n/2. \end{cases}$$

We naturally assume that $\beta < 1 - \alpha$.

By "sufficiently strong" in Theorem 1.1, we mean that β, γ are arbitrary and α is sufficiently small. Then everything will work provided r is sufficiently large.

Since $2\mathrm{in}(S) + \mathrm{out}(S) = r|S|$ we see that in an (α, β, γ)-expander

$$\mathrm{in}(S) \le \alpha r |S|/2 \qquad \text{when } |S| \le \gamma n. \tag{1.1}$$

In particular, random regular graphs and the (explicitly constructible) Ramanujan graphs of Lubotsky, Phillips and Sarnak [17] are (α, β, γ)-expanders. (See the discussion in [2].)

We have made no real attempt to optimize constants, and, in general, we claim only that inequalities dependent on n or r hold for n or r sufficiently large.

For a graph $G = (V, E)$ and $v \in V$ we let $d_G(v)$ denote the degree of v in G. We use $\delta(G)$ and $\Delta(G)$ to denote the smallest and largest degrees, respectively. For a set $S \subseteq V$ we let $\bar{S} = V \setminus S$ and define its neighbor set, $N_G(S)$, as

$$N_G(S) = \{v \in \bar{S} : \exists w \in S \text{ such that } \{v, w\} \in E\}.$$

For $v \in V$ and $S \subseteq V$ we let $d_G(v, S) = |N_G(v) \cap S|$.

Let $\Phi_S = \text{out}(S)/|S|$ and let the (edge-)expansion $\Phi = \Phi(G)$ of G be defined by

$$\Phi = \min_{\substack{S \subseteq V \\ |S| \le n/2}} \Phi_S.$$

We need an algorithm for splitting a strong expander into nine expander graphs. We could use the algorithm of [2] or [7]. The latter gives a better split and we arbitrarily choose to use it. $\epsilon > 0$ is a small constant. The expansion requirements for the algorithm are

$$\frac{r}{\log r} \ge 63\epsilon^{-2} \text{ and } \Phi \ge 36\epsilon^{-2} \log r, \tag{1.2}$$

which for us means

$$\beta \ge 36\epsilon^{-2} r^{-1} \log r. \tag{1.3}$$

The result we need from [7, Theorem 2] is as follows.

THEOREM 1.2. *Suppose that* (1.2), (1.3) *hold and that G is an r-regular (α, β, γ)-expander, r constant. Then there is a randomized polynomial time algorithm which with probability at least $1 - \delta$ constructs $E_1, E_2, \dots, E_9$ such that the edge-expansion Φ_i of $G_i = (V, E_i)$ satisfies*

$$\Phi_i \ge (1 - \epsilon)\frac{\Phi}{9} - (\alpha + 2\epsilon)\, r$$

for $i = 1, 2, \dots, 9$. This theorem is useful only if Φ is at least a constant multiple of r and α is sufficiently small. This is the case discussed in this paper.

The algorithm of [7] runs in $O(n^{O(\log r)} \log \delta^{-1})$ expected time. The focus of this paper is r constant. If $O(n^{O(\log r)})$ seems excessive, then we can use the linear time algorithm of [2] at the expense of more stringent expansion requirements.

2. The algorithm.

Our algorithm divides naturally into the four phases.

2.1. Phase 0.

Partition G into nine edge-disjoint graphs $G_i = (V, E_i)$, $1 \le i \le 9$. Phase 1 will use the graphs G_1, G_2 to replace the a_i, b_i by randomly chosen $\tilde{a}_i, \tilde{b}_i$; Phase 2 will use the graphs G_3 and G_4. G_3 will be used to try to connect $\tilde{a}_i$ to $\tilde{b}_i$ by a short path. Sometimes we will need G_4 for this task and sometimes we will fail; Phase 3 will use the graphs G_5–G_9 to handle the failures of Phase 2.

The input to our algorithm is a sufficiently strong (α, β, γ)-expander graph G and a set of pairs of vertices $\{(a_i, b_i) \mid i = 1, \dots, \kappa\}$ satisfying the premises of Theorem 1.1. The output is a set of κ edge-disjoint paths, $P_1, \dots, P_\kappa$ such that P_i connects a_i to b_i.

2.2. Phase 0. We start by partitioning G into nine edge-disjoint graphs $G_i = (V, E_i)$, for $1 \le i \le 9$. We use the algorithm SPLIT of Theorem 1.2. We take $\epsilon = \alpha$ in the theorem and assume that

$$\beta > 63\alpha. \tag{2.1}$$

Then each G_i satisfies

$$\Phi_i = \Phi(G_i) \ge \beta_0 r, \tag{2.2}$$

$$\beta_0 r \le \delta(G_i) \le \Delta(G_i) < r, \tag{2.3}$$

where

$$\beta_0 = \frac{\beta}{9} - 4\alpha > 3\alpha > 0. \tag{2.4}$$

2.3. Phase 1. Let $\tilde{A}, \tilde{B}$ be two randomly chosen *disjoint* κ-subsets of V. We are going to replace the problem of finding paths from a_i to b_i by that of finding paths from $\tilde{a}_i$ to $\tilde{b}_i$, where $\tilde{a}_i \in \tilde{A}$ and $\tilde{b}_i \in \tilde{B}$. Let A denote the set $\{a_1, a_2, \ldots, a_\kappa\}$ and $B = \{b_1, b_2, \ldots, b_\kappa\}$.

We connect A to $\tilde{A}$ via edge-disjoint paths in the graph G_1 using network flow techniques. We construct a network as follows:

- Each undirected edge of G_1 gets capacity 1.
- Each $v \in V$ becomes a source of capacity $|\{i : a_i = v\}|$ and each member of $\tilde{A}$ becomes a sink of capacity 1.

Then we find a flow from A to $\tilde{A}$ that satisfies all demands. Since the maximum flow has integer values, it decomposes naturally into $|A|$ edge-disjoint paths (together perhaps with some cycles). If a path joins a_i to $z \in A$, then we let $\tilde{a}_i = z$.

We connect B to $\tilde{B}$ by edge-disjoint paths in a similar manner using G_2.

Thus Phase 1 finds edge-disjoint paths $P_i^{(1)}$ from a_i to $\tilde{a}_i$ and $P_i^{(3)}$ from $\tilde{b}_i$ to b_i, $1 \le i \le \kappa$, where the vertex sets $\tilde{A}, \tilde{B}$ are chosen uniformly at random.

2.4. Phase 2.

2.4.1. Algorithm GENPATHS**.** Our aim is to join $\tilde{a}_i$ and $\tilde{b}_i$ for $i = 1, 2, \ldots, \kappa$ by a *short* path in G_3. After constructing a path, we remove its edges. It is important to ensure that short paths exist. This would not be a problem if we could ensure that G_3 remains an expander throughout. We have to be satisfied with identifying *large* subgraphs $\Gamma_t = (V_t, F_t)$ of $G_t, t = 3, 4$ which are expanders. Initially $\Gamma_3 = G_3, \Gamma_4 = G_4$, and V_3, V_4 lose vertices as the algorithm progresses. We ensure that Γ_3, Γ_4 remain expanders by keeping the degrees of vertices in the Γ_t close to their degree in G_t. This may involve deleting some (low degree) vertices after the construction of a path. We use the routine REMOVE to do this.

If the proposed start vertex v of a walk on Γ_3 does not lie in V_3, then we try to *connect it back* to V_3 by a path in Γ_3. The terminal endpoint of this walk is denoted by v'. We use a subroutine CONNECTBACK for this purpose. We do not expect to succeed all the time and our failures are kept in a set L for later consideration.

The walk from a_i to b_i is then the catenation of paths $P_i^{(t)}$, $t = 1, \ldots, 3$. These walks may each include a short walk W_{CB} at the beginning provided by CONNECTBACK.

1. **Algorithm** GenPaths
2. **begin**
3. $\Gamma_i \leftarrow G_i$, $i = 3, 4$.
4. **for** $i = 1$ **to** κ **do**
5. **Execute** REMOVE(Γ_3)
6. **Execute** ConnectBack($V_3, \tilde{a}_i, \tilde{a}'_i, i, W_{CB}^{(3a)}$)
7. **Execute** ConnectBack($V_3, \tilde{b}_i, \tilde{b}'_i, i, W_{CB}^{(3b)}$)
8. **if** $i \notin L$ **then**
9. Construct a shortest path Q_i from $\tilde{a}'_i$ to $\tilde{b}'_i$ in Γ_3.
10. $P_i^{(2)} \leftarrow (W_{CB}^{(3a)}, Q_i, W_{CB}^{(3b)})$
11. $\Gamma_3 \leftarrow \Gamma_3 \setminus E(P_i^{(2)})$
12. **fi**
13. **od**
14. **end** GenPaths

2.4.2. Subroutine REMOVE. The purpose of REMOVE is to delete vertices which might prevent a graph from being an expander. In GENPATHS we apply REMOVE to Γ_3 or Γ_4.

In step 4 we remove the set of vertices R_0 which have so far lost more than $\beta_0 r/4$ edges through the deletion of shortest paths. We then iteratively (steps 5–12) remove vertices which have at least $\beta_0 r/4$ neighbors among previously removed vertices. We therefore see that for $t = 3, 4$

$$v \in V_t \text{ implies } d_{\Gamma_t}(v) \geq d_{G_t}(v) - \beta_0 r/2 \geq \beta_0 r/2. \tag{2.5}$$

1. **Algorithm** REMOVE(Γ_t)
2. **begin**
3. $R_0 = \{v \in V_t : \ d_{\Gamma_t}(v) < d_{G_t}(v) - \beta_0 r/4\}$.
4. $\ell \leftarrow 0$.
5. **begin**
6. $\bar{R}_\ell \leftarrow V_t \setminus R_\ell$.
7. $d \leftarrow \max_v \{d_{\Gamma_t}(v, R_\ell) : \ v \in \bar{R}_\ell\}$.
8. **if** $d \leq \beta_0 r/4$ **terminate** REMOVE, otherwise
9. $R_\ell \leftarrow R_\ell \cup \{w\}$; $V_t \rightarrow V_t \setminus \{w\}$ where $w \in \bar{R}_\ell$ is such that $d_{\Gamma_t}(w, R_\ell) = d$.
10. $\ell \leftarrow \ell + 1$
11. **goto** 6.
12. **end**
13. **end** REMOVE

We can see from (2.5) that throughout the algorithm

$$\Phi_{\Gamma_t} \geq \Phi_t - \beta_0 r/2 \geq \beta_0 r/2 \qquad \text{for } t = 3, 4. \tag{2.6}$$

Indeed, (2.5) implies that for $S \subseteq V_t$ we have

$$\text{out}_{\Gamma_t}(S) \geq \text{out}_{G_t}(S) - \beta_0 r|S|/2 \geq (\Phi_t - \beta_0 r/2)|S|.$$

2.4.3. Subroutine ConnectBack**.** The purpose of ConnectBack is to connect a vertex x to V_3 by means of a random walk. (If $x \in V_3$ already, then ConnectBack does nothing except to relabel x as x'.) All walks are done on V_4 and in step 5 we check that the start point x lies in V_4. If not, we put i into L, where $x = \tilde{a}_i$ or $\tilde{b}_i$. Edge-disjoint paths for the pairs $(a_i, b_i), i \in L$ are found in Phase 3. Let

$$\omega = \lceil \log \log n \rceil^2.$$

We do a random walk W_{CB} from x until we reach V_3 or make ω steps. In the latter case we add the corresponding i to L.

1. **subroutine** ConnectBack(V_3, x, x', i, W_{CB})
2. **begin**
3. **if** $x \in V_3$ **then** $x' \leftarrow x$ **exit fi else**
4. **Execute** remove(Γ_4)
5. **if** $x \notin V_4$ **then** $L \leftarrow L \cup \{i\}$ **exit fi else**
6. Do a random walk W_{CB} starting at x in Γ_4, until V_3 is reached or ω steps have been taken.
7. In the latter case $L \leftarrow L \cup \{i\}$ and we **exit else**
8. $\Gamma_4 \leftarrow \Gamma_4 \setminus W_{CB}$
9. **end** ConnectBack

2.5. Phase 3. There is still the set L of pairs (a_i, b_i) which have not been connected by paths. We will show later that with probability at least $1 - o(1)$, $|L|$ is at most $n/(\log n)^4$. As such, these pairs can be dealt with by the algorithm of [6], using graphs G_5–G_9. It is convenient to join the $\tilde{a}_i, \tilde{b}_i$ as then we are guaranteed to have distinct endpoints.

3. Analysis of Phase 1. In this section we show that if (2.2) holds and

$$\beta_0 r \geq 1 \text{ and } \epsilon_2 \leq \beta_0, \tag{3.1}$$

then after we run Split, we can find edge-disjoint paths from a_i to $\tilde{a}_i$ in G_1 and edge-disjoint paths from b_i to $\tilde{b}_i$ in G_2, for $1 \leq i \leq \kappa$, for *any* choice of $a_1, \ldots, b_\kappa$ consistent with the premises of Theorem 1.1, and every choice for $\tilde{a}_1, \ldots, \tilde{a}_\kappa, \tilde{b}_1, \ldots, \tilde{b}_\kappa$.

Let A and $\tilde{A}$ be as defined in section 2.3. For $S \subseteq V$, let

$$\alpha(S) = |\{i : \ a_i \in S\}| \text{ and } \xi(S) = |S \cap \tilde{A}|.$$

For sets $S, T \subseteq V$, let $e_{G_1}(S, T)$ denote the number of edges of G_1 with an endpoint in S and the other endpoint in T. It suffices to prove that

$$e_{G_1}(S, \bar{S}) \geq \xi(\bar{S}) - \alpha(\bar{S}) \qquad \forall S \subseteq V. \tag{3.2}$$

Given (3.2), the existence of the required flow in G_1 is a special case of a theorem of Gale [9] (see Bondy and Murty [1, Theorem 11.8], in which case we see that (3.2) implies a successful run of Phase 2.

Now

$$\alpha(\bar{S}) = \kappa - \alpha(S) \geq \kappa - \epsilon_2 r|S|$$

and so

$$\xi(\bar{S}) - \alpha(\bar{S}) \leq |\tilde{A} \cap \bar{S}| - \kappa + \epsilon_2 r|S| \leq \epsilon_2 r|S|.$$

Thus (2.2) verifies (3.2) for $|S| \leq n/2$ provided we have $\epsilon_2 \leq \beta_0$. For $|S| > n/2$ we have $\Phi_1 \geq 1$ and then

$$e_{G_1}(S, \bar{S}) = e_{G_1}(\bar{S}, S) \geq \Phi_1|\bar{S}| \geq |\tilde{A} \cap \bar{S}| \geq |\tilde{A} \cap \bar{S}| - \kappa + \alpha(S) = \xi(\bar{S}) - \alpha(\bar{S});$$

and so Phase 1 succeeds with respect to $A, \tilde{A}$. The same argument applies to $B, \tilde{B}$. To ensure these paths are of length $O(\log n)$ we can solve a minimum cost maximum flow problem as indicated in Kleinberg and Rubinfeld [12].

4. Analysis of Phase 2. We first prove that V_3 stays large.

LEMMA 4.1. *Throughout the algorithm*

$$|V_3| \geq (1-\gamma_0)n,$$

where

$$\gamma_0 = \frac{\beta_0\gamma}{10}.$$

Proof. We know from (2.6) that Γ_3 is a $(\beta_0 r/2)$-expander throughout the execution of Phase 2. We can use the strong edge-expansion of Γ_3 to prove some vertex-expansion and conclude the diameter of Γ_3 is at most $\tau_0 = \lceil 2\log_{1+\beta_0/2} n\rceil + 1$. Indeed, in a θr-expander, every set S, $|S| \leq n/2$ has at least $\theta|S|$ neighbors. Thus the total number of edges in the paths that are removed from G_3 is $\leq \kappa\tau_0$. Hence the vertices B_1 of G_3 which are incident with $\beta_0 r/4$ edges of these paths satisfy

$$|B_1| \leq \frac{4\kappa\tau_0}{\beta_0 r} \leq \frac{\gamma_0 n}{3}$$

provided

$$\epsilon_1 \leq \frac{\beta_0^2\gamma}{250}\log\left(1+\frac{\beta_0}{2}\right), \tag{4.1}$$

where ϵ_1 is as in the statement of Theorem 1.1.

Let $X = \{x_1, x_2, \dots,\}$ be the remaining vertices removed by REMOVE. We claim that if $|B_1| \leq \frac{\gamma_0}{3}n$, then $|X| \leq 2|B_1| \leq \frac{2\gamma_0}{3}n$ implying that $|V_3| \geq (1-\gamma_0)n$ and the lemma follows.

Indeed, if $X_i = \{x_1, x_2, \dots, x_i\}$, then $X_i \cup B_1$ has $i + |B_1|$ vertices and contains at least $i\beta_0 r/4$ edges. The existence of x_i, $i = 2|B_1|$ contradicts (1.1) with $S = X_i \cup B_1$. Therefore

$$\text{in}(S) \geq |B_1|\beta_0 r/2 \geq |S|\beta_0 r/6 > |S|\alpha r/2$$

using (2.4). □

Our next task is to bound the size of the set L of pairs of vertices which are left to Phase 3. For this we need to establish some facts about random walks on graphs.

4.1. Random walks. A *random walk* on an undirected graph $G = (V, F)$ is a Markov chain $\{X_t\}$ on V associated with a particle that moves from vertex to vertex according to the following rule: The probability of a transition from vertex v, of degree d_v, to a vertex w is $1/(2d)_v$ if $\{v, w\} \in E$ and 0 otherwise. The particle stays at v with probability 1/2. This removes the possibility of periodicity and allows us to use the conductance bound of Jerrum and Sinclair. Its stationary distribution, denoted by π, is given by $\pi(v) = \frac{d_v}{2|E|}$ for $v \in V$.

Let P be the transition matrix of the associated Markov chain. Let λ be the second largest eigenvalue of P. According to Sinclair and Jerrum [20]

$$\lambda \leq 1 - \frac{\Psi^2}{2}, \tag{4.2}$$

where Ψ denotes the *conductance* of a random walk on G.

Here

$$\begin{aligned}\Psi &= \min_{\pi(S)\le 1/2} \frac{1}{\pi(S)} \sum_{\substack{v\in S\\ w\notin S}} \pi(v)P(v,w)\\ &\ge \min_{\pi(S)\le 1/2} \frac{1}{\pi(S)} \sum_{\substack{v\in S\\ w\notin S}} \frac{d_v}{\Delta|V|}\cdot\frac{1}{2d_v}\\ &= \min_{\pi(S)\le 1/2} \frac{\mathrm{out}(S)}{2\Delta|V|\pi(S)}\\ &\ge \frac{\Phi\delta}{2\Delta^2}. \qquad (4.3)\end{aligned}$$

Another fact we will need is

$$|P^t(v,w)-\pi(w)| \le \sqrt{\frac{\Delta}{\delta}}\lambda^t. \qquad (4.4)$$

A proof of this can be found, for example, in [20].

We also need a large deviation result. This can be taken from the works of Dinwoodie [4], Gillman [10] and Kahale [11]. We quote the consequences of Theorem 2.1 of [10]: Let q be the distribution of the start vertex of a random walk on a graph G. Let S be a set of vertices of G. Let Y denote the number of visits to S in the first t steps.

$$\mathbf{Pr}(Y - t\pi(S) \le -u) \le \left(1+\frac{(1-\lambda)u}{10t}\right) N_q e^{-(1-\lambda)u^2/(20t)}, \qquad (4.5)$$

where

$$N_q = \left(\sum_{v\in V}\frac{q(v)^2}{\pi(v)}\right)^{1/2}.$$

4.2. Analysis of CONNECTBACK**.** Arguing as in Lemma 4.1 we see that since $\kappa\omega = o(n)$ we will have $|V_4| = n - o(n)$ throughout the algorithm. Furthermore the minimum and maximum degrees of Γ_4 will satisfy

$$\beta_0 r/2 \le \delta \le \Delta \le r.$$

Consequently Γ_4 has at least $(1-o(1))\beta_0 rn/2$ edges and then for sufficiently large n, the steady state for a random walk on Γ_4 will always satisfy

$$\frac{\beta_0}{3n} \le \pi(v) \le \frac{3}{\beta_0 n} \qquad \forall v\in V_4.$$

From (2.6) and (4.3) we see that the conductance Ψ satisfies

$$\Psi \ge \frac{\beta_0^2}{8}. \qquad (4.6)$$

Let $L = L_5 \cup L_7$, where L_θ consists of the indices added to L in step θ of CONNECTBACK.

To probabilistically bound $|L_5|$ we first bound the expected value of the number M_t of vertices which are incident with five or more CONNECTBACK walks in Step t of

GENPATHS, $t = 6, 7$. Fix a t and enumerate these walks as $W_1, W_2, \dots, W_m$, $m \le \kappa$. Then for $c = 9/\beta_0^2$,

$$\begin{aligned}\mathbf{E}(M_t) &\le \sum_{i_1,\dots,i_5,v} \mathbf{Pr}(W_{i_t},\, t = 1,\dots,5 \text{ go through } v)\\ &\le \binom{\kappa}{5} n \left(\frac{c\,\omega}{n - o(n)}\right)^5 = O\left(\frac{\omega^5 n}{(\log n)^5}\right). \qquad (4.7)\end{aligned}$$

It is important to note here that regardless of the history of Phase 1, $\tilde{A}$ is a random set and we can assume that $\tilde{a}_1, \tilde{a}_2, \dots, \tilde{a}_\kappa$ is a random ordering of its elements. (Phase 1 may cause some correlation between the orderings of $\tilde{A}, \tilde{B}$, but we compute the contribution of each set separately.)

Explanation of (4.7). We first show that $c\omega/(n - o(n))$ bounds the probability that walk $W_{i_t} = (x_1, x_2, \dots, x_m)$ passes through v, given $W_{i_s}, 1 \le s < t$ pass through v. Observe that x_1 is chosen randomly from $V_3 \setminus X$, where X is the set of start vertices of $W_{i_s}, 1 \le u < t$. Therefore

$$\mathbf{Pr}(x_1 = v) \le \frac{1}{n - o(n)} \le \frac{3}{\beta_0}\pi(v).$$

Then by induction on j we get

$$\mathbf{Pr}(x_j = v) = \sum_{w \in V_4} \mathbf{Pr}(x_{j-1} = w)P(w, v) \le \sum_{w \in V_4} \frac{3}{\beta_0}\pi(w)P(w, v) = \frac{3}{\beta_0}\pi(v) \le \frac{9}{\beta_0^2 n}$$

and we have the claimed bound of $\frac{9\omega}{\beta_0^2 n}$ for the (conditional) probability that W_{i_s} goes through v. There are at most $\binom{\kappa}{t}$ choices for $W_{i_s}, 1 \le s < t$ and n choices for v and (4.7) follows.

It follows from (4.7) and the Markov inequality that

$$\mathbf{Pr}\left(M_t \ge \frac{n}{8(\log n)^4}\right) = o(1).$$

In addition to these $M = M_6 + M_7$ vertices we consider those vertices which are removed by REMOVE(Γ_4). Arguing as in Lemma 4.1 we see that if $M = o(n)$, then $|L_5| \le 3M$. We deduce that

$$\mathbf{Pr}\left(|L_5| \ge \frac{3n}{4(\log n)^4}\right) = o(1). \qquad (4.8)$$

We now estimate the probability that $i \in L_7$. We apply (4.5) with $G = \Gamma_4$, $S = V_3 \cap V_4$, and $q(v) = \frac{1}{|V_4|}$ for $v \in V_4$. Then we have

$$1 - \lambda \ge \frac{\beta_0^4}{64},\ N_q \le \left(\frac{3}{\beta_0}\right)^{1/2},\ \text{and } \pi(S) \ge 1 - \frac{3\gamma_0}{\beta_0} \ge \frac{1}{2}.$$

Putting $t = \omega$ and $u = t\pi(S)$ we get

$$\mathbf{Pr}(W \cap V_3 = \emptyset) = O(e^{-\beta_0^4\omega/5120})$$

and so

$$\mathbf{E}(|L_7|) = O(e^{-\beta_0^4\omega/5120}\kappa)$$

and

$$\mathbf{Pr}\left(|L_7| \geq \frac{n}{4(\log n)^4}\right) = o(1).$$

Combining this with (4.8) we see that $\mathbf{Pr}(|L| \geq n/(\log n)^4) = o(1)$.

REMARK 2. *Suppose G has the following vertex expansion property for small sets: $S \subseteq V$, $|S| \leq \frac{\gamma}{r}n$ implies that $|N_G(S)| \geq (1-\alpha)r|S|$. The algorithm of [7] can be modified to split the graph so that each subgraph G_i satisfies $|N_{G_i}(S)| \geq \frac{1-10\alpha}{8}r|S|$. Then the shortest paths in Γ_3 will be of length $O(\log_r n)$ and the claim in Remark 1 will follow.*

5. Analysis of Phase 3. We join the pairs in L using the algorithm of [6]. The algorithm is capable of joining $\Omega(n/(\log n)^{2+o(1)})$ *distinct* pairs, provided the graph has sufficient edge-expansion. Notice that $\tilde{a}_i, \tilde{b}_i$ are chosen as distinct vertices. We briefly describe how we can make this algorithm route $m \leq \frac{n}{(\log n)^4}$ pairs using the graphs G_5–G_9, assuming only that $\Phi_5, \ldots, \Phi_9 \geq 1$. Let $\lambda = \lceil \log n \rceil$.

(a) The aim here is to choose w_j, W_j, $1 \leq j \leq 2m$ such that (i) $w_j \in W_j$, (ii) $|W_j| = \lambda + 1$, (iii) the sets $W_j, 1 \leq j \leq 2m$ are pairwise disjoint, and (iv) W_j induces a connected subgraph of G_6.

As in [14] we partition an arbitrary spanning tree T of G_6. Since T has maximum degree at most r we can find $2m$ vertex disjoint subtrees T_j, $1 \leq j \leq 2m$ of T, each containing between $\lambda+1$ and $(r-1)\lambda+2$ vertices. We can find T_1 as follows: choose an arbitrary root ρ and let $Q_1, Q_2, \ldots, Q_\sigma$ be the subtrees of ρ. If there exists l such that Q_l has between $\lambda+1$ and $(r-1)\lambda+2$ vertices, then we take $T_1 = Q_l$. Otherwise we can search for T_1 in any Q_ℓ with more than $(r-1)\lambda+2$ vertices. Since $T \setminus T_1$ is connected, we can choose all of the T_j's in this way. Finally, W_j is the vertex set of an arbitrary $\lambda+1$ vertex subtree of T_j and w_j is an arbitrary member of W_j for $j = 1, 2, \ldots, 2m$.

(b) Let S_A, S_B denote the set of sources and sinks that need to be joined. Using a network flow algorithm in G_5 connect in an arbitrary manner the vertices of $S_A \cup S_B$ to $W = \{w_1, \ldots, w_{2m}\}$ by $2m$ edge-disjoint paths. The expansion properties of G_5 ensure that such paths always exist.

Let $\hat{a}_k$ (resp., $\hat{b}_k$) denote the vertex in W_i that was connected to the original end-point $\tilde{a}_k$ (resp., $\tilde{b}_k$). Our problem is now to find edge-disjoint paths joining $\hat{a}_k$ to $\hat{b}_k$ for $1 \leq k \leq m$.

(c) If w_t has been renamed as $\hat{a}_k$ (resp., $\hat{b}_k$), then rename the elements of W_t as $\hat{a}_{k,\ell}$ (resp., $\hat{b}_{k,\ell}$), $1 \leq \ell \leq \lambda$. Choose $\xi_j, 1 \leq j \leq \lambda m$ and $\eta_j, 1 \leq j \leq \lambda m$ independently at random from the steady state distribution π of a random walk on G_9. Using a network flow algorithm as in (b), connect $\{\hat{a}_{k,\ell} : 1 \leq k \leq m, 1 \leq \ell \leq \lambda\}$ to $\{\xi_j : 1 \leq j \leq \lambda m\}$ by edge-disjoint paths in G_7. Similarly, connect $\{\hat{b}_{k,\ell} : 1 \leq k \leq m, 1 \leq \ell \leq \lambda\}$ to $\{\eta_j : 1 \leq j \leq \lambda m\}$ by edge-disjoint paths in G_8. Rename the other endpoint of the path starting at $\hat{a}_{k,\ell}$ (resp., $\hat{b}_{k,\ell}$) as $a^*_{k,\ell}$ (resp., $b^*_{k,\ell}$). Once again the expansion properties of G_7, G_8 ensure that flows exist.

(d) Choose $x^*_{k,\ell}, 1 \leq k \leq m, 1 \leq \ell \leq \lambda$ independently at random from the steady state distribution π of a random walk on G_9. Let $W'_{k,\ell}$ (resp., $W''_{k,\ell}$) be a random walk of length $\tau = \theta \log n$ from $a^*_{k,\ell}$ (resp., $b^*_{k,\ell}$) to $x^*_{k,\ell}$. Here θ is sufficiently large that a random walk of this length on G_9 is "well mixed," e.g., until the variational distance between the τ-step distribution and the steady state is $O(n^{-3})$. That $\theta = O(1)$ follows from (4.4). The use of this intermediate vertex $\hat{x}_{k,\ell}$ helps to break some conditioning caused by the pairing up of the flow algorithm.

Let B'_k (resp., B''_k) denote the *bundle* of walks $W'_{k,\ell}, 1 \le \ell \le \lambda$ (resp., $W''_{k,\ell}, 1 \le \ell \le \lambda$). Following [16] we say that $W'_{k,\ell}$ is *bad* if there exists $k' \neq k$ such that $W'_{k,\ell}$ shares an edge with a walk in a bundle $B'_{k'}$ or $B''_{k'}$. Each walk starts at an independently chosen vertex and moves to an (almost) independently chosen destination. The steady state of a random walk is uniform on edges and so at each stage of a walk, each edge is equally likely to be crossed. Thus

$$\mathbf{Pr}(W'_{k,\ell} \text{ is bad}) \le \frac{2\lambda m\theta^2(\log n)^2}{\beta_0 rn} = O\left(\frac{1}{\log n}\right).$$

We say that index k is bad if either B'_k or B''_k contain more than $\lambda/3$ bad walks. If index k is not bad, then we can find a walk from $a^*_{k,\ell}$ to $b^*_{k,\ell}$ through $x^*_{k,\ell}$ for some ℓ which is edge-disjoint from all other walks. This gives a walk

$$a_k - \tilde{a}_k - \hat{a}_k - \hat{a}_{k,\ell} - a^*_{k,\ell} - x^*_{k,\ell} - b^*_{k,\ell} - \hat{b}_{k,\ell} - \hat{b}_k - \tilde{b}_k - b_k,$$

which is edge-disjoint from all other such walks.

The probability that index k is bad is at most

$$2\,\mathbf{Pr}(B(\lambda, O(1/(\log n)) \ge \lambda/3) = O(n^{-2}).$$

Therefore with probability 1-o(1) there are no bad indices. □

Acknowledgment. We thank Aravind Srinivasan for a careful reading of an earlier version of the paper.

REFERENCES

[1] J. A. Bondy and U. S. R. Murty, *Graph Theory with Applications*, American Elsevier, New York, 1976.

[2] A. Z. Broder, A. M. Frieze, and E. Upfal, *Existence and construction of edge-disjoint paths on expander graphs*, SIAM J. Comp., 23 (1994), pp. 976–989.

[3] A. Z. Broder, A. M. Frieze, and E. Upfal, *Existence and construction of edge low congestion paths on expander graphs*, Random Structures Algorithms, 14 (1999), pp. 87–109.

[4] I. H. Dinwoodie, *A probability inequality for the occupation probability of a reversible Markov chain*, Ann. Appl. Probab., 5 (1995), pp. 37–43.

[5] A. Frank, *Disjoint paths in rectilinear grids*, Combinatorica, 2 (1982), pp. 361–371.

[6] A. M. Frieze, *Disjoint paths in expander graphs via random walks: A short survey*, in Proceedings of Random '98, Lecture Notes in Comput. Sci. 1518 (1998), Springer, Berlin, 1998, pp. 1–14.

[7] A. M. Frieze and M. Molloy, *Splitting an expander graph*, J. Algorithms, 33 (1999), pp. 166–172.

[8] A. M. Frieze and L. Zhao, *Optimal construction of edge disjoint paths in random regular graphs*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, pp. 346–355.

[9] D. Gale, *A theorem on flows in networks*, Pacific J. Math., 7 (1957), pp. 1073–1082.

[10] D. Gillman, *A Chernoff bound for random walks on expander graphs*, SIAM J. Comput., 27 (1998), pp. 1203–1220.

[11] N. Kahale, *Large Deviation Bounds for Markov Chains*, DIMACS Technical Report, DIMACS, Rutgers University, New Brunswick, NJ, 1994.

[12] J. Kleinberg and R. Rubinfeld, *Short paths in expander graphs*, in Proceedings of the 37th Annual IEEE Symposium on Foundations of Computer Science, Burlington, VT, 1996, pp. 86–95.

[13] J. Kleinberg and E. Tardos, *Approximations for the disjoint paths problem in high diameter planar networks*, J. Comput. System Sci. 57 (1998), pp. 61–73.

[14] T. Leighton and S. Rao, *Circuit switching: A multicommodity flow based approach*, in Proceedings of a Workshop on Randomized Parallel Computing, Hawaii, 1996.

[15] T. Leighton, S. Rao, and A. Srinivasan, *Multi-commodity flow and circuit switching*, in Proceedings of the Hawaii International Conference on System Sciences, Hawaii, 1998.

[16] T. Leighton, S. Rao, and A. Srinivasan, *New algorithmic aspects of the local lemma with applications to routing and partitioning*, in Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, pp. 643–652.

[17] A. Lubotsky, R. Phillips, and P. Sarnak, *Ramanujan graphs*, Combinatorica, 8 (1988), pp. 261–277.

[18] D. Peleg and E. Upfal, *Constructing disjoint paths on expander graphs*, Combinatorica, 9 (1989), pp. 289–313.

[19] N. Robertson and P. D. Seymour, *Graph minors-*XIII*: The disjoint paths problem*, J. Combin. Theory Ser. B, 63 (1995), pp. 65–110.

[20] A. Sinclair and M. Jerrum, *Approximate counting, uniform generation, and rapidly mixing Markov chains*, Inform. Comput., 82 (1989), pp. 93–133.

[21] D. Wagner and K. Weihe, *A linear time algorithm for edge-disjoint paths in planar graphs*, in Proceedings of the First European Symposium on Algorithms (ESA '93), Lecture Notes in Comput. Sci. 726, Springer-Verlag, Berlin, 1992, pp. 384–395.

[22] X. Zhou, S. Tamura, and T. Nishizeki, *Finding edge-disjoint paths in partial k-trees*, Algorithmica, 26 (2000), pp. 3–30.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1802–1828

EFFICIENT DYNAMIC TRAITOR TRACING*

OMER BERKMAN†, MICHAL PARNAS†, AND JIŘÍ SGALL‡

Abstract. The notion of traitor tracing was introduced by Chor, Fiat, and Naor [*Tracing Traitors*, Lecture Notes in Comput. Sci. 839, 1994, pp. 257–270] in order to combat piracy scenarios. Recently, Fiat and Tassa [*Tracing Traitors*, Lecture Notes in Comput. Sci. 1666, 1999, pp. 354–371] proposed a dynamic traitor tracing scenario, in which the algorithm adapts dynamically according to the responses of the pirate. Let n be the number of users and p the number of traitors.

Our main result is an algorithm which locates p traitors, even if p is unknown, using a watermarking alphabet of size $p+1$ and an optimal number of $\Theta(p^2 + p \log n)$ rounds. This improves the exponential number of rounds achieved by Fiat and Tassa in this case. We also present two algorithms that use a larger alphabet: for an alphabet of size $p+c+1$, $c \geq 1$, an algorithm that uses $O(p^2/c + p \log n)$ rounds; for an alphabet of size $pc+1$, an algorithm that uses $O(p \log_c n)$ rounds.

Our final result is a lower bound of $\Omega(p^2/c + p \log_{c+1} n)$ rounds for any algorithm that uses an alphabet of size $p+c$, assuming that p is not known in advance.

Key words. analysis of algorithms, asymptotic complexity, cryptography, traitor tracing

AMS subject classifications. 68Q25, 94A60

PII. S0097539700367984

1. Introduction.

In the electronic world, where information is easily copied and retransmitted, the issue of protecting intellectual property becomes a great concern. While owners of such information are interested in selling it, they need to protect themselves. There are two ways to protect such property. One option is to prevent the redistribution. Another option is to devise means to detect misconduct once redistribution has occurred. The second issue is the one addressed by traitor tracing (of course, the existence of such a scheme may also deter piracy from even occurring).

An excellent example is the world of Pay-TV systems. The owners of a Pay-TV system would like to broadcast only to their paying customers, and to be able to add and delete viewers as needed, while ensuring that this process happens in a timely manner. In order to protect the material that is being broadcast, encryption is utilized and keys are changed periodically, in addition to the existence of secure hardware. The goal of the pirate is to enable nonpaying persons to view the broadcasts. The pirate, registered as a legitimate subscriber of the system, can achieve this goal by transferring the decryption keys or by rebroadcasting the entire content. In order to make tracing difficult, the pirate can control several subscriptions and alternate between them. We refer to each such subscription as a *traitor*.

We assume from now on that the pirate operates by rebroadcasting the content. Our algorithms can be easily adapted to deal with the other mode of piracy, where

*Received by the editors February 23, 2000; accepted for publication (in revised form) September 15, 2000; published electronically February 23, 2001. A preliminary version of this paper appeared in *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, San Francisco, 2000, pp. 586–595.

http://www.siam.org/journals/sicomp/30-6/36798.html

†The Academic College of Tel-Aviv-Yaffo, 4 Antokolsky st., Tel-Aviv 64044, Israel (omer@mta.ac.il, michalp@mta.ac.il).

‡Mathematical Institute, AS CR, Žitná 25, CZ-11567 Praha 1, Czech Republic, and the Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University, Praha (sgall@math.cas.cz, http://www.math.cas.cz/~ sgall). This author was partially supported by grant A1019901 of GA AV ČR and grant 201/97/P038 of GA ČR.

keys are published.

Traitor tracing schemes should be capable of detecting the traitors, so that they can be disconnected from the system without harming any legitimate user. Furthermore, such schemes should supply legal evidence of the pirate's identity.

Chor, Fiat, and Naor [3] introduced the notion of traitor tracing and provided a solution to this problem. They use a static approach, which means that all security measures are applied once at the onset of the protocol. As there is a single opportunity for applying these measures, they must suffice in order to locate the traitors once piracy is observed. If the number of traitors in reality is larger than that assumed by the static scheme, then the algorithm has no way of tracing the traitors, and the traitors may even frame an innocent user. Other related work can be found in [2, 5, 6, 9, 8, 10, 11]. For a comprehensive overview of static schemes and related work, see [7].

A basic tool used by many traitor tracing schemes is cryptographic fingerprinting (e.g., [13, 1, 2, 7]), which allows the schemes to generate different versions of the same content and send them to subsets of users. This can be done, for example, by watermarking each version, with no noticeable degradation in quality. A basic assumption of both previous schemes and our schemes is that the pirate cannot remove the watermarks or combine different versions into a new one; see, e.g., [4].

In a typical static scheme that uses fingerprinting, the content is divided into segments, and each segment is watermarked. The number of different versions that can be generated from a segment is referred to as the *watermarking alphabet size*. It is possible, of course, to use an alphabet whose size equals the number of subscribers and locate the traitors immediately. However, this requires an enormous bandwidth. Therefore, a smaller alphabet is used, so that the same version of a given segment is sent to many subscribers.

1.1. Dynamic traitor tracing schemes. Recently, Fiat and Tassa [7] introduced *dynamic* traitor tracing schemes. Their schemes adapt themselves throughout the algorithm in order to force the pirate to reveal more and more information. Eventually, the algorithm locates all traitors (or stops piracy). An additional nice feature of the dynamic schemes of [7] is that there is no need for an a priori bound on the possible number of traitors, as the algorithms adapt themselves when a new traitor is discovered.

The static schemes described above are modified by [7] to fit the dynamic setting as follows: In each round, the algorithm divides the set of users into disjoint subsets, where the number of subsets is bounded by the watermarking alphabet size. Then different versions of the content are transmitted by the system to these subsets of users, one version per subset. Whenever the pirate broadcasts one of these versions, it is evidence that the corresponding subset contains a traitor. When this happens, the algorithm changes the allocation of versions to the users, thus starting a new *round*. (Otherwise the pirate could continue to broadcast the same version with no further knowledge gained by the algorithm.) Eventually, if the pirate keeps rebroadcasting, the information gathered allows the algorithm to locate and disconnect all traitors.

A *segment* in the dynamic setting is defined as the part of the content transmitted to the users within the duration of a round. We note that sending a distinct version securely to each subset can be done by encryption. See section 2.1 in [7] for a discussion of the bandwidth overhead required for sending the encrypted versions.

We are interested in the following two complexity measures of such a dynamic traitor tracing scheme:

- the *number* of different versions used (i.e., the size of the watermarking alphabet), which is proportional to the bandwidth requirements, and
- the *time* or *number of rounds* needed, which is proportional to the number of segments used.

We ignore the computation performed by the traitor tracing algorithm, but we note that all proposed algorithms, both previous and ours, are efficient.

Let n be the number of all subscribers and p the number of traitors. It is shown in [7] that

(A) any deterministic algorithm must use at least $p+1$ versions in order to locate even a single traitor, regardless of the number of rounds, and even if p is known in advance;
(B) using $2p+1$ versions, it is possible to locate all p traitors in $O(p \log n)$ rounds;
(C) using $p+1$ versions, it is possible to locate all p traitors in $O(3^p p \log n)$ rounds.

1.2. Our results. We present a family of deterministic algorithms which locate all p traitors in a polynomial number of rounds. The exact bounds depend on the size of the watermarking alphabet. Specifically, we show the following:

(1) Using $p+1$ versions, it is possible to locate all p traitors in an optimal number of $\Theta(p^2 + p \log n)$ rounds (see lower bound (4) below). This improves the exponential bound in (C), thus solving an open problem raised in [7]. Note that the algorithm in (B) runs in a polynomial number of rounds but uses twice as many versions as our algorithm. By (A), the minimal number of versions needed is $p+1$. Thus our algorithm achieves the optimal time for the minimal alphabet size.
(2) Using $p+c+1$ versions, for any $1 \le c \le p$, it is possible to locate p traitors in $O(p^2/c + p \log n)$ rounds. For example, if $c = \Omega(p)$, the time is $O(p \log n)$.
(3) Using $pc+1$ versions, for any $c \ge 2$, it is possible to locate p traitors in $O(p \log_c n)$ rounds. For example, if $c = n^\epsilon$ for a constant $\epsilon > 0$, the time is $O(p)$.

All our algorithms, similar to algorithms (B) and (C) of [7], apply even if the number of traitors p is not known in advance. This means that, for example, if the algorithm is allowed to use $p+1$ versions, then the algorithm uses only $t+1$ versions when only $t \le p$ traitors are known to exist. As noted before, the computation performed by the algorithms between consecutive rounds is efficient.

In addition, we provide the following lower bound for the time needed to locate p traitors:

(4) The time needed to locate all p traitors, when p is not known in advance and $p+c$ versions are used, is $\Omega(p^2/c + p \log_{c+1} n)$.

This bound is tight when c is a constant (result (2) above) and when $c \ge p^{1+\varepsilon}$ for a constant $\varepsilon > 0$ (result (3) above). Otherwise the gap is very small—only a factor of $\log c$. It remains an open problem to close this gap.

Organization. The remainder of this paper is organized as follows. In section 2 we describe the problem and the general strategy that will be used by our algorithms. In section 3 we describe an algorithm that uses $p+1$ versions and $O(p^3 \log n)$ rounds. This algorithm is used as a building block by later algorithms. The optimal algorithm that uses $p+1$ versions is presented in section 4, thus establishing (1). In section 5 we present two algorithms that use more than the minimal number of versions required and prove bounds (2) and (3). In section 6 we present another algorithm that uses $p+1$ versions and runs in time $O(p^3 \log n)$. While this is not optimal, it is interesting that the same bound as that given in section 3 can be obtained using a different

approach. Finally, in section 7 we prove lower bound (4). We conclude with a few open problems in section 8.

2. Preliminaries.

2.1. The model. We now describe a combinatorial game which is an abstraction of the dynamic scenario described in the introduction and serves as a model for the rest of the paper.

Denote by U the set of users, and let $n = |U|$. Let $T \subseteq U$ be the set of traitors controlled by the pirate and let $p = |T|$.

In each *round*, the algorithm assigns to each user a version of the current segment from a set of versions C. If we give version $c \in C$ to a subset of users, then the subset is said to be *colored* by c. Then the pirate broadcasts one of the colors as an *answer*. If the pirate broadcasts color c, this implies that one of the users colored by c is a traitor. A traitor is *located* if only a single user is colored by c, and c is given as an answer. Since the number of traitors is not known in advance, this is the only way to locate a traitor. A traitor that has been located is *disconnected*, i.e., removed from U.

We assume that the pirate always gives an answer, as long as he has active traitors. The pirate loses the game when he does not answer (meaning that he goes out of business because the algorithm located all traitors). We measure the maximal number of colors used in any single round and the number of rounds needed in the worst case. The goal of the algorithm is to minimize these numbers in the worst case; more precisely, to achieve some given tradeoff between the two measures.

In fact, the pirate does not have to decide in advance who the p traitors are (as we implicitly assumed above). Instead, the pirate can choose the traitors in an adaptive manner throughout the game. However, once he decides to corrupt a user, he is committed to this decision until the end of the game. (This is similar to the general situation in the area of on-line algorithms and other adversary arguments. Since the algorithms are deterministic, the adversary can simulate the algorithm in advance, and thus committing to the set T in advance does not pose any restriction.)

2.2. A simplified model. The following lemma shows that locating a single traitor is as hard as locating all p traitors in the worst case. Specifically, we show that this seemingly easier problem of locating only one traitor saves at most $p-1$ rounds. This may be somewhat counterintuitive at first, and so we try to give some intuition.

Let us call a pirate's strategy *reasonable* if it never answers by a color given to a single user, unless he has no choice (when each one of the traitors received a unique color given only to him). For reasonable strategies, the statement holds trivially, since the algorithm can locate a traitor only when all traitors have a unique color. Then it is easy to locate the remaining traitors in additional $p-1$ rounds, as claimed.

It remains to prove that reasonable strategies result in the maximal number of rounds for a given fixed number of colors. If the goal of the algorithm is to locate one traitor, this is trivial, since it brings no advantage for the pirate to lose the game before he must. However, when the goal of the algorithm is to locate all p traitors, then it is not clear in advance that a reasonable strategy is the pirate's best resort. It may be better in this case for the pirate to reveal one traitor and keep a bigger uncertainty about the location of the remaining traitors. We thus have to prove that the statement is true for any strategy, not necessarily reasonable strategies.

LEMMA 2.1. *There exists an algorithm that locates one of the p traitors in m rounds if and only if there exists an algorithm that locates all p traitors in $m+p-1$*

rounds. The bound on the number of colors is the same for both algorithms.

Proof. If we have an algorithm that locates all traitors, it must eventually get p answers that allow it to locate all traitors. If we stop after the first traitor is located, we have an algorithm that locates one traitor, takes at most m rounds instead of $m + p - 1$, and does not use more colors.

Now suppose that we have an algorithm A that locates a single traitor in at most m rounds. We devise an algorithm B that uses algorithm A as a black box in order to locate all traitors. To do this, algorithm B runs algorithm A and passes to algorithm A the pirate's answers with the following exception: whenever a color c that was given by algorithm A to a single user is given as an answer by the pirate, algorithm B disconnects that user (which is clearly a traitor), but *does not* pass the pirate's answer to algorithm A. Thus, as far as algorithm A is concerned, it still waits for an answer from the pirate, and the user that was colored by c is still a part of the user set U. However, since this user has been disconnected, the pirate cannot give the color c that was given only to this user as an answer in the next round.

As far as algorithm B is concerned a round has occurred, and we charge this round in which a traitor was located to algorithm B. All other rounds are charged to algorithm A, including the final round in which a traitor is located. Algorithm B stops when all traitors are located, that is, when no more answers are received from the pirate. Clearly, the number of colors used by algorithm B is the same as the number of colors used by algorithm A.

To finish the proof we show that algorithm B performs at most $m+p-1$ rounds, assuming that algorithm A performs at most m rounds. Indeed, we charged $p - 1$ rounds to algorithm B (the rounds in which the first $p - 1$ traitors were located). Also, the sequence of answers that are passed by algorithm B to algorithm A is valid, and thus algorithm A locates a single traitor (the last traitor located) in at most m rounds. Thus the total number of rounds performed by algorithm B is at most $m + p - 1$ as claimed. □

We will assume from now on that any algorithm stops after locating one traitor. Thus, without mentioning it explicitly in the algorithm, we assume that the pirate never broadcasts a color given to a single user (otherwise the algorithm stops immediately). By Lemma 2.1 the change in the number of rounds is at most $p - 1$, which is insignificant compared to the bounds that we show. In fact, locating all p traitors requires $\Omega(p)$ rounds no matter how many colors are used.

In practice it may be desired to reduce the number of colors used after a traitor is located. It is easy to modify our algorithms to achieve this as well.

2.3. Graph notation. The algorithms that we present partition the set of all users into disjoint subsets and give all users in the same subset a common color. We represent the current state of the algorithm by an undirected graph $G = (V, E)$. Each vertex of G represents a subset of users, and each user belongs to exactly one vertex. An edge (X, Y) means that the subset $X \cup Y$ contains a traitor, i.e., in some previous round, the answer was the color of $X \cup Y$ or (less frequently) the color of some subset of $X \cup Y$. A special vertex I represents the subset of "innocent users" (i.e., the subset of users not known at the present stage to contain a traitor). A vertex is called a *singleton* if it contains exactly one user.

2.4. A basic algorithm. Two of the schemes presented in [7] are based on the following basic algorithm, which is formulated here using our graph notation. In this algorithm the graph consists of $2t + 1$ vertices, one of which is the special vertex I, and t disjoint edges, none of which is adjacent to I. Recall that each edge implies

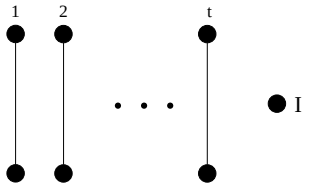


FIG. 2.1. *An example of the graph G used by the basic algorithm.*

that at least one of its endpoints contains a traitor. Since the graph has t disjoint edges, there exist at least t traitors. See Figure 2.1.

BASIC ALGORITHM. Start with a graph $G = (V, E)$ with $I = U$, $V = \{I\}$, $E = \emptyset$, and $t = 0$. Repeat forever:

(1) Find a vertex X that contains a traitor.
(2) If $X = I$: Split I into two new vertices of equal size (or differing in size by 1 if $|X|$ is odd), and connect them by an edge. Set $I = \emptyset$ and $t = t + 1$.
(3) Otherwise: Let Y be the vertex that is connected to X by an edge. Set $I = I \cup Y$, split X into two vertices of (almost) equal size, and connect them by an edge.

The first algorithm in [7] uses $2t + 1$ colors, giving each vertex a distinct color. Thus, in each round the pirate has to broadcast the color of some vertex X, and all p traitors are located in $O(p \log n)$ rounds. The second algorithm in [7] uses only $t + 1$ colors, but step (1) requires an exponential time in t, and thus the algorithm is impractical.

2.5. Generalized graph notation. In order to decrease the number of colors needed, our algorithms use graphs that generalize the above structure. Instead of the t pairs of connected vertices, we have disjoint cliques and possibly some additional edges. More formally, we define the following graphs.

DEFINITION 2.2. *Let $t \geq k \geq 0$ be two integers. A graph $G = (V, E)$ is a* (t, k)-graph *if*

(1) *G contains $t+k+1$ vertices, one of which is the special vertex I. The vertices are subsets of U, all of them except possibly I are nonempty, and every user belongs to exactly one vertex;*
(2) *for any edge $(X, Y) \in E$, the subset $X \cup Y$ contains a traitor;*
(3) *the vertices in $V \setminus \{I\}$ are partitioned into k disjoint cliques $\{Q_1, \ldots, Q_k\}$, where $|Q_i| = t_i \geq 2$ for each $1 \leq i \leq k$.*

Note that a (t, k)-graph may contain additional edges between the cliques. Further constraints on the structure of the graph will be described in each one of the algorithms. The following lemma is easy to verify if we remember that two vertices are connected by an edge if at least one of them contains a traitor.

LEMMA 2.3. *Let G be a (t, k)-graph. A clique Q_i of t_i vertices contains at least $t_i - 1$ traitors, and the graph G contains at least t traitors. The number of vertices of a (t, k)-graph is at most $2t + 1$.*

3. The clique algorithm. In this section we describe an algorithm that uses $p+1$ colors. The algorithm runs in $O(p^3 \log n)$ rounds and is not optimal. In the next section this algorithm is used to derive an algorithm that uses $(p+1)$ colors and runs in an optimal number of $\Theta(p^2 + p \log n)$ rounds.

We start with an overview of the clique algorithm. By Lemma 2.3, a (t, k)-graph is known to contain at least t traitors. Therefore we are allowed to use $t + 1$ colors.

Since a (t,k)-graph can contain up to $2t+1$ vertices, we may not have enough colors to give each vertex a distinct color. Thus our algorithm pairs the k cliques and forces the pirate in each round to either transmit a color given to only one vertex or to disclose an edge connecting a pair of cliques. If a color given to only one vertex is transmitted, then we split this vertex into two vertices (producing a new clique of size 2), and thus advance towards locating at least one of the traitors. Disclosed edges, on the other hand, are added to the graph in order to merge eventually each pair of cliques into one larger clique (containing all but one of the vertices of the two cliques). This process is repeated until the graph contains exactly one clique of size $t+1$. Then we can give each one of the vertices a distinct color and force the pirate to transmit a color given to only one vertex (allowing us again to split this vertex).

3.1. Data structures and invariants.

The graph. We maintain a (t,k)-graph $G=(V,E)$ with a special vertex I and cliques Q_i of size t_i.

Zones and blocks. The vertices of the graph G are partitioned into Zones Z_1, Z_2. The vertices in Zone Z_1 are partitioned into blocks. There are no edges connecting vertices from different blocks or zones. The zones and the blocks are defined as follows:

- ($\mathbf{Z_1}$) The vertices in this zone are partitioned into blocks. Each block is a graph induced by the vertices of two cliques Q_i and Q_j, of sizes t_i and t_j, respectively. The block does not contain a clique of size t_i+t_j-1. Equivalently, this means that there are four distinct vertices $X_1, X_2 \in Q_i$ and $Y_1, Y_2 \in Q_j$ such that $(X_1,Y_1),(X_2,Y_2) \notin E$.
- ($\mathbf{Z_2}$) This zone contains the special vertex I. In addition it may contain one clique Q_i of size t_i. The vertex I and the clique Q_i do not form a clique of size t_i+1. Equivalently, this means that there exists a vertex $X \in Q_i$ such that $(X,I) \notin E$.

See Figure 3.1 for an example of the data structures used by the clique algorithm.

3.2. Description of the algorithm. Start with a $(0,0)$-graph $G=(V,E)$ with $I=U$, $V=\{I\}$, $E=\emptyset$, and $t=0$. Repeat the following two phases:

Phase 1: Distributing the colors to the vertices of G.

Allocate t_i-1 colors to each clique Q_i, for a total of t colors. The last color is allocated to I. We now describe how to distribute these allocated colors in each zone and block of G.

(1) Color each block in Zone Z_1 separately, using the colors allocated to the pair of cliques in it: Let B be a block in Zone Z_1 and let Q_i and Q_j be the two cliques in B. By the definition of the blocks in Zone Z_1, there exist four distinct vertices $X_1, X_2 \in Q_i$ and $Y_1, Y_2 \in Q_j$ such that $(X_1,Y_1),(X_2,Y_2) \notin E$. Color $X_1 \cup Y_1$ with one color and $X_2 \cup Y_2$ with another color. Color the remaining t_i+t_j-4 vertices of Q_i, Q_j using the remaining t_i+t_j-4 colors allocated to these two cliques, one color per vertex.

(2) If Zone Z_2 contains only the vertex I, then color I using the color allocated to it. Otherwise Zone Z_2 contains also a clique Q_i and there exists a vertex $X \in Q_i$ such that $(X,I) \notin E$. Use one color for $X \cup I$ and color the remaining t_i-1 vertices of Q_i using t_i-1 colors, one color per vertex. (If I is empty, all vertices of Q_i are colored with distinct colors.) Therefore t_i colors are used, which is the total number of colors allocated to Q_i and I.

See Figure 3.1 for an example of the distribution of colors.

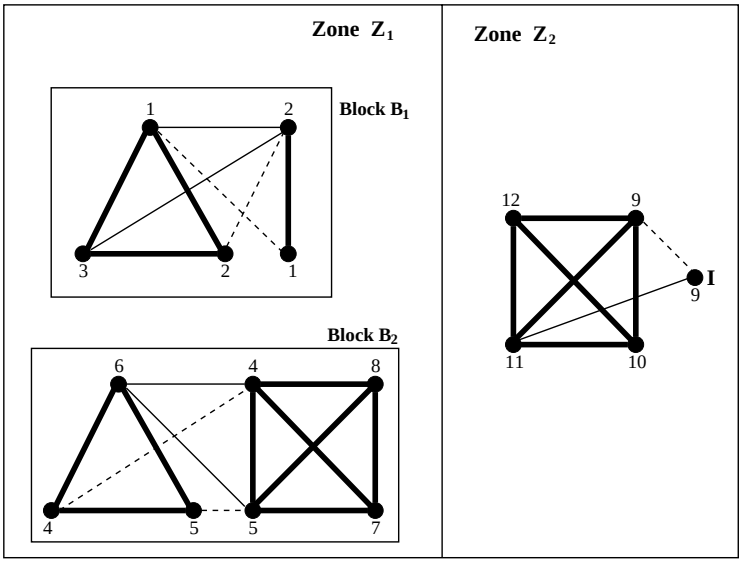


FIG. 3.1. *An example of the data structures used by the clique algorithm. This is a* (t, k)*-graph, where* $t = 11$ *and* $k = 5$*. The bold lines represent clique edges, and the thin lines represent edges between cliques, or between a clique and* I*. This is also an example of the distribution of colors in Phase* 1 *of the clique algorithm. The labels denote the colors given to the vertices. The dashed lines denote the pairs of nonadjacent vertices that received the same color.*

Phase 2: Reorganizing the graph following the pirate's response.

(1) If the pirate broadcasts a color given to only one vertex X: Remove X and the edges incident with it from G. Split X into two new vertices X_1, X_2 of equal size (or differing in size by one if $|X|$ is odd), and add the edge (X_1, X_2).
  (1.1) If $X = I$: Define a new clique Q consisting of the two vertices X_1, X_2. Set $I = \emptyset$, $t = t + 1$, and $k = k + 1$. Incorporate the clique Q into the zones (see step (3)).
  (1.2) Otherwise: Let Q_i be the clique to which X belongs. Set $Q_i = Q_i \setminus \{X\}$.
    (1.2.1) If $|Q_i| = 1$, then remove all edges incident with the vertex remaining in Q_i. Remove this vertex from Q_i and add it to I. Place X_1 and X_2 into Q_i. The clique Q_i now consists of the two vertices X_1, X_2.
    (1.2.2) Otherwise Q_i is still a legal clique. In this case the two sets X_1, X_2 will form a new clique Q. Set $k = k + 1$ and incorporate the new clique Q into the zones (see step (3)).

(2) If the pirate broadcasts a color given to a pair of vertices X, Y: Add the edge (X, Y) to G, and
  (2.1) if this edge connects vertices belonging to a pair of cliques Q_i, Q_j in some block B, and after adding this edge, the block B contains a large clique Q of size $t_i + t_j - 1$: Let Z be the remaining vertex in B that does not belong to this large clique (i.e., $Z \notin Q$). Remove Z and all edges incident with it from its clique and add Z to I. Furthermore, set $k = k - 1$, remove the cliques Q_i, Q_j and their block B. Incorporate the new clique Q into the zones (see step (3)). See Figure 3.2 for an example.
  (2.2) If this edge connects I and a vertex of the clique Q_i in Zone Z_2, and

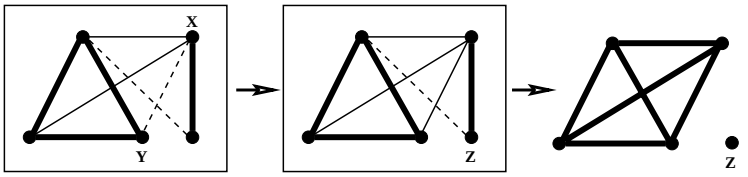


FIG. 3.2. *An example of merging a pair of cliques in step* (2.1) *of Phase* 2 *of the clique algorithm. On the left: A block of Zone* Z_1*, in which* X *and* Y *received the same color. In the middle: The block after the pirate answered with the joint color of* X *and* Y*, and the edge* (X, Y) *was added. Finally, on the right: The two cliques were merged into one big clique, and all edges incident with* Z *were removed.*

after adding this edge, Q_i and I form a clique on $t_i + 1$ vertices: Add I as a vertex to Q_i and create a new special vertex $I = \emptyset$. Set $t = t + 1$ (since t_i was increased by 1).

(3) If a new clique Q was created during one of the above steps, reorganize the zones as follows to incorporate Q: If Zone Z_2 contains only I, place Q in Zone Z_2. Otherwise, Zone Z_2 contains a clique R in addition to I. In this case, remove all edges incident with I, remove R from Zone Z_2, pair it with the new clique Q, and create in Zone Z_1 a new block containing the cliques Q and R.

Remark. In step (3) of Phase 2, we removed all edges incident with the vertex I. Removing these edges makes the structure of G simpler, but in practice we may want to leave them in the graph, since they provide more information. Since the number of these edges is $O(p)$, it turns out that the analysis of the algorithm (Theorem 3.3 below) is valid whether these edges are removed or not.

3.3. Correctness and efficiency.

LEMMA 3.1. *The invariants of the algorithm are preserved at all times.*

Proof.

The graph. An edge is added only if we got an answer from the pirate corresponding to that edge, and thus one of its endpoints must contain a traitor. Whenever the structure of the cliques is changed in steps (1) or (2) of Phase 2, we verify that every clique has at least two vertices and that the values of t and k are updated properly, so that the new graph is a (t, k)-graph for the new values. Step (3) of Phase 2 only removes edges incident with I, which maintains the structure of a (t, k)-graph.

Zone constraints. If adding a new edge would violate the zone invariants, i.e., it would create a big clique, then we reorganize the cliques in step (2) of Phase 2. In steps (1) and (3) of Phase 2, no edges are added between cliques. Reorganizing the cliques does not violate the zone constraints. If two cliques are placed in the same block, then there are no edges between them (since they were in different blocks or zones before). Similarly if a clique is placed in Zone Z_2, then there are no edges between this clique and I.

Edges between different zones and blocks. Whenever the cliques are reorganized or a new clique is created, all the edges that would connect vertices from different zones or blocks are removed. □

In order to bound the number of rounds used by the algorithm, we first need to bound the number of split operations performed throughout the algorithm. Define a *split* to be any round in which a vertex of G is split (i.e., a round in which step (1) in Phase 2 is performed).

LEMMA 3.2. *The total number of splits is* $O(p^2 \log n)$.

Proof. Whenever the set I is split, t increases. During the algorithm t never decreases and its value is at most p. Thus the set I can be split at most p times.

For every traitor i, define x_i as follows. If traitor i is in I, then $x_i = 0$. Otherwise traitor i is in a vertex $X \neq I$ and $x_i = \lceil \log |X| \rceil$. Each x_i is an integer between 0 and $\lceil \log n \rceil$. Between any two consecutive splits of I, x_i does not increase. It may be set to zero (if the corresponding vertex is added to I), decrease by one or two (if the corresponding vertex is split), or remain the same (otherwise). At each split of a vertex $X \neq I$, we know that X contains a traitor. It follows that when a spilt occurs, then for some traitor i, x_i is decreased by at least one. Thus there are at most $p\lceil \log n \rceil$ splits between any two consecutive splits of I, and the total number of splits is $O(p^2 \log n)$. □

THEOREM 3.3. *The clique algorithm locates a traitor in* $O(p^3 \log n)$ *rounds, using* $p + 1$ *colors. Hence, all* p *traitors can be located in* $O(p^3 \log n)$ *rounds, using* $p + 1$ *colors.*

Proof. By Lemma 3.1, the algorithm maintains a (t, k)-graph, and thus Lemma 2.3 implies that $t \leq p$. It is immediate from the allocation of colors in Phase 1 that the algorithm uses $t + 1 \leq p + 1$ colors.

We now analyze the number of rounds needed. In each round some progress is made: either a set is split, or an edge is added to the graph (since we assumed that the pirate always gives an answer as long as he has active traitors). We show that the number of such events is at most $O(p^3 \log n)$, and thus the algorithm must terminate and its running time is $O(p^3 \log n)$. This also proves that the algorithm eventually finishes by locating a traitor. By Lemma 2.1, all p traitors can be located in the worst case in at most $O(p^3 \log n + (p - 1)) = O(p^3 \log n)$ rounds.

By Lemma 3.2, the total number of splits is $O(p^2 \log n)$. It remains to bound the number of edges added during the algorithm. The final graph is a (t, k)-graph for some k, where $t \leq p$. Such a graph can contain at most $2t + 1 \leq 2p + 1$ vertices and thus at most $O(p^2)$ edges. However, in some rounds edges may be removed. We show below that the number of edges removed during the algorithm is $O(p^3 \log n)$. It implies that at most $O(p^2 + p^3 \log n) = O(p^3 \log n)$ edges are added during the algorithm and the theorem follows.

Edges are removed when (i) a set is split or (ii) two cliques are merged into a larger clique. Each time such an event happens, at most $O(p)$ edges are removed (including the edges that are possibly removed in step (3) of Phase 2). Event (i) can happen at most $O(p^2 \log n)$ times by Lemma 3.2. The number of times two cliques are merged is bounded from above by the number of times the number of cliques increases, and this happens only when a vertex is split. Thus event (ii) can happen at most $O(p^2 \log n)$ times. Therefore, at most $O(p^3 \log n)$ edges are removed during the algorithm. □

It may seem at first that a more careful analysis can show that the number of splits is $O(p \log n)$, which would improve the overall running time by a factor of p. However, it is possible to demonstrate a run of the algorithm where indeed $\Theta(p^2 \log n)$ splits occur and $\Theta(p^3 \log n)$ rounds are needed.

4. An optimal algorithm. In this section we describe an algorithm that uses $p + 1$ colors and runs in an optimal number of $\Theta(p^2 + p \log n)$ rounds. We start with an overview of the algorithm.

The clique algorithm loses efficiency because of the high number of $O(p^2 \log n)$ splits, and due to the fact that whenever we split a vertex, we may remove $O(p)$ edges.

We solve these problems as follows:

The users are partitioned into two areas, Areas 1 and 2. In Area 1 the users are organized into cliques of singletons (recall that a vertex is called a singleton if it contains exactly one user). Here we allow cliques of any size and use the clique algorithm. Since all vertices are singletons, there are no split operations. It follows that the total number of rounds in Area 1 is $O(p^2)$.

Initially, all the users are placed in Area 2. The only case when users are moved from Area 2 to Area 1 is when we find a set of two users containing a traitor. Then we add them to Area 1 as a clique of two singletons. Area 2 is divided into blocks, each with a constant number of known traitors. This allows us to achieve an amortized cost of $O(1)$ rounds per split and an amortized cost of $O(\log n)$ rounds until we pin down a traitor to a set of two users. Thus, at an amortized cost of $O(\log n)$, we increase the number of traitors in Area 1 (by placing there the new clique of two singletons that we found in Area 2). The number of known traitors can increase at most p times, and thus the resulting number of rounds is $O(p^2 + p \log n)$, exactly matching our lower bound. See Figure 4.1 for a sketch of the two areas.

This grand plan has some difficulties. The main obstacle is that the clique algorithm cannot be used on the blocks of Area 2, since we are not allowed to create large cliques on nonsingleton vertices (as this may result in many splits). An additional problem is that in each block of Area 2 we cannot use more colors than the number of traitors known to exist in the block (otherwise we would violate the bound of $p+1$ colors). We now sketch briefly the solutions used in Area 2 to overcome these difficulties.

Most of Area 2 is organized into blocks containing at least three known traitors each (Zone Z_4 below). In each such block, our goal is to either (i) find a set of two users containing a traitor, or (ii) split the block into two parts, one part with three known traitors and one part with one known traitor. In case (i), we can place these two users as a clique of two singletons in Area 1. In case (ii), the number of known traitors in Area 2 has increased, while maintaining the invariant of blocks with a constant number of known traitors in this area. Since our means are very limited, it may happen that we can show that there are four traitors in the block, but we are not able to partition them into two parts as required in (ii). In such a case we continue with a block known to contain more traitors. Eventually, when the number of traitors known to exist in the block raises to seven, we are able to achieve the required partition, using the basic algorithm from section 2.4 with seven colors.

Note that even if we know that there are seven traitors in the block, we do not necessarily know their exact location among the two parts into which the block was partitioned. Hence, the total number of traitors known to exist in some particular block may decrease (this never happens in the clique algorithm).

Another major problem is that we cannot treat the two areas, Area 1 and Area 2, as two completely independent processes. If we did, we would need in each of the two areas one more color than the number of traitors known to exist in that area. This would add up to $p + 2$ colors instead of the desired $p + 1$ colors. To solve this, we distinguish between the total number of traitors known to exist and the number of traitors known to exist in each one of the areas. Using a single additional color for both areas we are able to prove the existence of an additional traitor without knowing to which area it belongs. (Technically this is done by using the concept of *marking* of vertices in the algorithm.) Once we know that an additional traitor exists, we can safely use one extra color and continue the algorithm with one additional color in each

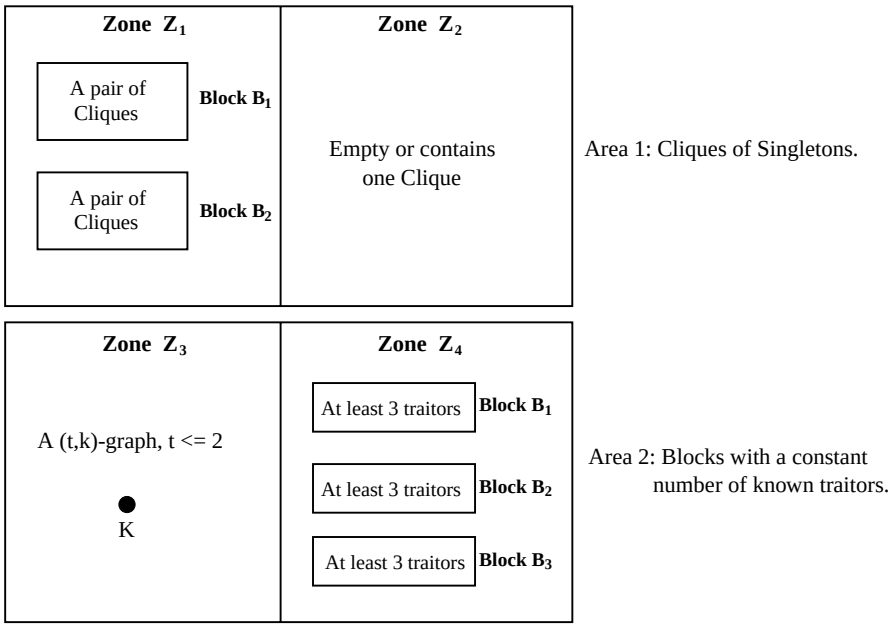


FIG. 4.1. *An example of the data structures and the graph used by the optimal algorithm.*

area.

4.1. Data structures and invariants.

The graph. We maintain an undirected graph $G = (V, E)$. The vertices are subsets of the user set U and form a partition of U. If there is an edge $(X, Y) \in E$, then $X \cup Y$ contains a traitor.

Zones and blocks. The vertices of the graph G are partitioned into Zones Z_1, Z_2, Z_3, Z_4. Furthermore, the vertices in Zones Z_1 and Z_4 are partitioned into blocks B_i. There are no edges connecting vertices from different zones or from different blocks. Let b_i be the number of traitors known to exist in block B_i, and z_i the number of traitors known to exist in Zone Z_i. The value of z_1 and z_4 is always the sum of all b_i's over the blocks of the corresponding zone. The zones and the blocks are defined as follows:

Area 1 (cliques of singletons).

- ($\mathbf{Z_1}$) This zone is partitioned into blocks B_i, where the number of vertices in B_i is $b_i + 2$, and all vertices are singletons. The vertices of block B_i can be partitioned into two cliques, each containing at least two vertices (therefore, the number of traitors known to exist in B_i is b_i). However, B_i does not contain a clique of size $b_i + 1$.
- ($\mathbf{Z_2}$) This zone either is empty or contains a clique of size $z_2 + 1 \geq 2$ whose vertices are all singletons.

Area 2 (blocks with a constant number of known traitors).

- ($\mathbf{Z_3}$) This zone is a (t, k)-graph, where $t \leq 2$. The number of traitors known to exist in Zone Z_3 is $z_3 = t \leq 2$. The special vertex I that contains "innocent users" is denoted here by K. Users from other zones may be added to the set K during the algorithm.

($\mathbf{Z_4}$) This zone is partitioned into blocks B_i, where the number of traitors known to exist in B_i is $3 \leq b_i \leq 7$.

See Figure 4.1 for an example of the data structures used by the optimal algorithm.

Number of traitors. Let $z = z_1 + z_2 + z_3 + z_4$. The value of z is updated automatically, according to the values of b_i and z_i in the individual blocks and zones. The value z is a lower bound on the number of traitors known to exist; however, it may increase or decrease. We can always use safely $z + 1$ colors.

We define another value T. Similar to z, the value T is a lower bound on the number of traitors known to exist. However, T never decreases and always $T \geq z$. Whenever z increases so that we would have $T < z$, we set $T = z$. (T may be larger than z, if we know that there exist additional traitors, but we do not know to which zone they belong. The practical implication is that the algorithm may use in this case $z + 2$ colors.)

Marks. The vertices in Zones Z_1 and Z_2 are all singletons, and they may be *marked*. The invariant is that either the marked user is a traitor, or that the total number of traitors is at least $T + 1$ (but we do not know which is the case). Additionally, in no clique in Zones Z_1 and Z_2 are all vertices marked. As a rule, whenever T increases, all marks are removed.

4.2. Description of the algorithm. We describe the algorithm with respect to each block and zone separately. When the pirate broadcasts a color given to the vertices in one of the blocks or zones, we advance the corresponding algorithm and do not change the coloring in the remaining zones.

We now describe in detail the algorithm with respect to each one of the zones. The initial graph is $G = (V, E)$, where $K = U$, $V = \{K\}$ and $E = \emptyset$ (hence, at the beginning all zones are empty except Zone Z_3). Set $T = z = 0$.

Zone Z_1. We run the clique algorithm. We need only z_1 colors, since the number of cliques in Zone Z_1 is even (as they are paired into blocks and there is no vertex I in Zone Z_1). Since all vertices are singletons, every step creates a new edge (unless we locate a traitor).

Eventually, two cliques in some block B_i are merged into one clique Q of size $b_i + 1$. Let Z be the remaining vertex in B_i that does not belong to this large clique Q. Remove all edges incident with Z and add Z to K in Zone Z_3. Reorganize the zones as follows: If Zone Z_2 is empty, the clique Q is placed in Zone Z_2. Otherwise we pair the clique in Zone Z_2 with Q into a new block in Zone Z_1, and Zone Z_2 becomes empty.

Zones Z_2 and Z_3. We use $z_2 + z_3 + 1$ colors for these two zones if $z = T$ and $z_2 + z_3 + 2$ colors if $z < T$.

(1) In Zone Z_3 we do the following:
  (1.1) We run the clique algorithm on Zone Z_3, until we know that it contains $z_3 = 3$ traitors. To do this in only $O(\log n)$ rounds, let c be a constant such that if there were only two traitors among the users, then the clique algorithm would run for at most $c \log n$ steps. If the clique algorithm runs for more than $c \log n$ steps without reaching $z_3 = 3$, we conclude that there are three traitors in the zone and we set $z_3 = 3$.
  (1.2) When $z_3 = 3$, all vertices in Zone Z_3 are placed as a block in Zone Z_4. Zone Z_3 is now empty, and we set $z_3 = 0$ and $K = \emptyset$.
(2) If Zone Z_2 is not empty and $z < T$: Use a separate color for each vertex of Zone Z_2.

(3) If Zone Z_2 is not empty and $z = T$: If the algorithm for Zone Z_3 uses at the current round only z_3 colors (out of the $z_3 + 1$ colors allocated to it), use a separate color for each vertex of Zone Z_2.
Otherwise, modify the algorithm for Zone Z_3 as follows. Suppose that the algorithm for Zone Z_3 would use at this round $q = z_3 + 1$ colors for coloring the subsets $X_1, \ldots, X_q$ (note that $q \leq 3$). Replace this round by the following q rounds. Fix an unmarked vertex S in Zone Z_2. In round k, $k = 1, \ldots, q$, color the subsets $X_1, \ldots, X_q$ in Zone Z_3 as before, color S with the color of X_k, and color the remaining vertices of Zone Z_2 with distinct colors. (The number of colors used in this case is $q + z_2 = z_2 + z_3 + 1$ as required.) Then
(3.1) if the pirate broadcasts the color of X_j, for some $j \neq k$: Advance the algorithm in step (1) above of Zone Z_3.
(3.2) Otherwise the pirate eventually broadcasts the q colors of all pairs $X_k \cup S$. In this case mark S. If all the vertices in Z_2 are marked, set $T = T + 1$ (and remove all marks).

Remark. Recall that the clique in Zone Z_2 may be moved to Zone Z_1, and in a later stage may be moved back to Zone Z_2 as part of a larger clique (after two cliques are merged). Therefore there may be marks on vertices of both Zones Z_1 and Z_2.

The algorithm used for the blocks in Zone Z_4 is fairly complex. The main building block used here is the following Algorithm (X). We now provide the specification of Algorithm (X) and then describe the algorithm for Zone Z_4. The description and analysis of Algorithm (X) is postponed to section 4.4.

Specification of Algorithm (X). *The input is a block containing at least three traitors. In each step the algorithm uses q colors, where q is the number of traitors currently known to exist in the block. At all times $3 \leq q \leq 7$. The algorithm is allowed to discard some users; in the context of the whole algorithm these users are added to the set K in Zone Z_3.*

After $O(\log n)$ rounds the algorithm either (i) *finds a subset of two users (in the block) that contains a traitor, or* (ii) *finds two disjoint subsets of the input block, one containing at least three traitors and one containing at least one traitor.*

Zone Z_4. We run Algorithm (X) on each of the blocks in this zone, thus using z_4 colors. When Algorithm (X) finishes in one of the blocks B_i, we have one of the following two cases.

(1) Block B_i is split into two subsets, one with at least one traitor and the second with at least three traitors: The first subset is added to Zone Z_3 (see step (3)). The second subset remains in this zone as a block of itself.
(2) We have a subset of two users in B_i that contains a traitor: We create a new clique Q of size 2, with two singleton vertices containing these two users, and reorganize the zones as follows. If Zone Z_2 is empty, we place the clique Q in Zone Z_2. Otherwise, we remove the single clique in Zone Z_2, pair it with the new clique Q, and create in Zone Z_1 a new block containing these two cliques; Zone Z_2 is now empty. The rest of block B_i is added to Zone Z_3 (see step (3)).
(3) In each of the above cases, some subset X of block B_i, known to contain at least one traitor, is added to Zone Z_3. This is done as follows: Remove the edges incident with X, divide X into two (almost) equal vertices connected by an edge, and add them as a clique of size 2 to Zone Z_3. Restructure the cliques of Zone Z_3 as necessary, in a similar way to step 3 in Phase 2 of the clique algorithm. Set $z_3 = z_3 + 1$. If $z_3 = 3$, go to step (1.2) in Zone Z_3.

4.3. Correctness and efficiency.

LEMMA 4.1. *The invariants of the algorithm are preserved, assuming that the same holds for Algorithm* (X).

Proof.

The graph, zone constraints, and edges between different zones and blocks. In Area 1 and Zone Z_3 these invariants are maintained because of the properties of the clique algorithm. In Zone Z_4 they are maintained because of the properties of Algorithm (X). One additional invariant is that $z_3 \leq 2$. If this invariant is violated, then the zones are restructured appropriately in step (1.2) of Zone Z_3. At all times the vertices in Area 1 are singletons: this is true when new vertices are created in step (2) of Zone Z_4 and maintained by the clique algorithm.

We need to check that the invariants are not violated when adding users to Zone Z_3 from other zones. If some users are added to K in Zone Z_3, then edges incident with K are still valid since the corresponding set of users can only become bigger. If a 2-clique is added to Zone Z_3 by step (3) of Zone Z_4, then the structure of Zone Z_3 is changed consistently with the clique algorithm. If $z_3 > 2$, then the zones are restructured.

The marks. The vertices are only marked in step (3.2) of Zones Z_2 and Z_3. If a vertex is marked and is not a traitor, it follows that all the $q = z_3 + 1$ sets $X_1, \ldots, X_q$ described there contain a traitor. This would imply that there are at least $z + 1$ traitors overall as required. Since this step is invoked only when $z = T$, the vertex is marked consistently with the invariant. The invariant of the marked vertices is maintained as long as T is unchanged, and as soon as T increases, all marks are removed.

It cannot happen that all vertices in some clique are marked. If this happens upon marking a new vertex in Zone Z_2, then the value of T is increased and all marks are removed. A new clique is created when two cliques in Zone Z_1 are merged. In this case the new clique contains all the vertices of one of the merged cliques, and thus they cannot be all marked.

The number of traitors. We need to check that there are always at least T traitors. The value T is changed in two cases—first, when z increases so that $z > T$. In this case the existence of T traitors is guaranteed by the invariants of the individual blocks and zones.

The second case is in step (3.2) of Zones Z_2 and Z_3 when all vertices in Zone Z_2 are marked. If some vertex in Zone Z_2 is not a traitor, then by the marking invariant there are at least $T + 1$ traitors, and the increase of T is justified. Otherwise all the vertices in Zone Z_2 are traitors, for a total of $z_2 + 1$ traitors. Thus, the total number of traitors is at least $z + 1$, and since $T = z$ in this step, the increase of T is justified as well. □

THEOREM 4.2. *The algorithm locates a traitor in* $O(p^2 + p \log n)$ *rounds, using* $p+1$ *colors. Hence, all* p *traitors can be located in* $O(p^2 + p \log n)$ *rounds, using* $p+1$ *colors.*

Proof. The number of colors used is at most $z_1 + z_2 + z_3 + z_4 + 1 = z + 1$ if $z = T$, and $z_1 + z_2 + z_3 + z_4 + 2 = z + 2$ if $z < T$. In both cases, at most $T + 1 \leq p + 1$ colors are used.

We now analyze the number of rounds needed by the algorithm. After at most $q = 3$ rounds in step (3) of Zones Z_2 and Z_3, we either mark a vertex or advance one of the other algorithms (most often the clique algorithm used in step (1) of Zone Z_3, but it may happen that one of the other algorithms changes z, in which case we

cannot continue the series of rounds in step (3) of Zones Z_2 and Z_3). In the first case we charge these rounds to marking a vertex. The second case adds at most $q-1$ steps per one step of the other algorithms, so the total number of rounds may increase by a factor of 3.

Let $s = z_1 + z_2$ be the number of traitors known to exist in Zones Z_1 and Z_2 (which contain only singleton vertices). Let $r = z_1 + z_2 + z_3 + 3b$, where b is the number of blocks in Zone Z_4. Note that r is a lower bound on the number of traitors in all zones, since the blocks of Zone Z_4 contribute only three traitors each instead of b_i (which may be as large as 7).

The values of T, s, and r are between 0 and p at all times. The values of s and T never decrease, and thus each of them increases at most p times. The value of r decreases only when a subset of two users containing a traitor is found in a block in Zone Z_4. In such a case s increases by 1 and r decreases by 2 (since b decreases by 1 and z_3 increases by 1). It follows that r may increase at most $3p$ times.

The value of r increases after each (amortized) $O(\log n)$ rounds in a block in Zones Z_4 and Z_3, not counting the rounds charged to marking of vertices in Zone Z_2. Thus the number of these rounds is $O(p \log n)$.

Each of the remaining rounds either is charged to marking a vertex or creates an edge between two cliques of singletons. We unmark vertices or remove edges when T increases or when two cliques are merged. We increase T at most p times, and each time $O(p)$ vertices are unmarked. This accounts for $O(p^2)$ rounds. A clique of singletons is never split. Therefore we get a new clique only when s increases, which happens at most p times. Thus two cliques are merged at most p times. Each time we remove at most p edges and one marked vertex. This accounts for $O(p^2)$ rounds. At the end we have $O(p)$ marked vertices and $O(p^2)$ edges. This accounts for $O(p^2)$ rounds as well. Thus the total number of rounds needed to locate one traitor is $O(p^2 + p \log n)$. Again, by Lemma 2.1, all p traitors can be located in the worst case in at most $O(p^2 + p \log n + (p-1)) = O(p^2 + p \log n)$ rounds. □

4.4. Algorithms for blocks in zone Z_4. Algorithm (X) described in this section receives as an input a block known to contain at least three traitors. Its goal is to either (i) find a subset of two users containing a traitor, or (ii) partition the block into two parts, one with three traitors and one with a single traitor. This is achieved in $O(\log n)$ rounds.

The plan is to first partition the block into a constant number of $q \geq 3$ parts, where each part contains at least one traitor. Then we either halve one of these q parts, and thus come closer to achieving goal (i), or we prove that the block contains at least $q+1$ traitors and thus advance towards goal (ii).

Since we cannot use more colors than the number of traitors known to exist in this block, the pirate can confuse our algorithm if there are in fact more traitors. We found the following conditional invariant extremely useful in achieving our aims.

DEFINITION 4.3. *A block is q-good if it is partitioned into $q+1$ sets of users $A_1, \ldots, A_q$, and J with the property that either each of the sets A_i contains one traitor or the block contains more than q traitors.*

In other words, either a q-good block contains exactly q traitors located in q disjoint sets, or there are more than q traitors, but we have no knowledge about their location. (Such a block is good in the sense that the algorithm is making progress in the desired direction.)

Thus, instead of finding an unconditional partitioning of the block into three parts, each containing a traitor, we try to find a partition which demonstrates that

the block is 3-good. However, this may not succeed, and instead we might just learn that there are four traitors. Then we try to find a 4-good block, etc. Eventually, when we are able to prove that there are at least seven traitors in the block, we can partition the block unconditionally as required in (ii), using the basic algorithm of section 2.4 with seven colors.

The main part of the algorithm is how to halve one of the sets A_i in a q-good block (and still remain with a q-good block), or to deduce that there are at least $q+1$ traitors in the block. Our algorithm is motivated by the exponential algorithm of Fiat and Tassa [7]. Their method is as follows: Split each one of the sets $A_1, \ldots, A_q$ into two (almost) equal parts. Choose one part from each one of the sets A_i, and color these q chosen parts with q unique colors. Use an additional color for all the remaining users (consisting of the parts of the sets A_i that did not receive a unique color and of the set J). Try all 2^q combinations of choosing q parts from each one of the sets A_i. If we get an answer in one of the parts that were colored by a unique color, then we have achieved the required split of the corresponding set A_i. Otherwise we can conclude that there are more than q traitors. However, this method uses $q+1$ colors, and we can only use q colors. Thus our algorithm is modified to use one common color for two of the q chosen parts of the sets A_i. After trying all of the $\binom{q}{2}2^q$ combinations and analyzing the answers, we are able to halve a set or prove that there are at least $q+1$ traitors. Since q is a constant, the number of combinations and rounds needed is also a constant.

4.4.1. Description of the algorithms. Algorithms (I) through (IV) described below are used as building blocks for Algorithm (X), which is then used by the algorithm for Zone Z_4 in section 4.2.

Algorithms (I)–(IV) and Algorithm (X) stop whenever they find *a subset of two users that contains a traitor*. In fact, this event always stops not only the current algorithm but also stops Algorithm (X) in the current block. We do not mention this in the descriptions of the algorithms explicitly. This also means that when we have an edge in this part of the graph, at least one of its endpoints is not a singleton. Similarly, recall that the algorithm stops upon locating a traitor. Therefore, if we get a vertex as an answer, it is not a singleton.

The algorithms are allowed to *discard* some users. In the context of the optimal algorithm described in section 4.2, these users are added to the set K in Zone Z_3.

The correctness of the individual algorithms follows immediately from the remarks in their description. After the description of each algorithm we argue briefly to bound its running time.

Algorithm (I). *The input is a constant $q \geq 3$ and a q-good block. Using q colors and $O(\log n)$ rounds, the algorithm proves that there are more than q traitors in this block. (The algorithm also stops upon finding a set of two users containing a traitor. It never discards any users.)*

Throughout the algorithm, we maintain a q-good block. Additionally, each set A_i is split into two disjoint sets $A_{i,0}$ and $A_{i,1}$ of equal size (more exactly, their cardinalities may differ by one and if A_i is a singleton, then one of the sets may be empty).

(1) In successive $\binom{q}{2}2^q = O(1)$ rounds, use all $\binom{q}{2}2^q$ colorings of the following form. For any $a_1, \ldots, a_q \in \{0,1\}$, and $1 \leq j < j' \leq q$: Use one color for the set $A_{j,a_j} \cup A_{j',a_{j'}}$ and $q-2$ colors for the sets A_{i,a_i}, $i \notin \{j, j'\}$, one color per set. The last color is given to the union of the remaining $q+1$ sets (including J).

(2) If the pirate ever broadcasts a color given to only one (nonsingleton) set $A_{i,a}$:

Add $A_{i,1-a}$ to J, set $A_i = A_{i,a}$, split A_i into $A_{i,0}$ and $A_{i,1}$, and go to step (1).

(3) If in the above $\binom{q}{2}2^q$ rounds the pirate broadcasts the colors of all possible 2^q sets containing J, each at least once, then the block contains more than q traitors and we are done.

(4) Otherwise we may suppose after renumbering that the pirate never broadcasts the color of $J \cup A_{1,1} \cup \cdots \cup A_{q,1}$. This means that the pirate broadcasts the color of $A_{j,0} \cup A_{j',0}$ for all $j \neq j'$. If two of the sets A_j and $A_{j'}$ are singletons, then $A_{j,0} \cup A_{j',0} \subseteq A_j \cup A_{j'}$ is a subset of cardinality at most two that contains a traitor, and we are done.

Otherwise there is at most one singleton set. Renumber the sets so that A_1 and A_2 are not singletons (using $q \geq 3$). The sets $A_{1,0}, \ldots, A_{q,0}$ form a clique of size q and therefore contain at least $q-1$ traitors. Below we find a set $A_{j,1}$ that does not contain a traitor, assuming that there are only q traitors. Given such a set $A_{j,1}$, we add it to J, set $A_j = A_{j,0}$, and go to step (1).

In order to find such a set $A_{j,1}$, we distinguish between two subcases. Let us examine the answer given by the pirate in the round when we used one color for the set $J \cup A_{1,0} \cup A_{2,1} \cup \cdots \cup A_{q,1}$ and one color for the set $A_{1,1} \cup A_{2,0}$.

(4.1) If the answer was $A_{1,1} \cup A_{2,0}$: Assuming that there are only q traitors, $A_{2,1}$ does not contain a traitor. Otherwise $A_{2,0}$ has no traitor, and thus $A_{1,1}$ contains a traitor; it follows that there are more than q traitors (together with the $q-1$ traitors in $A_{1,0} \cup \cdots \cup A_{q,0}$).

(4.2) If the answer was $J \cup A_{1,0} \cup A_{2,1} \cup \cdots \cup A_{q,1}$: Assuming that there are only q traitors, $A_{1,1}$ does not contain a traitor. Otherwise $A_{1,0}$ does not contain a traitor, and thus either J or some $A_{j,1}$, $j \neq 1$ contains a traitor; it follows that there are more than q traitors.

After $O(1)$ rounds, if the algorithm is not done, one of the nonsingleton sets is split and half of it is added to J. Thus the total number of rounds is $O(q \log n) = O(\log n)$ (since $q = O(1)$).

Algorithm (II). *The input is a constant $q \geq 3$ and a block known to contain at least q traitors. Using q colors and $O(\log n)$ rounds, the algorithm either* (i) *proves that there are more than q traitors in the block or* (ii) *finds a subset of the input block which consists of k cliques with a total of $k+q$ vertices (i.e., q of the vertices have to contain a traitor) for some odd number k. (The algorithm also stops upon finding a set of two users containing a traitor.)*

We run the clique algorithm on this block, with its own special set I. Initially, I contains all the users. Let c be a constant such that if there were only q traitors among the users, then the clique algorithm would run for at most $c \log n$ rounds. If the clique algorithm does not finish in $c \log n$ steps, we conclude that there exist more than q traitors and we are done by (i).

If the clique algorithm needs $q+1$ colors, then there are k cliques with a total of $k+q$ vertices for some k. If k is odd, we are done. Otherwise we discard the set I and continue running the clique algorithm with only q colors, since we do not need the color for I. Whenever a vertex is supposed to be added to I, it is discarded (this happens when a vertex in a clique of size two is split, or when two cliques are merged). Eventually, either two cliques are merged, or a vertex contained in a clique of size at least 3 is split. In either case we get an odd number of cliques and we are done by (ii).

The number of rounds of this algorithm is $O(\log n)$ as claimed because we explic-

itly restricted it.

Algorithm (III). *The input is a block consisting of a clique Q on five vertices. Using four colors and $O(\log n)$ rounds, the algorithm either* (i) *finds a 4-good block which is a subset of the input block or* (ii) *finds two disjoints subsets of the input block, one containing at least three traitors and one containing at least one traitor. (The algorithm also stops upon finding a set of two users containing a traitor.)*

(1) Let A be a nonsingleton vertex of Q. Remove the edges incident with A and split A into two vertices A_1 and A_2 of (almost) equal size. The block now contains the clique $Q \setminus \{A\}$ and the vertices A_1 and A_2.
(2) Find two disjoint nonedges, one containing A_1 and one containing A_2. Use two colors for these nonedges and two colors for the remaining two vertices. (A *nonedge* is a pair of vertices in this block that are not connected by an edge in the graph G.)
(3) If the pirate broadcasts a color given to one vertex: We are done by (ii), since this vertex is a subset with one traitor and the remaining four vertices contain three traitors.
(4) If the pirate broadcasts a color given to two vertices: Add that edge and repeat step (2). If the two disjoint nonedges in step (2) do not exist, one of the following cases occurs:
  (4.1) We have a 5-clique containing one of A_1 and A_2: Discard the other set A_i and go back to step (1).
  (4.2) We have a graph on the six vertices containing all edges except for those in the triangle A_1, A_2, B, for some vertex B: This is a 4-good block with $J = \emptyset$, one of the sets being $A_1 \cup A_2 \cup B$ and the other three sets being the remaining vertices. Thus we are done by (i).

After a constant number of rounds, if the algorithm does not end, it goes back to step 1 with a 5-clique in which one of the vertices is halved. Thus after $O(\log n)$ rounds we obtain a clique with two singletons and the algorithm stops.

Algorithm (IV). *The input is a block consisting of a clique Q on four vertices. Using three colors and $O(\log n)$ rounds, the algorithm finds a 3-good block which is a subset of the input block. (The algorithm also stops upon finding a set of two users containing a traitor.)*

(1) Let A be a nonsingleton vertex of Q. Remove the edges incident with A and split A into two vertices A_1 and A_2 of (almost) equal size.
(2) Find two disjoint nonedges, one containing A_1 and one containing A_2. Use two colors for these nonedges and one color for the remaining vertex.
(3) If the pirate broadcasts a color given to only one (nonsingleton) vertex B, one of the following cases occurs:
  (3.1) If the edge (A_1, A_2) does not exist: Unite back $A = A_1 \cup A_2$ and split B into (almost) equal parts B_1 and B_2 connected by an edge. The remaining three vertices form a clique, and there are no other edges. Set $A_1 = B_1$ and $A_2 = B_2$ and go to step (2).
  (3.2) Otherwise: We have a 3-good block with $J = \emptyset$, and the three sets are A, B, and the union of the remaining two vertices. Thus we are done.
(4) If the pirate broadcasts a color given to two vertices: Add that edge and repeat step (2). If the two disjoint nonedges in step (2) do not exist, one of the following cases occurs:
  (4.1) We have a 4-clique containing one of A_1 and A_2. Discard the other set A_i and go back to step (1).

(4.2) We have a graph on the five vertices containing all edges except for those in the triangle A_1, A_2, B, for some vertex B. This is a 3-good block with $J = \emptyset$, one of the sets is $A_1 \cup A_2 \cup B$, and the other two sets are the remaining vertices. Thus we are done.

After a constant number of steps, if the algorithm does not end, it goes back to step 1 with a 4-clique in which one of the vertices is halved (note that step (3.1) may occur only once). Thus after $O(\log n)$ rounds we obtain a clique with two singletons and the algorithm stops.

Algorithm (X). *The input is a block containing at least three traitors. In each step the algorithm uses q colors, where q is the number of traitors currently known to exist in the block. After $O(\log n)$ rounds the algorithm finds two disjoint subsets of the input block, one containing at least three traitors and one containing at least one traitor. (The algorithm also stops upon finding a set of two users containing a traitor.)*

(1) Prove that the block contains at least four traitors (or finish): Run Algorithm (II) with $q = 3$. If it proves that there are four traitors, go to step (2). Otherwise we have k cliques with a total of $k + 3$ vertices, where $k \in \{1, 3\}$. If $k = 3$, then the block is 3-good (with $J = \emptyset$ and the three sets each being the union of one clique). If $k = 1$, we have a 4-clique: Run Algorithm (IV) to find a 3-good block. In either case we get a 3-good block. Run Algorithm (I) to prove that the block contains four traitors, and go to step (2).

(2) Prove that the block contains at least five traitors (or finish): Now we have a block with four traitors. Run Algorithm (II) with $q = 4$. If it proves that there are five traitors, go to step (3). Otherwise we have k cliques with a total of $k + 4$ vertices, where $k \in \{1, 3\}$.

(2.1) If $k = 3$, then the number of traitors in the three cliques is 1, 1, and 2. We split the block into a clique with one traitor and a subset containing the remaining two cliques with three traitors, and we are done.

(2.2) If $k = 1$, then we have a 5-clique, and we run Algorithm (III). If we get two subsets with one and three traitors, we are done. Otherwise we have a 4-good block. We run Algorithm (I) with $q = 4$ to prove that the block contains five traitors, and go to step (3).

(3) Now we have a block with five traitors. Run Algorithm (II) with $q = 5$.

(3.1) If we get cliques, we split the block into two subsets—one with at least one traitor and one with at least three traitors—and we are done. This is done as follows: If there are at least three cliques, one subset is the smallest clique and the other one is the rest. If there is a single clique, one subset is the union of two of the vertices of the clique, and the other subset contains the remaining vertices.

(3.2) If we get a block with six traitors: Run Algorithm (II) with $q = 6$. If we get cliques, we split them as in step (3.1). Otherwise we have a block with seven traitors. We run the basic algorithm described in section 2.4 (for locating p traitors with $2p+1$ colors) until it needs more than seven colors. This happens when it finds four disjoint subsets each containing a traitor (the pairs in the basic algorithm). Split the block into one of these subsets and the remaining subsets.

Algorithm (X) runs Algorithms (I)–(IV) $O(1)$ times, and each of them takes $O(\log n)$ rounds. Step (3.2) also needs $O(\log n)$ rounds. Thus the total number of rounds is $O(\log n)$.

5. Using more colors. In this section we extend our results and present two algorithms which use more than $p+1$ colors. The number of rounds needed to locate all traitors is reduced as the number of colors grows.

5.1. Using $p+c+1$ colors. We modify the optimal algorithm from section 4 to use $p+c+1$ colors for any $1 \le c \le p$. The additional colors are used for cliques of singletons with more than p/c vertices. This ensures that all cliques have at most $O(p/c)$ vertices, and therefore the number of edges removed in each round is smaller. The additional colors allow us not to use marks on the vertices in the corresponding part of the optimal algorithm. (Recall that the marks were used to decrease the number of colors from $p+2$ to $p+1$.)

5.1.1. Data structures and invariants. The basic structure is similar to that of the optimal algorithm with the following modifications.

Zones and blocks.

($\boldsymbol{Z_1, Z_2}$) As in the optimal algorithm. Additionally, any clique in these zones is required to have at most T/c vertices.

($\boldsymbol{Z_3, Z_4}$) As in the optimal algorithm.

($\boldsymbol{Z_5}$) This new zone is partitioned into blocks B_i, where each B_i contains one clique of singletons of size b_i+1, such that $T/c < b_i + 1 \le 2T/c$.

Marks. There are no marks on the vertices.

5.1.2. Description of the algorithm.

Zone $\boldsymbol{Z_1}$. As in the optimal algorithm (i.e., run the clique algorithm using z_1 colors). If a new clique with more than T/c vertices is produced by merging two cliques, add it as a new block in Z_5 (instead of incorporating it into Z_1, Z_2).

Zones $\boldsymbol{Z_2}$ and $\boldsymbol{Z_5}$. Each vertex gets a unique color. If the size of a clique in Z_5 becomes $\le T/c$ (because T increased), incorporate this clique into Z_1 and Z_2 (similarly as a new clique in the clique algorithm).

Zone $\boldsymbol{Z_3}$. Run the clique algorithm as in the optimal algorithm, using z_3+1 colors, with no modifications due to marks in Z_2.

Zone $\boldsymbol{Z_4}$. As in the optimal algorithm, using z_4 colors. If $T/c < 2$ and a new clique of two singletons is created, add it as a new block in Z_5 (instead of incorporating it into Z_1, Z_2).

5.1.3. Correctness and efficiency.

LEMMA 5.1. *The invariants of the algorithm are preserved at all times.*

Proof. Each clique added to Zone Z_5 is produced by merging two cliques, each of size at most T/c, so its size is at most $2T/c$ as required. The bounds on the size of the cliques are maintained; to verify this we also use the fact that T never decreases. All other invariants are maintained similarly as in the optimal algorithm. ∎

THEOREM 5.2. *The algorithm locates a traitor in $O(p^2/c + p\log n)$ rounds, using $p+c+1$ colors, for any $1 \le c \le p$. Hence, all p traitors can be located in $O(p^2/c + p\log n)$ rounds, using $p+c+1$ colors.*

Proof. Compared to the optimal algorithm, we need one extra color for each clique in Zones Z_2 and Z_5. Let m be the number of these cliques, where $m-1$ of these cliques belong to Zone Z_5. Thus $T \ge z_2 + z_5 \ge 1 + (m-1)T/c$. It follows that $m \le c$ and thus the number of colors used is at most $p+c+1$.

The analysis of the number of rounds is the same as in Theorem 4.2, with the following differences in the area of singletons. First, we have no marks, so we do not charge any work to marking and unmarking vertices. Second, the degree of any

vertex in Zones Z_1 and Z_2 is $O(p/c)$. Thus, the number of edges removed during the algorithm and the final number of edges in G is bounded by $O(p^2/c)$ instead of $O(p^2)$. Hence, the total number of rounds needed to locate a traitor is $O(p^2/c + p \log n)$, and the theorem follows by using Lemma 2.1. □

5.2. Using $pc + 1$ colors. Now we allow for the use of $tc + 1$ colors when t traitors are known to exist for any integer $c \geq 2$. The algorithm is an immediate extension of the basic algorithm described in section 2.4.

In each stage of the algorithm we keep t disjoint sets S_i, each known to contain a traitor. Furthermore, each set is partitioned into c subsets $S_{i,j}$ of almost equal sizes. In addition we have the set I of innocent users, which at the beginning contains all users.

(1) Distribute the colors as follows: For each set S_i, color the c subsets $S_{i,j}$ using c colors, one color per subset. Color I with an additional color. The total number of colors used is thus $tc + 1$.
(2) If the pirate broadcasts the color of I: Create a new set $S_{t+1} = I$ and partition it into c (almost) equal subsets. Set $I = \emptyset$ and $t = t + 1$.
(3) If the pirate broadcasts the color of a subset $S_{i,j}$: Add all subsets $S_{i,j'}$, $j' \neq j$, to I. Set $S_i = S_{i,j}$ and partition S_i into c new (almost) equal subsets.

THEOREM 5.3. *The algorithm locates a traitor in $O(p \log_c n)$ rounds, using $pc+1$ colors, for $c \geq 2$. Hence, all p traitors can be located in $O(p \log_c n)$ rounds, using $pc+1$ colors.*

Proof. A subset is split at each round of the algorithm. Since a split divides a subset into c equal parts, each subset other than I can be split at most $O(\log_c n)$ times. The subset I may be split an additional p times. Thus the algorithm finishes in $O(p \log_c n)$ rounds. □

6. The degree algorithm. In this section we present an algorithm based on a different idea. This algorithm uses $p + 1$ colors and runs in time $O(p^3 \log n)$. Thus the bounds are the same as for the clique algorithm.

The knowledge of the algorithm is represented by a similar graph as before. That is, vertices represent disjoint subsets of users from U, and an edge between vertices X and Y represents the fact that the subset $X \cup Y$ contains a traitor.

The main idea here is to eliminate vertices with a large degree. To do that, note that if the degree of a vertex is larger than p, then this vertex must contain a traitor (otherwise each of its neighbors would contain a traitor, resulting in more than p traitors). Therefore, if we knew p in advance, we would know that such a vertex contains a traitor and the algorithm could have split it.

Since p is unknown, we maintain a value T (as in the previous algorithms), which is the number of traitors known by the algorithm to exist. Whenever a vertex of a large degree appears, we *suspect* that it contains a traitor. (The notion of a large degree will be defined later.) Note that our knowledge about these vertices is conditional only: we know that either the vertex contains a traitor, or that the actual number of traitors is greater than T (however, we do not know which is the case). Consequently, when T is increased, we have no information about any traitor in these suspected vertices and we have to treat the users as innocent.

Among the remaining vertices, we distribute the colors so that each color is used by one vertex or by two vertices not connected by an edge. An existence of such a coloring is implied by the fact that the degree of these vertices is small. In each round progress is made by adding a new edge, splitting a vertex, or finding a vertex of a large degree.

6.1. Data structures and invariants.

The graph. We maintain an undirected graph $G = (V, E)$. The vertices are subsets of the user set U and form a partition of U. If there is an edge $(X, Y) \in E$, then $X \cup Y$ contains a traitor.

The zones. The vertices of the graph G are partitioned into Zones Z_1, Z_2. There are no edges adjacent to vertices in Zone Z_1. We also maintain a value T, which is the number of traitors known to exist. The zones are defined as follows:

- (**Z_1**) This zone contains z_1 vertices. The invariant is that if the actual number of traitors is exactly T, then each vertex in Z_1 contains a traitor. As stated above, the degree of all vertices in this zone is 0.
- (**Z_2**) This zone is a (z_2, z_2)-graph, i.e., a graph on $2z_2 + 1$ vertices containing z_2 disjoint cliques of size 2 each, and the special vertex I (there may be edges between the cliques). The invariant is that all vertices in this zone have degree at most $T - z_1$.

Number of traitors. Let $z = z_1 + z_2$. Thus z is the number of traitors known to exist in distinct vertices of G, assuming that the actual number of traitors is T. The invariant is that the total number of traitors is at least T and that $T \geq z$. This implies that the algorithm may use $T + 1$ colors. As a rule, T never decreases. Note that the total number of vertices in the graph is $|V| = z_1 + 2z_2 + 1 = z + z_2 + 1$.

6.2. Description of the algorithm. Before describing the algorithm we give a lemma about the existence of a large matching, and show how it is applied to find a proper coloring in the algorithm. A *matching* in a graph is a set of disjoint edges. The *size* of the matching is the number of edges in the matching. The proof of the following lemma is standard and can be found in Appendix A.

LEMMA 6.1. *Let H be a graph with at least $2d$ vertices with degree at least d each. Then there exists a matching in H of size at least d.*

Let G' be the complement graph of G restricted to Zone Z_2, i.e., the vertices of G' are the vertices in Zone Z_2, and two distinct vertices are connected by an edge in G' if and only if they are not connected by an edge in G. Zone Z_2 contains $2z_2 + 1$ vertices, each with degree at most $T - z_1$. Thus the minimum degree of the graph G' is $2z_2 - (T - z_1) = z_2 + z - T$. By taking $d = z_2 + z - T$ in Lemma 6.1, there exists a matching of size at least $z_2 + z - T$ in G'.

We now describe the algorithm. The initial graph is $G = (V, E)$ with $I = U$, $V = \{I\}$, and $E = \emptyset$. Zone Z_1 is empty and Zone Z_2 contains the single vertex I. Set $T = z = 0$. Repeat the following two phases:

Phase 1: Distributing the colors to the vertices of G.

Find a matching of size at least $z_2 + z - T$ in G' (which exists by Lemma 6.1 and the above discussion). Use one color for each edge in the matching (i.e., for the two vertices of the edge). Use a distinct color for every remaining vertex, including the vertices of Zone Z_1. The number of colors used is at most $|V| - (z_2 + z - T) = (z + z_2 + 1) - (z_2 + z - T) = T + 1$.

Phase 2: Reorganizing the graph following the pirate's response.

(1) If the pirate transmitted a color given to a single vertex X: Remove X and all edges incident with X. Split X into two (almost) equal subsets X_1 and X_2, and add the edge (X_1, X_2). Place X_1 and X_2 as a new clique in Zone Z_2.
  (1.1) If X was in Zone Z_1: Set $z_1 = z_1 - 1$ and $z_2 = z_2 + 1$.

(1.2) If X was in Zone Z_2 in a clique with Y: Remove Y and all edges incident with Y, and set $I = I \cup Y$.

(1.3) If X was in Zone Z_2 and $X = I$: Create a new special vertex $I = \emptyset$ and set $z_2 = z_2 + 1$.

(2) If the pirate transmitted a color given to a pair of vertices (X, Y) (necessarily in Zone Z_2): Add the edge (X, Y) to G, and

(2.1) for any vertex $Z \in Z_2$ of degree larger than $T - z_1$ do the following: Move Z to Zone Z_1, remove all edges incident with Z, and set $z_1 = z_1 + 1$. If $Z = I$, create a new special vertex $I = \emptyset$.

(2.2) Repeat step (2.1) until all vertices have degree at most $T - z_1$. (Note that after step (2.1) the value of $T - z_1$ decreases, which may cause some more vertices to have a large degree.)

(2.3) Add to the special vertex I all the vertices which remain in Zone Z_2 and do not belong to a clique of size 2 (i.e., the vertex I will contain all the vertices who had a neighbor Z (in their clique) that was moved to Zone Z_1. The vertex I will also contain the old I if it was not moved to Zone Z_1). Update z_2 to reflect the current number of cliques of size 2 in Zone Z_2.

(3) If after one of the previous steps $z > T$: Set $T = T + 1$ and add to I all the vertices that belong to Zone Z_1. Zone Z_2 remains unchanged.

6.3. Correctness and efficiency.

LEMMA 6.2. *The invariants of the algorithm are preserved at all times.*

Proof.

The graph. An edge is added only if we got an answer from the pirate corresponding to that edge.

Zone Z_1. First consider the moment when we decide to place a vertex X in Zone Z_1. At that point X is in Zone Z_2 and its degree is larger than $T - z_1$. Assume that the invariant is violated, i.e., X does not contain a traitor and the actual number of traitors is exactly T. It follows that there are more than $T - z_1$ traitors in the neighbor vertices of X in Zone Z_2. Also, Zone Z_1 contains z_1 traitors, by the invariant of the zone. Thus the total number of traitors is strictly larger than T, a contradiction. We conclude that the invariant is maintained when a vertex is placed in Zone Z_1.

As long as a vertex remains in Zone Z_1, the value of T does not change, by step (3) of the algorithm. Thus the invariant is maintained at all times.

Zone Z_2. Whenever one vertex of a clique is removed in step (2.1), the other vertex is also removed in step (2.3), and the value of z_2 is updated accordingly. Thus, the (z_2, z_2)-graph is maintained. No vertex in Zone Z_2 has degree larger than $T - z_1$ after a round because we remove in step (2.2) all such vertices from Zone Z_2.

The number of traitors. T is increased only in step (3). Assume to the contrary that there are only T traitors when T is increased. Then there are z_1 traitors in Zone Z_1 by its invariant and z_2 traitors in Zone Z_2. However, then the total number of traitors is at least $z_1 + z_2 = z > T$, a contradiction. We conclude that there are at least $T + 1$ traitors and the increase of T is justified. Thus, there are at least T traitors at all times.

Note that we cannot conclude that there are z traitors, since a vertex in Zone Z_1 is known to contain a traitor only if the actual number of traitors is exactly T. Since this was found to be untrue as T has increased, we cannot assume any more that the vertices in Zone Z_1 contain traitors. □

THEOREM 6.3. *The degree algorithm locates a traitor in $O(p^3 \log n)$ rounds, using at most $p+1$ colors. Hence, all p traitors can be located in $O(p^3 \log n)$ rounds, using at most $p+1$ colors.*

Proof. The algorithm always uses at most $T+1 \leq p+1$ colors, by the calculation in Phase 1 of the algorithm.

We now estimate the number of rounds. The value of T is increased at most p times and never decreases. Therefore it is sufficient to prove that for each value of T, the number of rounds is at most $O(T^2 \log n)$. Since $T \leq p$, the total number of rounds until a traitor is located is $O(p^3 \log n)$.

When T does not increase, the algorithm takes one of the following actions: either (1) a vertex is split or moved to Zone Z_1 and at most $2T$ edges are removed or (2) an edge is added to G. Note that (1) may happen more than once during a single round. Zone Z_1 has at most T vertices at the end, so the number of times a vertex is moved to Zone Z_1 is at most T plus the number of splits. Similarly, as in the previous algorithms, there are at most $O(T \log n)$ splits before T increases. Consequently, the number of steps of type (2) is bounded by the number of edges of G at the end, plus the number of edges removed during the algorithm. This gives the bound of $O(T^2 \log n)$ rounds before T increases. ∎

7. Lower bounds. We now prove a lower bound on the number of rounds needed by any algorithm that uses $p+c$ colors, assuming that p is not known in advance.

LEMMA 7.1. *Using $p+c$ colors, at least $\Omega(p^2/c)$ rounds are needed to locate p traitors, when p is not known in advance.*

Proof. If $c \geq p$, the bound is true, since at least p rounds are needed to locate p traitors. Let $c < p$ and let $n = p + c + 1$ be the number of users, i.e., there are $c+1$ innocent users. In any coloring with $p+c$ colors there are at least two users with the same color.

We use an adversary argument. In each round, the pirate transmits a color given to (at least) two users. We prove that the pirate can continue for at least $\Omega(p^2/c)$ rounds, consistently with the fact that there are p traitors. We maintain a graph G with users as vertices, initially with no edges. In each round we add an edge between the two users whose color was given as an answer (if the algorithm colors more than two users by the same color, we choose any two of them; this can only decrease the number of rounds). The pirate can continue as long as there is an independent set of size $c+1$ in G. (Recall that an independent set is a set of vertices with no edges between them.) In this case the pirate's answers are consistent with the p traitors being the vertices not in the independent set.

Let H be the graph on n vertices consisting of c disjoint cliques, each with $\lfloor n/c \rfloor$ or $\lceil n/c \rceil$ vertices, and no other vertices and edges. By Turan's theorem (see [12]) used for the complement graph of G, if G does not contain an independent set of size $c+1$, then G has at least as many edges as H. The number of edges in H is $\Theta(p^2/c)$ (the cliques are nontrivial, since $c < p$). Thus the pirate can always continue for at least $\Omega(p^2/c)$ rounds. ∎

THEOREM 7.2. *Using $p+c$ colors, at least $\Omega(p^2/c + p \log_{c+1} n)$ rounds are needed to locate p traitors if p is not known in advance and $n \geq (1+\varepsilon)c$ for some constant $\varepsilon > 0$.*

Proof. We use an adversary argument which proceeds in phases. In phase t, $t = 1, \ldots, p$, the pirate has already chosen the location of the first $t-1$ traitors and is revealing the location of the next traitor. The algorithm knows about the existence of only t traitors and thus can use $t+c$ colors. The pirate keeps a set A of candidates for

the next traitor. At the beginning of each phase, the set A contains all users except for the first $t-1$ traitors. If any of the first $t-1$ traitors does not get a unique color, then the pirate broadcasts that color. Otherwise A is colored by at most $c+1$ colors. Then the pirate broadcasts the color given to most users in A and removes all the other users from A. When the size of A is at most $c+1$, the pirate fixes one user in A as the next traitor, and a new phase begins. No traitor is located until the end of phase p.

Assume, without loss of generality, that $\varepsilon < 1$. The number of rounds in phase t is at least $\lceil \log_{c+1}(n+1-t) \rceil - 1$. For $t < \varepsilon p/2 \le \varepsilon n/2$, we have $n+1-t > c+1$ and $n+1-t > n/2$. Therefore, the number of rounds in phase t is at least $\Omega(\log_{c+1} n)$ and the total number of rounds is at least $\varepsilon p/2 \cdot \Omega(\log_{c+1} n) = \Omega(p \log_{c+1} n)$. Together with Lemma 7.1 this proves the theorem. ∎

8. Open problems. We conclude with a few open problems. First it would be nice to close the small gap of $O(\log c)$ between the lower and upper bound in the case that $p+c$ colors are used, and p is not known in advance.

We achieve an asymptotically optimal number of rounds in the optimal algorithm described in section 4. However, it is very complex and the hidden constants are large. From a more practical viewpoint, it would be desired to find a simpler optimal algorithm for the $p+1$ case, with possibly better constants.

It may be interesting to investigate randomized algorithms. The first question is what is the proper model in this case. In the deterministic case, the pirate is able to simulate the algorithm. Thus, for example, the pirate can deduce completely the distribution of the colors. In the randomized case, it is not realistic to assume this, and the game needs to be specified precisely.

Appendix A. Proof of Lemma 6.1.

LEMMA 6.1. *Let H be a graph with at least $2d$ vertices with degree at least d each. Then there exists a matching in H of size at least d.*

Proof. The proof is by induction. Given a matching of size $c < d$, we show how to find a matching of size $c+1$. Starting at $c = 0$ and repeating this process, we find a matching of size d.

Let v and w be two unmatched vertices of degree at least d each. If some neighbor of v or w is unmatched, then we extend the matching trivially. Otherwise, let $x_1, \ldots, x_d$ be d distinct neighbors of v in H, and $y_1, \ldots, y_d$ their respective neighbors in the matching of cardinality c. Some vertices y_j may be equal to some x_i, but in any case all the vertices y_j are distinct. Out of the $2c$ vertices of the matching of size c there are $2c-d$ vertices that are distinct from all the vertices y_j. Since $c < d$, we have $2c-d < d$. As all the neighbors of w are matched and its degree is at least d, it follows that w is connected to y_j for some j. Replace the edge (x_j, y_j) in the matching by the edges (v, x_j) and (y_j, w). ∎

Acknowledgments. We are in debt to Amos Fiat, Aria Lev, Tal Rabin, and Tamir Tassa for useful discussions and comments. We would especially like to thank the anonymous referees for the many suggestions and comments they made, which helped to improve the presentation of this paper.

REFERENCES

[1] G. R. BLAKLEY, C. MEADOWS, AND G. B. PURDY, *Fingerprinting long forgiving messages,* in Proceedings of Advances in Cryptology—CRYPTO'85, Lecture Notes in Comput. Sci. 218, Springer-Verlag, New York, 1985, pp. 180–189.

[2] D. Boneh and J. Shaw, *Collusion-secure fingerprinting for digital data*, IEEE Trans. Inform. Theory, 44 (1998), pp. 1897–1905.
[3] B. Chor, A. Fiat, and M. Naor, *Tracing traitors*, in Proceedings of Advances in Cryptology—CRYPTO'94, Lecture Notes in Comput. Sci. 839, Springer-Verlag, New York, 1994, pp. 257–270.
[4] I. J. Cox, J. Kilian, T. Leighton, and T. Shamoon, *A secure, robust watermark for multimedia*, in Information Hiding, Lecture Notes in Comput. Sci. 174, Springer-Verlag, New York, 1996, pp. 185–206.
[5] C. Dwork, J. Lotspiech, and M. Naor, *Digital signets: Self-enforcing protection of digital information*, in Proceedings of the 28th Symposium on the Theory of Computing, 1996, pp. 489-498.
[6] A. Fiat and M. Naor, *Broadcast encryption*, in Proceedings of Advances in Cryptology—CRYPTO'93, Lecture Notes in Comput. Sci. 773, Springer-Verlag, New York, 1993, pp. 480–491.
[7] A. Fiat and T. Tassa, *Dynamic traitor tracing,* in Proceedings of Advances in Cryptology—CRYPTO'99, Lecture Notes in Comput. Sci. 1666, Springer-Verlag, New York, 1999, pp. 354–371.
[8] M. Naor and B. Pinkas, *Threshold traitor tracing*, in Proceedings of Advances in Cryptology—CRYPTO'98, Lecture Notes in Comput. Sci. 1462, Springer-Verlag, New York, 1998, pp. 502–517.
[9] B. Pfitzmann, *Trials of traced traitors*, in Information Hiding, Lecture Notes in Comput. Sci. 1174, Springer-Verlag, New York, 1996, pp. 49–64.
[10] D. R. Stinson and R. Wei, *Combinatorial properties and constructions of traceability schemes and frameproof codes*, SIAM J. Discrete Math., 11 (1998), pp. 41–53.
[11] J. Schwenk and J. Ueberberg, *Tracing Traitors Using Finite Geometries*, Technical report, Deutsche Telekom AG, 1997.
[12] P. Turán, *On an extremal problem in graph theory*, Math. Fiz. Lapok, 48 (1941), pp. 436–452.
[13] N. R. Wagner, *Fingerprinting,* in Proceedings of the 1983 IEEE Symposium on Security and Privacy, IEEE Computer Society Press, Los Alamitos, CA, pp. 18–22.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1829–1841

QUANTUM ENTANGLEMENT AND COMMUNICATION COMPLEXITY*

HARRY BUHRMAN†, RICHARD CLEVE‡, AND WIM VAN DAM§

Abstract. We consider a variation of the communication complexity scenario, where the parties are supplied with an extra resource: particles in an entangled quantum state. We note that "quantum nonlocality" can be naturally expressed in the language of communication complexity. These are communication complexity problems where the "output" is embodied in the correlations between the outputs of the individual parties. Without entanglement, the parties must communicate to produce the required correlations; whereas, with entanglement, *no* communication is necessary to produce the correlations. In this sense, nonlocality proofs can also be viewed as communication complexity problems where the presence of quantum entanglement reduces the amount of necessary communication. We show how to transform examples of nonlocality into more traditional communication complexity problems, where the output is explicitly determined by each individual party. The resulting problems require communication with or without entanglement, but the required communication is less when entanglement is available. All these results are a noteworthy contrast to the well-known fact that entanglement cannot be used to actually simulate or compress classical communication between remote parties.

Key words. complexity theory, communication complexity, quantum computing

AMS subject classifications. 68Q15, 68Q40

PII. S0097539797324886

1. Introduction and summary of results.

One of the most remarkable aspects of quantum physics is the notion of *quantum entanglement*. If two particles are in an entangled state, then, even if the particles are physically separated by a great distance, they behave in some respects as a single entity. The entangled particles exhibit what physicists call *nonlocal* effects. Informally, these are effects that cannot occur in a world governed by the laws of "classical" physics unless communication occurs between the particles. Moreover, if the physical separation between the particles is large and the time between the observations is small, then this entailed communication may exceed the speed of light! Nonlocal effects were alluded to in a famous 1935 paper by Einstein, Podolsky, and Rosen [13]. Einstein later referred to this as *spukhafte Fernwirkungen* (spooky actions at a distance) (see [12, 25, 30] for more historical background). In 1964, Bell [3] formalized the notion of two-particle nonlocality in terms of correlations among probabilities in a scenario where one of a number of a measurements are performed on each particle. He showed that the results of the measurements that occur quantum physically can be correlated in a way that cannot occur classically unless the type of measurement selected to be per-

*Received by the editors July 24, 1997; accepted for publication (in revised form) August 3, 2000; published electronically March 13, 2001.

http://www.siam.org/journals/sicomp/30-6/32488.html

†CWI, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands (buhrman@cwi.nl). This author was supported in part by NWO, SION Project 612-34-002, EU through NeuroCOLT ESPRIT Working Group 8556, HC&M grant CCR 92-09833, and Fifth Framework Program FET project QAIP IST-1999-11234.

‡Department of Computer Science, University of Calgary, Calgary, Alberta, Canada T2N 1N4 (cleve@cpsc.ucalgary.ca). This author was supported in part by Canada's NSERC.

§CWI, and Computer Science Division, University of California, 665 Soda Hall, Berkeley, CA 94720 (vandam@eecs.berkeley.edu). This author was supported by iLLc Amsterdam, EU project QAIP IST-1999-11234, and NWO's TALENT grant S 62-552.

formed on one particle affects the result of the measurement performed on the other particle.

In reality—which is quantum physical—the nonlocal effects exhibited by entangled particles do not involve any communication (consequently, nonlocality does not entail communication faster than the speed of light). In operational terms, the "spooky actions at a distance" that Einstein referred to cannot be used to simulate a communication channel. More precisely, if two physically separated parties, Alice and Bob, initially possess entangled particles and then Alice is given an arbitrary n-bit string x, there is no way for Alice to manipulate her particles in order to convey any information about x to Bob (unless she explicitly sends that information to him). Moreover, entanglement cannot even be used to *compress* the information in x: for Alice to convey x to Bob, she must in general send n bits—any smaller number will not suffice. The proof of this is based on a fundamental theorem in quantum information theory due to Holevo [17] (see also [16, 10]). Similar results apply to communication involving more than two parties.

Now consider the *communication complexity* scenario introduced by Yao [33]. Alice obtains an n-bit string x and Bob obtains an n-bit string y, and the goal is for them to determine $f(x, y)$, for some function $f : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}$, with as little communication between them as possible. Clearly, $n + 1$ bits of communication always suffice (Alice sends all her n bits to Bob, Bob computes $f(x, y)$ and sends the one-bit answer to Alice), but for some functions fewer bits suffice. This scenario and variations of it have been widely studied (see [23] for an extensive survey).

In one variation of the above communication complexity scenario, there are more than two parties, each of which is given a subset of the input data. In another variation, all parties have access to a common "public" string of random bits. This string can be assumed to have been communicated during a "set up" stage prior to the parties being given their input data. For some functions, this prior random string reduces the communication complexity for a worst-case input if a small error probability is permitted (here, a worst-case input is understood to be chosen independently of the random string).

The first variation of the communication complexity scenario that incorporates quantum information was proposed by Yao [34]. In this model, Alice and Bob are allowed to communicate with *quantum bits (qubits)* rather than classical bits. Kremer [22] includes many important definitions and basic results for this model, including a proof that for the Inner Product function $f(x, y) = x_0 \cdot y_0 + x_1 \cdot y_1 + \cdots + x_{n-1} \cdot y_{n-1} \bmod 2$, the qubit communication must be $\Omega(n)$ qubits. These works leave open the question of whether quantum information can ever be advantageous over classical information for a communication complexity problem.

In the present paper, we consider an alternate way of incorporating quantum information into the communication complexity scenario. Here Alice and Bob's communication is with classical bits, but they are provided with a priori information that is entangled. On the face of it, it may appear that a prior quantum entanglement cannot reduce communication complexity because of the aforementioned theorem of Holevo. Consider the following informal argument, where Alice and Bob are given input strings x and y, and their goal is to collectively determine $f(x, y)$:

1. Assume that the classical communication complexity of function $f(x, y)$ is k. That is, k bits of communication are necessary for Alice and Bob to acquire the answer.

2. By Holevo's Theorem [17], the prior entanglement cannot simulate or even compress any particular message in a classical communication protocol.
3. Ergo, even with prior entanglement, the communication complexity of $f(x, y)$ is k.

A similar informal argument could be made for three-party scenarios. We shall demonstrate that this conclusion is incorrect for both scenarios.

Our first counterexample is in a three-party setting. We give an example of a function $f : \{0,1\}^2 \times \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}$, where, without prior quantum entanglement, four bits of communication are *necessary* to compute $f(x, y, z)$; whereas, with prior quantum entanglement, three bits of communication are *sufficient* to compute $f(x, y, z)$. The function is actually a partial function, defined on a subset of $\{0,1\}^2 \times \{0,1\}^2 \times \{0,1\}^2$ (i.e., the input data (x, y, z) obeys a certain "promise"). If we want to allow any input combination, then f can be defined as a relation (rather than a function). The protocol employing quantum entanglement uses less communication than necessary by any classical protocol by manipulating the entanglement so as to *circumvent* (rather than simulate) communication. Our technique is based on an interesting example of tripartite nonlocality due to Mermin [26]. Mermin's result is a refinement (from four components to three) of a similar result by Greenberger, Horne, and Zeilinger [14].

We also give an example of a two-party probabilistic communication complexity scenario with a function $g : \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}$ for which, with a classical shared random string but no prior entanglement, three bits of communication are *necessary* to compute $g(x, y)$ with probability at least $\cos^2(\frac{\pi}{8}) = 0.853\ldots$; whereas, with prior entanglement, two bits of communication are *sufficient* to compute $g(x, y)$ with the same probability. Unlike the previous three-party example, this function does not require a promise on the input data (x, y). The correlations in this two-party scenario are based on an example of nonlocality due to Clauser et al. [8].

Although, in both of the above cases, the savings in communication are not in an asymptotic setting, these results demonstrate that quantum entanglement can change the nature of communication complexity. After the initial announcement of these results and those of [9], several stronger quantum vs. classical separations appeared, and these are briefly reviewed in section 5.

2. Three-party deterministic scenarios. Let us begin by considering the following scenario, which is a reformulation of the one in [26] but cast in terms of data processing. Alice, Bob, and Carol receive input bits x, y, and z, respectively, which are arbitrary subject to the condition that $x \oplus y \oplus z = 0$. Once they receive their input data, they are forbidden from having any communication between them. Their goal is to produce output bits a, b, and c, respectively, such that

$$a \oplus b \oplus c = \begin{cases} 0 & \text{if } xyz = 000, \\ 1 & \text{if } xyz \in \{011, 101, 110\}. \end{cases} \tag{2.1}$$

Let us consider whether or not the trio can accomplish the above in terms of classical information. Since Alice cannot receive any information from Bob or Carol, her output bit a can depend only on the value of her input bit x. Let a_0 (respectively, a_1) be Alice's output when her input bit is 0 [1]. Similarly, let b_0, b_1 and c_0, c_1 be Bob and Carol's outputs for their respective input values. Note that the six bits $a_0, a_1, b_0, b_1, c_0, c_1$ completely characterize any (deterministic) strategy of Alice, Bob, and Carol (and probabilistic strategies will not help here since no error probability is

permitted). The conditions of the problem translate into the equations

$$\begin{aligned} a_0 \oplus b_0 \oplus c_0 &= 0, \\ a_0 \oplus b_1 \oplus c_1 &= 1, \\ a_1 \oplus b_0 \oplus c_1 &= 1, \\ a_1 \oplus b_1 \oplus c_0 &= 1. \end{aligned} \tag{2.2}$$

It is impossible to satisfy all four equations simultaneously. This is because summing the four equations (modulo two) yields $0 = 1$. Therefore, for any strategy, there exists an input configuration $xyz \in \{000, 011, 101, 110\}$ for which it fails.

Now consider the same problem, but where Alice, Bob, and Carol are supplied with qubits Q_A, Q_B, and Q_C, respectively, where the state of $\mathsf{Q}_A\mathsf{Q}_B\mathsf{Q}_C$ is initialized to

$$\tfrac{1}{2}(|000\rangle - |011\rangle - |101\rangle - |110\rangle). \tag{2.3}$$

The parties are allowed to apply unitary transformations and perform measurements on their individual qubits, but communication between the parties is still forbidden. It turns out that now the parties *can* produce a, b, c satisfying (2.1). This is achieved by the following procedures:

Procedure for Alice:
if $x = 1$ **then apply** H **to** Q_A
measure Q_A **yielding bit** a

Procedure for Bob:
if $y = 1$ **then apply** H **to** Q_B
measure Q_B **yielding bit** b

Procedure for Carol:
if $z = 1$ **then apply** H **to** Q_C
measure Q_C **yielding bit** c

In the above, H is the Hadamard transform, which is represented in the standard basis ($|0\rangle$ and $|1\rangle$) as

$$H = \tfrac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{2.4}$$

and the measurements are performed in the standard basis.

We claim that the described procedure produces three output bits a, b, and c, satisfying (2.1). To see why this is so, first consider the case where $xyz = 000$. In this case, no H transform is applied to any of the three qubits. Therefore $\mathsf{Q}_A\mathsf{Q}_B\mathsf{Q}_C$ is measured directly in state (2.3), so the results will satisfy $a \oplus b \oplus c = 0$.

Next, in the case where $xyz = 011$, a Hadamard transform is applied to Q_B and to Q_C but not to Q_A. Therefore $\mathsf{Q}_A\mathsf{Q}_B\mathsf{Q}_C$ is measured in state

$$I \otimes H \otimes H \left(\tfrac{1}{2}(|000\rangle - |011\rangle - |101\rangle - |110\rangle)\right) = \tfrac{1}{2}(|001\rangle + |010\rangle - |100\rangle + |111\rangle), \tag{2.5}$$

so $a \oplus b \oplus c = 1$. The remaining cases where $xyz = 101$ and 110 are similar by the symmetry of state (2.3).

Note that a, b, and c by themselves are just random bits, uncorrelated with xyz. It is only the *trivariate correlations* among a, b, and c that are related to the input data xyz. Although the above task has some of the flavor of a communication complexity problem, it is technically different in that individual parties acquire no information.

We now construct a function based on the above where the presence of entanglement reduces its communication complexity.

Consider the function f defined on all triples $(x, y, z) \in \{0, 1, 2, 3\} \times \{0, 1, 2, 3\} \times \{0, 1, 2, 3\}$ which satisfy the condition

$$x + y + z \equiv 0 \pmod 2 \tag{2.6}$$

for which its value is given as

$$f(x, y, z) = \frac{(x + y + z) \bmod 4}{2} \tag{2.7}$$

(the value is always 0 or 1 by (2.6)). We represent the numbers x, y, and z in binary notation as x_1x_0, y_1y_0, and z_1z_0. In terms of these bits, the condition of (2.6) is

$$x_0 \oplus y_0 \oplus z_0 = 0, \tag{2.8}$$

and the function of (2.7) for inputs satisfying (2.8) is

$$f(x, y, z) = x_1 \oplus y_1 \oplus z_1 \oplus (x_0 \vee y_0 \vee z_0). \tag{2.9}$$

We assume the standard multiparty communication channel, where each bit that a party sends is broadcast to all other parties. Also, at the conclusion of the protocol, *all* parties must be able to determine the value of the function.

In the following two subsections, we show that with a prior entanglement, three bits of communication are sufficient to compute $f(x, y, z)$, whereas, without a prior entanglement, four bits of communication are necessary to compute $f(x, y, z)$.

2.1. The communication complexity with quantum entanglement is three bits. We now show that if Alice, Bob, and Carol initially share qubits Q_A, Q_B, and Q_C, respectively, in state (2.3), then there is a protocol for f where each party broadcasts only a single classical bit. The idea is based on applying the procedures at the beginning of this section using $x_0y_0z_0$ as the input. This requires no communication and provides Alice, Bob, and Carol with bits a, b, and c, respectively, such that $a \oplus b \oplus c = x_0 \vee y_0 \vee z_0$ (by (2.1)). Next Alice broadcasts the bit $(x_1 \oplus a)$, Bob broadcasts $(y_1 \oplus b)$, and Carol broadcasts $(z_1 \oplus c)$. At this point, each party knows $(x_1 \oplus a)$, $(y_1 \oplus b)$, and $(z_1 \oplus c)$, from which they can each determine the bit

$$\begin{aligned}(x_1 \oplus a) \oplus (y_1 \oplus b) \oplus (z_1 \oplus c) &= x_1 \oplus y_1 \oplus z_1 \oplus (a \oplus b \oplus c) \\ &= x_1 \oplus y_1 \oplus z_1 \oplus (x_0 \vee y_0 \vee z_0) \\ &= f(x, y, z),\end{aligned} \tag{2.10}$$

as required.

2.2. The communication complexity without quantum entanglement is four bits. In this section, we show that in the classical setting, four bits of communication are necessary to compute $f(x, y, z)$.

One can view any k-bit protocol as a binary tree of depth k, where each node that is not a leaf is labeled A(lice), B(ob), or C(arol). This labeling indicates which party will broadcast the next bit. An execution of the protocol corresponds to a path from the root of the tree to a leaf. Each leaf node is labeled 0 or 1, indicating the common output that results from the execution leading to that leaf. To establish our lower

bound, it suffices to show that no protocol-tree of depth three correctly computes $f(x, y, z)$.

We use the following lemma, which implies that in any correct protocol, all three parties must broadcast at least one bit.

LEMMA 2.1. *For any correct protocol-tree, on every path from its root to a leaf, each of A, B, and C must occur as a label at least once.*

Proof. Suppose that there exists a path along which one party, say, A, does not occur as a label. Let the leaf of that path be labeled $l \in \{0, 1\}$. Since this path does not include any reference to Alice's data, the same path is taken if x_1 is negated and all other input bits are held constant. However, by (2.9), negating x_1 also negates the value of $f(x, y, z)$, so the protocol cannot be correct for both possible values of x_1. □

Next suppose we have a protocol-tree of depth three for $f(x, y, z)$. Assume, without loss of generality, that the root of the tree is labeled A. The bit that Alice broadcasts is some function $\phi : \{0, 1\}^2 \to \{0, 1\}$ of her input data x alone. The function ϕ partitions $\{0, 1\}^2$ into two classes, $\phi^{-1}(0)$ and $\phi^{-1}(1)$. Call these two classes S_0 and S_1 and assume (without loss of generality) that $00 \in S_0$.

Next assume for the moment that the two children of the root of the protocol-tree are both labeled B (we shall see later that the other cases can be handled similarly). Then, by Lemma 2, the four children of B are all labeled C. Therefore, after Alice and Bob each send a bit, Carol must have enough information to determine the value of $f(x, y, z)$, since Carol broadcasts the third bit and does not gain any information from doing this. We shall show that this is impossible whatever S_0 and S_1 are.

There are two cases (the second of which has three subcases).

Case 1. $|S_0| = 1$. Recall that $00 \in S_0$, so $01, 10, 11 \in S_1$. Now, should the bit that Alice broadcasts specify that $x \in S_1$, Bob must follow this by broadcasting one bit from which Carol can completely determine the value of $f(x, y, z)$. Suppose that Bob sends the bit consistent with $y = 01$. If $z = 00$, then, from Carol's perspective, the possible values of (x, y, z) include $(01, 01, 00)$ and $(11, 01, 00)$ for which the respective values of $f(x, y, z)$ are 1 and 0. Therefore Carol cannot determine the value of $f(x, y, z)$ in this case.

Case 2. $|S_0| \geq 2$. There are three subcases where S_0 contains 01, 10, or 11 in addition to 00.

Case 2.1. S_0 *contains* 00 *and* 01. Here we consider the case where Alice broadcasts the bit specifying that $x \in S_0$. Bob must follow this by broadcasting one bit from which Carol can completely determine the value of $f(x, y, z)$. The bit that Bob broadcasts induces a partition of the possible values for y into two classes. If $z = 00$, then, from Carol's perspective, after receiving Alice's bit but before receiving Bob's bit, the possible values of (x, y, z) include $(00, 00, 00)$, $(00, 10, 00)$, $(01, 01, 00)$, and $(01, 11, 00)$, and the respective values of $f(x, y, z)$ on these points are 0, 1, 1, and 0. Therefore, for the protocol to be successful in this case, the partition that Bob's bit induces on y must place 00 and 11 in one class and 01 and 10 in the other class (otherwise Carol would not be able to determine $f(x, y, z)$ when $z = 00$). On the other hand, if $z = 01$, then, from Carol's perspective, the possible values of (x, y, z) include $(00, 01, 01)$, $(00, 11, 01)$, $(01, 00, 01)$, and $(01, 10, 01)$, and the respective values of $f(x, y, z)$ on these points are 1, 0, 1, and 0. Since we have established that Bob's bit does not distinguish between $y = 00$ and $y = 11$, Bob's bit is not sufficient information for Carol to determine $f(x, y, z)$ in this case.

Case 2.2. S_0 *contains* 00 *and* 10. The argument is similar to that in Case 1. Assume that Alice sends the bit specifying that $x \in S_0$. If Bob follows this by sending the bit consistent with $y = 00$ and $z = 00$, then, from Carol's perspective, the possible values of (x, y, z) include $(00, 00, 00)$ and $(10, 00, 00)$, and the respective values of $f(x, y, z)$ on these points are 0 and 1. Thus Carol cannot determine the value of $f(x, y, z)$ in this case.

Case 2.3. S_0 *contains* 00 *and* 11. The argument is similar to Case 2.1. Suppose that Alice broadcasts the bit specifying that $x \in S_0$. Consider Carol's perspective. If $z = 00$, then the possible values of (x, y, z) include $(00, 00, 00)$, $(00, 10, 00)$, $(11, 01, 00)$, and $(11, 11, 00)$, and the respective values of $f(x, y, z)$ on these points are 0, 1, 0, and 1; whereas, if $z = 01$, then the possible values of (x, y, z) include $(00, 01, 10)$, $(00, 11, 01)$, $(11, 00, 01)$, and $(11, 10, 01)$, and the respective values of $f(x, y, z)$ on these points are 1, 0, 0, 1. No binary partitioning of y will work for both possibilities.

The cases where the two children of the root of the protocol-tree are CC, CB, and BC have an analogous proof as above with the roles of B and C possibly reversed.

This completes the proof of the lower bound of four bits. The following deterministic four-bit protocol shows that this bound is tight.

A classical four bit protocol. First, Bob and Carol start by broadcasting the bits y_0, y_1 (Bob) and z_1 (Carol). After that, Alice now knows—by the promise of (2.8)—the bit $z_0 = x_0 \oplus y_0$ and hence all six bit values involved. The fourth and last bit of communication is therefore the announcement of the answer $f(x, y, z)$ by Alice to Bob and Carol.

3. Two-party probabilistic scenarios. The following scenario can be viewed as a reformulation of the nonlocality proof in [8] into data processing terminology. Alice and Bob receive input bits x and y, respectively, and, after this, they are forbidden from communicating with each other. Their goal is to produce output bits a and b, respectively, such that

$$a \oplus b = x \wedge y, \tag{3.1}$$

or, failing that, to satisfy this condition with as high a probability as possible.

To analyze the situation in terms of classical information, first consider the case of deterministic strategies. For these, Alice's output bit depends solely on her input bit x and similarly for Bob. Let a_0, a_1 be the two possibilities for Alice and b_0, b_1 be the two possibilities for Bob. These four bits completely characterize any deterministic strategy. Condition (3.1) translates into the equations

$$\begin{aligned} a_0 \oplus b_0 &= 0, \\ a_0 \oplus b_1 &= 0, \\ a_1 \oplus b_0 &= 0, \\ a_1 \oplus b_1 &= 1. \end{aligned} \tag{3.2}$$

It is impossible to satisfy all four equations simultaneously (since summing them modulo two yields $0 = 1$). Therefore it is impossible to satisfy condition (3.1) absolutely.

By using a probabilistic strategy, Alice and Bob can satisfy condition (3.1) with probability $\frac{3}{4}$. For such a strategy, we allow Alice and Bob to have a priori classical random variables, whose distribution is independent of that of the inputs x and y. Note that any three of the four equations of (3.2) *can* be simultaneously satisfied. The probabilistic strategy now works as follows. Alice and Bob have random variables

R_A and R_B, respectively, which are each uniformly distributed over $\{0,1,2,3\}$ and completely correlated with each other (i.e., $R_A = R_B$). These variables specify to both of them one of the four equations to violate while satisfying the other three. Alice and Bob then follow the deterministic procedure corresponding to a preagreed a_0, a_1, b_0, b_1 which satisfy the three equations determined by R_A and R_B. It is easy to see that (a) for any input xy, the resulting outputs satisfy condition (3.1) with probability $\frac{3}{4}$, and (b) this is optimal in that no probabilistic strategy can attain a success probability greater than $\frac{3}{4}$.

Now consider the same problem but where Alice and Bob are supplied with qubits Q_A and Q_B, respectively (instead of random variables), where the state of $\mathsf{Q}_A\mathsf{Q}_B$ is initialized to

$$\tfrac{1}{\sqrt{2}}(|00\rangle - |11\rangle). \tag{3.3}$$

It turns out that now the parties can produce data that satisfies condition (3.1) with probability $\cos^2(\frac{\pi}{8}) = 0.853\ldots$, which is higher than what is possible in the classical case. This is achieved by the following procedures:

Procedure for Alice:	Procedure for Bob:
if $x = 0$ **then**	**if** $y = 0$ **then**
apply $R(-\frac{\pi}{16})$ to Q_A	**apply** $R(-\frac{\pi}{16})$ to Q_B
else	**else**
apply $R(\frac{3\pi}{16})$ to Q_A	**apply** $R(\frac{3\pi}{16})$ to Q_B
measure Q_A **yielding bit** a	**measure** Q_B **yielding bit** b

In the above, $R(\theta)$ is the rotation by angle θ which is represented in the standard basis as

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}, \tag{3.4}$$

and the measurements are performed in the standard basis. If Alice rotates by θ_1 and Bob rotates by θ_2, then the state of $\mathsf{Q}_A\mathsf{Q}_B$ becomes

$$\tfrac{1}{\sqrt{2}}\left(\cos(\theta_1+\theta_2)(|00\rangle - |11\rangle) + \sin(\theta_1+\theta_2)(|01\rangle + |10\rangle)\right), \tag{3.5}$$

and, after the measurements, the probability that $a \oplus b = 0$ is $\cos^2(\theta_1 + \theta_2)$. It is now straightforward to verify that condition (3.1) is satisfied with probability $\cos^2(\frac{\pi}{8})$ for all input possibilities.

From the above, we can construct a function for which the presence of entanglement reduces its communication complexity in a probabilistic sense. Define $g : \{0,1\}^2 \times \{0,1\}^2 \to \{0,1\}$ as

$$g(x,y) = x_1 \oplus y_1 \oplus (x_0 \wedge y_0). \tag{3.6}$$

An execution of a probabilistic protocol for g is considered successful if and only if the value determined by Alice and the value determined by Bob are *both* correct.

In the following two subsections, we show that, with a prior quantum entanglement and two bits of communication, the probability of success can be at least $\cos^2(\frac{\pi}{8}) = 0.853\ldots$, whereas, with a shared random string instead of quantum entanglement and two bits of communication, the probability of success cannot exceed $\frac{3}{4}$. Thus, without prior entanglement, to achieve a success probability of at least $\cos^2(\frac{\pi}{8})$, *three* bits of communication are necessary.

TABLE 3.1
The values of $g(x, y)$. The columns are indexed by x and the rows are indexed by y.

$g(x,y)$	00	01	10	11
00	0	0	1	1
01	0	1	1	0
10	1	1	0	0
11	1	0	0	1

3.1. With quantum entanglement. Here we show that if Alice and Bob initially share qubits Q_A and Q_B, respectively, in state (3.3), then there is a protocol which successfully computes g with probability $\cos^2(\frac{\pi}{8})$. Alice and Bob first apply the procedures at the beginning of this section using x_0y_0 as input. This requires no communication and provides Alice and Bob with bits a and b, respectively, such that $\Pr[a \oplus b = x_0 \wedge y_0] = \cos^2(\frac{\pi}{8})$. Then Alice sends $(a \oplus x_1)$ to Bob, and Bob sends $(b \oplus y_1)$ to Alice. At this point, each party can determine the bit

$$(a \oplus x_1) \oplus (b \oplus y_1) = x_1 \oplus y_1 \oplus (a \oplus b), \tag{3.7}$$

which equals $x_1 \oplus y_1 \oplus (x_0 \wedge y_0) = g(x, y)$ with probability $\cos^2(\frac{\pi}{8})$, as required.

3.2. With shared classical random bits but no quantum entanglement. We now show that if Alice and Bob initially share classical random bits but no quantum entanglement, then there is no two-bit protocol in which both parties output the correct value of $g(x, y)$ with probability greater than $\frac{3}{4}$. By Theorem 3.20 of [23], it is sufficient to prove the lower bound on the error probability for all deterministic protocols with respect to *random inputs* from $\{0,1\}^2 \times \{0,1\}^2$ (which we can take to be uniformly distributed). As noted in section 2.2, we can represent any two-bit protocol as a binary tree of depth two with nonleaf nodes labeled A(lice) and B(ob).

Assume, without loss of generality, that the root of the protocol-tree is labeled A. The first bit that Alice sends is some function $\phi : \{0,1\}^2 \to \{0,1\}$ of her input data x alone. The function ϕ partitions $\{0,1\}^2$ into two classes $S_0 = \phi^{-1}(0)$ and $S_1 = \phi^{-1}(1)$. Let the first and second children of the root correspond to the paths traversed when the first bit sent (by Alice) indicates that $x \in S_0$ and $x \in S_1$, respectively. We must consider all partitions S_0 and S_1 in combination with all cases where the two children of the root are BB, AB, or AA (the case BA can be omitted by symmetry).

LEMMA 3.1. *If the child corresponding to S_i is labeled B, then, conditioned on $x \in S_i$, the probability that Bob correctly determines $g(x, y)$ is at most 1 if $|S_i| = 1$; $\frac{3}{4}$ if $|S_i| = 2$; $\frac{2}{3}$ if $|S_i| = 3$; and $\frac{1}{2}$ if $|S_i| = 4$. There is no well-defined probability for the empty set $|S_i| = 0$.*

Proof. The case where $|S_i| = 1$ is trivial.

For the case where $|S_i| = 2$, first consider the subcase where $S_i = \{00, 01\}$. Under the condition $x \in S_i$, (x, y) is a position in one of the first two columns of Table 3.1, and Alice's bit to Bob indicates this to him. From Bob's perspective, if $y = 00$, then $g(x, y) = 0$, so Bob can determine the correct answer. Similarly, if $y = 10$, then $g(x, y) = 1$, so Bob can determine the correct answer. However, if $y = 01$, then, since the first two columns of the table differ in this row, whatever function of Alice's message and y Bob computes, the probability that it will match $g(x, y)$ is at most $\frac{1}{2}$. Similarly, if $y = 01$, then Bob computes the correct answer with probability at most $\frac{1}{2}$. Since these four values of y are equiprobable, the probability that Bob correctly computes $g(x, y)$ conditioned on $x \in S_i$ is at most $\frac{1}{4} \cdot 1 + \frac{1}{4} \cdot 1 + \frac{1}{4} \cdot \frac{1}{2} + \frac{1}{4} \cdot \frac{1}{2} = \frac{3}{4}$. The other five subcases in which $|S_i| = 2$ are handled similarly.

For the case where $|S_i| = 3$, first consider the subcase where $S_i = \{00, 01, 10\}$. Under the condition $x \in S_i$, (x, y) is a position in one of the first three columns of Table 3.1, and Alice's bit to Bob indicates this to him. By looking at these three columns of Table 3.1, we observe that, from Bob's perspective, whatever the value of y, the probability of Bob determining $g(x, y)$ is at most $\frac{2}{3}$. The other two subcases in which $|S_i| = 3$ are handled similarly.

The last case $|S_i| = 4$ immediately implies $S_i = \{00, 01, 10, 11\}$, where Bob receives no information about the the string x of Alice. For all possible y's, the probability that Bob guesses $g(x, y)$ correctly is therefore $\frac{1}{2}$.

As the probabilities are conditioned on x being an element of S_i, the case of the empty set $|S_i| = 0$ is not well-defined. □

Now, by Lemma 3.1, if the two children of the root are BB, then the probability that Bob correctly determines $g(x, y)$ is at most $\frac{1}{4} \cdot 1 + \frac{3}{4} \cdot \frac{2}{3} = \frac{3}{4}$ if $|S_0| \neq |S_1|$, and $\frac{1}{2} \cdot \frac{3}{4} + \frac{1}{2} \cdot \frac{3}{4} = \frac{3}{4}$ if $|S_0| = |S_1|$.

Next, we show that for protocol-trees in which the two children of the root are *not* BB, the correctness probability is actually less than $\frac{3}{4}$.

LEMMA 3.2. *If the child corresponding to S_i is labeled A, then, conditioned on $x \in S_i$, the probability that Alice correctly determines $g(x, y)$ is at most $\frac{1}{2}$.*

Proof. If the condition $x \in S_i$ occurs, then Alice receives no information from Bob. Therefore, from Alice's perspective, the value of $g(x, y)$ is either y_1, $y_1 \oplus y_0$, $1 \oplus y_1$, or $1 \oplus y_1 \oplus y_0$ (corresponding to the cases $x = 00$, 01, 10, and 11, respectively). The result now follows from the fact that, from Alice's perspective, y is uniformly distributed over $\{0, 1\}^2$. □

By Lemma 3.2, it follows that if the two children of the root are AA, then the probability that Bob correctly determines $g(x, y)$ is at most $\frac{1}{2}$. The remaining case is where the two children of the root are AB. By applying Lemma 3.2 for the first child and Lemma 3.1 for the second child, the probability that both Alice and Bob correctly determine $g(x, y)$ is at most

- $\frac{1}{4} \cdot \frac{1}{2} + \frac{3}{4} \cdot \frac{2}{3} = \frac{5}{8}$ if $|S_0| = 1$ and $|S_1| = 3$,
- $\frac{1}{2} \cdot \frac{1}{2} + \frac{1}{2} \cdot \frac{3}{4} = \frac{5}{8}$ if $|S_0| = 2$ and $|S_1| = 2$,
- $\frac{3}{4} \cdot \frac{1}{2} + \frac{1}{4} \cdot 1 = \frac{5}{8}$ if $|S_0| = 3$ and $|S_1| = 1$.

This completes the proof that no two-bit protocol is correct with probability more than $\frac{3}{4}$. There is a straightforward errorless three-bit protocol.

4. The qubit model of communication complexity. In the previous two sections, novel protocols were obtained in the *entanglement model* for communication complexity, where communication is with classical bits, but the parties have an a priori supply of entangled qubits. In the *qubit model* (introduced by Yao [34] and Kremer [22]), the parties have no entanglement but are allowed to communicate with qubits in place of classical bits. Qubits cannot be broadcast [32], so in a multiparty setting a qubit of communication must be sent to a specific party.

In this section, we show how to translate the protocols from sections 2 and 3 in the entanglement model into protocols in the qubit model. By doing so we also prove the same separation between qubit communication complexity and classical communication complexity.

4.1. A deterministic three-party qubit protocol. The following protocol requires only two qubits plus one classical bit of communication to compute f from section 2 with the one-qubit rotation R that we used earlier.

Procedure for Alice:
initialize qubit Q **to state** $|0\rangle$
apply $R(x \cdot \frac{\pi}{4})$ **to** Q
send Q **to Bob**

Procedure for Bob:
receive Q **from Alice**
apply $R(y \cdot \frac{\pi}{4})$ **to** Q
send Q **to Carol**

Procedure for Carol:
receive Q **from Bob**
apply $R(z \cdot \frac{\pi}{4})$ **to** Q
measure Q**, yielding bit** m
announce the answer m

It is straightforward to verify that the final state of qubit Q is

$$\begin{aligned} R(z \cdot \tfrac{\pi}{4}) \cdot R(y \cdot \tfrac{\pi}{4}) \cdot R(x \cdot \tfrac{\pi}{4})|0\rangle &= \cos((x+y+z) \cdot \tfrac{\pi}{4})|0\rangle + \sin((x+y+z) \cdot \tfrac{\pi}{4})|1\rangle \\ &= |f(x,y,z)\rangle, \end{aligned} \tag{4.1}$$

which implies that the answer as announced by Carol is indeed correct.

4.2. A probabilistic two-party qubit protocol. Also the two-party problem of section 2 can be translated to the qubit model. The following protocol computes g from section 2 with correctness probability $0.853\ldots$ using only one qubit and one bit of communication:

Procedure for Alice:
initialize qubit Q **to state** $|0\rangle$
apply $R((2x_1 + x_0 - \frac{1}{2}) \cdot \frac{\pi}{4})$ **to** Q
send Q **to Bob**

Procedure for Bob:
receive Q **from Alice**
apply $R((2y_1 + y_0) \cdot \frac{\pi}{4})$ **to** Q
measure Q**, yielding bit** m
announce the answer m

For any combination of input x and y, this final answer m will be the correct value $g(x,y)$ with probability $\cos^2(\frac{\pi}{8}) = 0.853\ldots$, which is identical to the success rate of the entanglement protocol of section 3.1.

5. Discussion of subsequent work. After the publication of [9] and the announcement of this article in 1997, several other results in quantum communication complexity were obtained. In [6] the three-party problem of Chapter 2 was generalized into a k-party problem for which the separation between quantum and classical communication complexity is k versus $\Theta(k \log k)$ bits. For the one-round, three-party setting this article also proved a difference of $n+1$ versus $(3/2)n+1$ bits between communication with and without initial entanglement.

A lower bound of $\Omega(n)$ on the quantum communication complexity of the (two-party) Inner Product function in [10] showed that the entanglement model does not always allow an improvement over the classical scenario. On the other hand, a significant decrease in communication complexity was established for the Disjoint function

$$\textsc{Disjoint}(x,y) = \begin{cases} 0 & \text{if there exists an } i \text{ such that } x_i = y_i = 1, \\ 1 & \text{otherwise.} \end{cases} \tag{5.1}$$

This well-studied problem has a classical probabilistic communication complexity of $\Omega(n)$ [18, 28], while the authors of [5] gave a qubit protocol requiring only $O(\sqrt{n} \log n)$ qubits of communication. The question whether there exists a more efficient quantum protocol for Disjoint is still an important open problem. This is especially relevant as

DISJOINT is a complete problem for the communication class "co-NP" [2]. The same article [5] also contained the first exponential separation for the exact distributed computation of a partial two-party function that is related to the Deutsch–Jozsa problem of [11].

In [27] Raz improved on these results by establishing an exponential separation between classical and quantum communication in the bounded-error probabilistic setting. The problem involved is the question for Alice whether her normalized n-dimensional vector $\vec{v} \in \mathbf{C}^n$, after Bob's unitary transformation U, lies in a particular subspace $S \subset \mathbf{C}^n$ or in the orthogonal complement $S^\perp$. (The promise here is that the vector $U\vec{v}$ is close to either S or $S^\perp$.) It is clear that this can be solved with only $2 \log n$ qubits of communication if we store the coefficients of $\vec{v}$ (and $U\vec{v}$) in the amplitudes of a $\log n$ qubit message. The classical lower bound, on the other hand, was proved to be polynomial in n. It is currently still an open problem if it is possible to have an exponential quantum vs. classical reduction in communication complexity for a *total* function.

The "sampling complexity" of a function f and a probability distribution μ is the amount of communication that is required to create a mixture of the possible states $(f(x, y), x, y)$ according to the distribution $\mu(x, y)$ over the input states. In [1] it was shown that we can have an exponential gap between the quantum and the classical sampling complexity of the DISJOINT function.

Separations for nondeterministic (quantum) communication complexity are exhibited in the articles [31] and [24]. In [7], there is a general framework for establishing lower bounds on exact communication complexity with entanglement. For the zero-error (Las Vegas) model, Klauck [20] has given a polynomial difference between the quantum and the classical setting. The question whether quantum information, in general, can reduce the number of rounds is addressed in [21]. (See [29, 19] for spectacular examples of such a reduction in the context of interactive proof systems.)

The preliminary communication complexity results that appear in this article were inspired by examples of quantum nonlocality. Conversely, in [4] new powerful examples of nonlocality are given that follow from the results in [5].

Acknowledgments. We would like to thank Charles Bennett, Lance Fortnow, Richard Jozsa, and Lev Vaidman for helpful discussions.

REFERENCES

[1] A. AMBAINIS, L.J. SCHULMAN, A. TA-SHMA, U. VAZIRANI, AND A. WIGDERSON, *The quantum communication complexity of sampling*, in Proceedings of the 39th Annual Symposium on Foundations of Computer Science, 1998, pp. 342–351.

[2] L. BABAI, P.G. FRANKL, AND J. SIMON, *Complexity classes in communication complexity theory*, in Proceedings of the 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 337–347.

[3] J.S. BELL, *On the Einstein-Podolsky-Rosen paradox*, Physics, 1 (1964), pp. 195–200.

[4] G. BRASSARD, R. CLEVE, AND A. TAPP, *Cost of exactly simulation quantum entanglement with classical communication*, Phys. Rev. Lett., 83 (1999), pp. 1874–1877.

[5] H. BUHRMAN, R. CLEVE, AND A. WIGDERSON, *Quantum vs. classical communication and computation*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, 1998, pp. 63–68.

[6] H. BUHRMAN, W. VAN DAM, P. HØYER, AND A. TAPP, *Multiparty quantum communication complexity*, Phys. Rev. A, 60 (1999), pp. 2737–2741.

[7] H. BUHRMAN AND R. DE WOLF, *Communication complexity lower bounds by polynomials*, in Proceedings of the 16th Annual Conference on Computational Complexity, 2001, to appear.

[8] J.F. CLAUSER, M.A. HORNE, A. SHIMONY, AND R.A. HOLT, *Proposed experiment to test local hidden-variable theories*, Phys. Rev. Lett., 23 (1969), pp. 880–884.

[9] R. Cleve and H. Buhrman, *Substituting quantum entanglement for communication*, Phys. Rev. A, 56 (1997), pp. 1201–1204.
[10] R. Cleve, W. van Dam, M. Nielsen, and A. Tapp, *Quantum entanglement and the communication complexity of the inner product function*, in Proceedings of the First NASA International Conference on Quantum Computing and Quantum Communications, Colin P. Williams, ed., Lecture Notes in Comput. Sci. 1509, Springer-Verlag, Berlin, 1999, pp. 61–74.
[11] D. Deutsch and R. Jozsa, *Rapid solution of problems by quantum computation*, Proc. Roy. Soc. London Ser. A, 439 (1992), pp. 553–558.
[12] A. Einstein, *The Born-Einstein Letters; Correspondence between Albert Einstein and Max and Hedwig Born from* 1916 *to* 1955, Walker, New York, 1971.
[13] A. Einstein, B. Podolsky, and N. Rosen, *Can quantum-mechanical description of physical reality be complete?*, Phys. Rev., 47 (1935), pp. 777–780.
[14] D.M. Greenberger, M. Horne, and A. Zeilinger, *Going beyond Bell's theorem*, in Bell's Theorem, Quantum Theory, and Conceptions of the Universe, M. Kafatos, ed., Kluwer Academic, Dordrecht, 1989, pp. 69–72.
[15] L.K. Grover, *Quantum Telecomputation*, quant-ph archive, report 9704012, 1997.
[16] P. Hausladen, R. Jozsa, B. Schumacher, M. Westmoreland, and W.K. Wootters, *Classical information capacity of a quantum channel*, Phys. Rev. A (3), 54 (1996), pp. 1869–1876.
[17] A.S. Holevo, *Some estimates of the information transmitted by quantum communications channels*, Problemy Peredachi Informatsii, 9 (1973), pp. 3–11, Problems of Information Transmission (USSR), 9 (1973), pp. 177–183 (in English).
[18] B. Kalyanasundaram and G. Schnitger, *The probabilistic communication complexity of set intersection*, SIAM J. Discrete Math., 5 (1992), pp. 545–557.
[19] A. Kitaev and J. Watrous, *Parallelization, amplification, and exponential time simulation of quantum interactive proof systems*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 608–617.
[20] H. Klauck, *On quantum and probabilistic communication: Las Vegas and one-way protocols*, in Proceedings of the 32nd Annual ACM Symposium on Theory of Computing, 2000, pp. 644–651.
[21] H. Klauck, A. Nayak, A. Ta-Shma, and D. Zuckerman, *Interaction in quantum communication and the complexity of set disjointness*, in Proceedings of the 33rd Annual ACM Symposium on Theory of Computing, 2001, to appear.
[22] I. Kremer, *Quantum Communication*, Master's thesis, Hebrew University of Jerusalem, Jerusalem, Israel, 1995.
[23] E. Kushilevitz and N. Nisan, *Communication Complexity*, Cambridge University Press, Cambridge, UK, 1997.
[24] S. Massar, D. Bacon, N. Cerf, and R. Cleve, *Classical simulation of quantum entanglement without local hidden variables*, Phys. Rev. A, to appear.
[25] N.D. Mermin, *Is the moon there when nobody looks? Reality and the quantum theory*, Phys. Today, 38 (1985), pp. 38–47.
[26] N.D. Mermin, *What's wrong with these elements of reality?*, Phys. Today, 43 (1990), pp. 9–11.
[27] R. Raz, *Exponential separation of quantum and classical communication complexity*, in Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 358–367.
[28] A.A. Razborov, *On the distributional complexity of disjointness*, Theoret. Comput. Sci., 6 (1992), pp. 385–390.
[29] J. Watrous, *PSPACE has constant-round quantum interactive proof systems*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, 1999, pp. 341–351.
[30] J.A. Wheeler and W.H. Zurek, eds., *Quantum Theory and Measurement*, Princeton University Press, Princeton, NJ, 1983.
[31] R. de Wolf, *Characterization of non-deterministic quantum query and quantum communication complexity*, in Proceedings of the 15th Annual IEEE Conference on Computational Complexity, 2000, pp. 271–278.
[32] W.K. Wootters and W.H. Zurek, *A single quantum cannot be cloned*, Nature, 299 (1982), pp. 802–803.
[33] A.C. Yao, *Some complexity questions related to distributed computing*, in Proceedings of the 11th Annual ACM Symposium on Theory of Computing, 1979, pp. 209–213.
[34] A.C. Yao, *Quantum circuit complexity*, in Proceedings of the 34th IEEE Symposium on Foundations of Computer Science, 1993, pp. 352–361.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1842–1862

REGULAR LANGUAGES ARE TESTABLE WITH A CONSTANT NUMBER OF QUERIES*

NOGA ALON†, MICHAEL KRIVELEVICH‡, ILAN NEWMAN§, AND MARIO SZEGEDY¶

Abstract. We continue the study of combinatorial property testing, initiated by Goldreich, Goldwasser, and Ron in [*J. ACM*, 45 (1998), pp. 653–750]. The subject of this paper is testing regular languages. Our main result is as follows. For a regular language $L \in \{0,1\}^*$ and an integer n there exists a randomized algorithm which always accepts a word w of length n if $w \in L$ and rejects it with high probability if w has to be modified in at least ϵn positions to create a word in L. The algorithm queries $\tilde{O}(1/\epsilon)$ bits of w. This query complexity is shown to be optimal up to a factor polylogarithmic in $1/\epsilon$. We also discuss the testability of more complex languages and show, in particular, that the query complexity required for testing context-free languages cannot be bounded by any function of ϵ. The problem of testing regular languages can be viewed as a part of a very general approach, seeking to probe testability of properties defined by logical means.

Key words. property testing, regular languages, context-free languages

AMS subject classifications. 68Q25, 68Q45, 68W20

PII. S0097539700366528

1. Introduction.

Property testing deals with the question of deciding whether a given input x satisfies a prescribed property P or is "far" from any input satisfying it. Let P be a property, i.e., a nonempty family of binary words. A word w of length n is called ϵ-far from satisfying P if no word w' of the same length, which differs from w in no more than ϵn places, satisfies P. An *ϵ-test* for P is a randomized algorithm, which, given the quantity n and the ability to make queries about the value of any desired bit of an input word w of length n, distinguishes with probability at least $2/3$ between the case of $w \in P$ and the case of w being ϵ-far from satisfying P. Finally, we say that property P is *(c,ϵ)-testable* if for every $\epsilon > 0$ there exists an ϵ-test for P whose total number of queries is bounded by c.

Property testing was defined by Goldreich, Goldwasser, and Ron [7] (inspired by [13]). It emerges naturally in the context of PAC learning, program checking [6, 3, 10, 13], probabilistically checkable proofs [2], and approximation algorithms [7].

*Received by the editors February 3, 2000; accepted for publication (in revised form) September 27, 2000; published electronically March 13, 2001. A preliminary version of this paper appeared in the Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, pp. 645–655.

http://www.siam.org/journals/sicomp/30-6/36652.html

†Department of Mathematics, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel, and AT&T Labs–Research, Florham Park, NJ 07932 (noga@math.tau.ac.il). This author's research was supported by a USA Israeli BSF grant, by a grant from the Israel Science Foundation, and by the Hermann Minkowski Minerva Center for Geometry at Tel Aviv University.

‡Department of Mathematics, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University, Tel Aviv 69978, Israel (krivelev@math.tau.ac.il). Part of this research was performed when this author was with DIMACS Center, Rutgers University, Piscataway NJ, 08854, and AT&T Labs–Research, Florham Park, NJ 07932. This author's research was supported in part by a DIMACS postdoctoral fellowship.

§Department of Computer Science, University of Haifa, Haifa, Israel (ilan@cs.haifa.ac.il). Part of this research was performed when this author was visiting AT&T Labs–Research, Florham Park, NJ 07932.

¶School of Mathematics, Institute for Advanced Study, Olden Lane, Princeton, NJ 08540 (szegedy@math.ias.edu). Part of this research was performed when this author was with AT&T Labs–Research, Florham Park, NJ 07932.

In [7], the authors mainly consider graph properties, such as bipartiteness, and they show (among other things) the quite surprising fact that testing bipartiteness can be done by randomly testing a polynomial in $1/\epsilon$ number of edges of the graph, answering the above question with constant probability of failure. They also raise the question of obtaining general results as to when there is, for every $\epsilon > 0$, an ϵ-test for a property using $c = c(\epsilon)$ queries (i.e., c is a function of ϵ but independent of n) with constant probability of failure. We call properties of this type ϵ-*testable*. So far, such answers are quite sparse; some interesting examples are given in [7], and several additional ones can be obtained by applying the regularity lemma as we show in a subsequent paper [1].

In this paper we address testability of formal languages (see [8] as a general reference). A language $L \subseteq \{0,1\}^*$ is a property which is usually viewed as a sequence of Boolean functions $f_n : \{0,1\}^n \to \{0,1\}$, with $f_n^{-1}(1) = L \cap \{0,1\}^n = L_n$. Our main result states that all regular languages are ϵ-testable with a query complexity of only $\tilde{O}(1/\epsilon)$. We also show that this complexity is optimal up to a factor polylogarithmic in $1/\epsilon$. This positive result cannot be extended to context-free languages, for there is an example of a very simple context-free language which is not testable.

Since regular languages can be characterized using second order monadic logic, we thus obtain a large set of logically defined objects which are testable. In [1] we provide testable graph properties described by logical means as well. These results indicate a strong interrelation between testability and logic. Although our result on regular languages can be viewed as a separate result having no logical bearing at all, our opinion is that logic does provide the right context for testability problems, which may lead to the discovery of further classes of testable properties.

The rest of this paper is organized as follows. In section 2 we present the proof of the main result showing that every regular language is testable. In section 3 we show that the upper bound of $\tilde{O}(1/\epsilon)$ for the query complexity of testing regular languages, obtained in Theorem 2.9, is tight up to a polylogarithmic factor. Section 4 is devoted to the discussion of testability of context-free languages. There we show, in particular, that there exist nontestable context-free languages. We also discuss testability of the Dyck languages. Section 5 contains some concluding remarks and outlines new research directions.

2. Testing regular languages. In this section we prove the main result of the paper, namely, that regular languages are $(\tilde{O}(\frac{1}{\epsilon}), \epsilon)$-testable. As this result is asymptotic, we assume that n is big enough with respect to $\frac{1}{\epsilon}$ (and with respect to any other constant that depends only on the fixed language we are working with). All logarithms are binary unless stated explicitly otherwise.

We start by recalling the standard definition of a regular language, based on finite automata. This definition is convenient for algorithmic purposes.

DEFINITION 2.1. *A deterministic finite automaton (DFA) M over $\{0,1\}$ with states $Q = \{q_1, \ldots, q_m\}$ is given by a function $\delta : Q \times \{0,1\} \to Q$ together with a set $F \subseteq Q$. One of the states, q_1, is called the initial state. The states belonging to the set F are called accepting states, and δ is called the transition function.*

We can extend the transition function δ to $\{0,1\}^*$ recursively as follows. Let γ denote the empty word. Then

$$\begin{aligned} \delta(q, \gamma) &= q, \\ \delta(q, u0) &= \delta(\delta(u, q), 0), \\ \delta(q, u1) &= \delta(\delta(u, q), 1). \end{aligned}$$

Thus, if M starts in a state q and processes string u, then it ends up in a state $\delta(q,u)$.

We then say that M accepts a word u if $\delta(q_1,u) \in F$. M rejects u means that $\delta(q_1,u) \in Q \setminus F$. Finally, the language accepted by M, denoted by L_M, is the set of all $u \in \{0,1\}^*$ accepted by M. We use the following definition of regular languages.

DEFINITION 2.2. *A language is regular iff there exists a finite automaton that accepts it.*

Therefore, we assume in this section that a regular language L is given by its automaton M so that $L = L_M$.

A word w of length n defines a sequence of states $(q_{i_0}, \dots, q_{i_n})$ in the following natural way: $q_{i_0} = q_1$, and for $1 \le j \le n$, $q_{i_j} = \delta(q_1, w[1] \dots w[j])$. This sequence describes how the automaton M moves while reading w. Later in the paper we will occasionally refer to this sequence as the *traversal path* of w.

A finite automaton M defines a directed graph $G(M)$ by $V(G(M)) = Q$ and $E(G(M)) = \{(q_i,q_j) \mid \delta(q_i,0) = q_j\} \cup \{(q_i,q_j) \mid \delta(q_i,1) = q_j\}$. The *period* $g(G)$ of a directed graph G is the greatest common divisor of cycle lengths in G. If G is acyclic, we set $g(G) = \infty$.

We will use the following lemma about directed graphs.

LEMMA 2.3. *Let $G = (V,E)$ be a nonempty, strongly connected directed graph with a finite period $g(G)$. Then there exist a partition $V(G) = V_0 \cup \dots \cup V_{g-1}$ and a constant $m = m(G)$ which does not exceed $3|V|^2$ such that the following hold.*

1. *For every $0 \le i, j \le g-1$ and for every $u \in V_i$, $v \in V_j$ the length of every directed path from u to v in G is $(j-i) \bmod g$.*
2. *For every $0 \le i, j \le g-1$, for every $u \in V_i$, $v \in V_j$, and for every integer $r \ge m$, if $r = (j-i) \pmod g$, then there exists a directed path from u to v in G of length r.*

Proof. To prove part 1, fix an arbitrary vertex $z \in V$ and for each $0 \le i \le g-1$, let V_i be the set of all those vertices which are reachable from v by a directed (not necessarily simple) path of length $i \bmod g$. Note that since any closed (directed) walk in G is a disjoint union of cycles, the length of each such walk is divisible by g. This implies that the sets V_i are pairwise disjoint. Indeed, assume this is false, and suppose w lies in $V_i \cap V_j$ with $i \ne j$. As G is strongly connected, there is a path p_1 from w to z, and by definition there are a path p_2 of length $i \bmod g$ from z to w as well as a path p_3 of length $j \bmod g$ from z to w. Now the number of edges of either $p_1 \cup p_2$ or $p_1 \cup p_3$ is not divisible by g, which is impossible. Therefore, the sets V_i form, indeed, a partition of V. For $u \in V_i$ and $v \in V_j$, the union of any (directed) path from z to u with a (directed) path from u to v forms a path from z to v, and as any such path must have length $j \bmod g$; the assertion of part 1 follows.

We next prove part 2. Consider any set of positive integers $\{a_i\}$ whose greatest common divisor is g. It is well known that there is a smallest number t such that every integer $s \ge t$ which is divisible by g is a linear combination with nonnegative integer coefficients of the numbers a_i. Moreover, it is known (see [9, 5]) that t is smaller than the square of the maximal number a_i. Fix a closed (directed) walk in G that visits all vertices and whose length is at most $|V|^2$. (This is easily obtained by numbering the vertices of G arbitrarily as $v_0, v_1, \dots, v_{k-1}$ and by concatenating directed paths from v_i to v_{i+1} for each $0 \le i \le k-1$, where the indices are taken modulo k.) Now associate the set of cycle lengths in this walk with the set of positive integers $\{a_i\}$ as above. Then, following this closed walk and traversing each directed cycle as many times as desired, we conclude that every integer divisible by g and exceeding $2|V|^2$ is the length of a closed walk passing through all vertices of the graph. Given now a

vertex $u \in V_i$, a vertex $v \in V_j$, and an integer $r > 3|V|^2$ satisfying $r = (j-i) \bmod g$, fix a shortest path p from u to v, and note that its length l satisfies $l = (j-i) \bmod g$ and $l < |V| (\leq |V|^2)$. Adding to p a closed walk of length $r-l$ from v to itself, we obtain the required path, completing the proof. □

We call the constant m from the above lemma the *reachability constant* of G and denote it by $m(G)$. In what follows we assume that m is divisible by g.

If $L_M \cap \{0,1\}^n = \emptyset$, a testing algorithm can reject any input without reading it at all. Therefore, we can assume that we are in the nontrivial case $L_M \cap \{0,1\}^n \neq \emptyset$.

We now introduce a key definition for what follows.

DEFINITION 2.4. *Given a word $w \in \{0,1\}^n$, a subword (run) w' of w starting at position i is called* feasible *for language L_M if there exists a state $q \in Q$ such that q is reachable from q_1 in G in exactly $i-1$ steps, and there is a path of length $n - (|w'| + i - 1)$ in G from the state $\delta(q, w')$ to at least one of the accepting states. Otherwise, w' is called* infeasible.

Of course, finding an infeasible run in w proves that $w \notin L$. Our aim is to show that if a given word w of length n is far from any word of length n in L, then many short runs of w are infeasible. Thus a choice of a small number of random runs of w almost surely contains an infeasible run. First we treat the following basic case.

DEFINITION 2.5. *We call an automaton M* "essentially strongly connected" *if*

(1) *M has a unique accepting state q_{acc};*
(2) *the set of states of the automaton, Q, can be partitioned into two parts, C and D, so that*
   - *$q_1, q_{acc} \in C$,*
   - *the subgraph of $G(M)$ induced on C is strongly connected, and*
   - *no edges in $G(M)$ go from D to C (but edges can go from C to D).*

   (Note that D may be empty.)

LEMMA 2.6. *Assume that the language $L = L_M$ contains some words of length n and that M is essentially strongly connected with C and D being the partition of the states of M as in Definition 2.5. Let m be the reachability constant of $G[C]$. Assume also that $\epsilon n \geq 64m \log(4m/\epsilon)$. Then if, for a word w of length $|w| = n$, $dist(w, L) \geq \epsilon n$, then there exists an integer $1 \leq i \leq \log(4m/\epsilon)$ such that the number of infeasible runs of w of length 2^{i+1} is at least $\frac{2^{i-4}\epsilon n}{m \log(4m/\epsilon)}$.*

Proof. Our intention is to construct a sequence $(R_j)_{j=1,\ldots}$ of disjoint infeasible runs, each being minimal in the sense that each of their prefixes is feasible, and so that each is a subword of the given word w. We then show that we can concatenate these subwords to form a word in the language that is not too far from w. ("Not too far" will essentially depend on the number of runs that we have constructed.) This in turn will show that if $dist(w, L) \geq \epsilon n$, then there is a lower bound on the number of these infeasible runs.

For reasons to become obvious later, we also want these runs to be in the interval $[m+1, n-m]$.

A natural way to construct such a sequence is to repeat the following procedure starting from coordinate $m+1$. Let R_1 be the shortest infeasible run starting from $w[m+1]$ and ending before $w[n-m+1]$. If there is no such run we stop. Assume that we have constructed so far $R_1, \ldots, R_{j-1}$ ending at $w[c_{j-1}]$; next we construct R_j by taking the minimal infeasible run starting at $w[c_{j-1}+1]$ and ending before $w[n-m+1]$. Again, if there is no such run, we stop.

Assume we have constructed in this way runs $R_1, \ldots, R_h$. Note that each run is a subword of w, the runs are pairwise disjoint, and their concatenation in order

forms a (continuous) subword of w. Also note that by the definition of each run, R_j being minimal infeasible, its prefix $R_j^{(-)}$, obtained by discarding the last bit of R_j, is feasible. This, in turn, implies that R'_j, which is obtained from R_j by flipping its last bit, is feasible. In addition, by Definition 2.4, this means that for each R'_j there is a state $q_{i_j} \in C$ so that $\delta(q_{i_j}, R'_j) \in C$ and such that q_{i_j} is reachable from q_1 in $c_{j-1}+1$ steps.

Next we inductively construct a word $w^* \in L$ such that $dist(w, w^*) \leq hm+2m+2$. Assuming that $dist(w, L) \geq \epsilon n$, this will imply a lower bound on h. The general idea is to "glue" together the R'_j for $j = 1, \ldots, h$, each being feasible and yet very close to a subword of w (except for the last bit in each). The only concern is to glue the pieces together so that as a whole word they will be feasible. This will require an extra change of m bits per run, plus some additional $2m$ bits at the end of the word.

We maintain during the induction that, for $j = 0, \ldots$, the word w_j we construct is feasible starting from position 1, and it ends in position c_j. For the base case, let $c_0 = m$, and let w_0 be any word of length m which is feasible starting from position 1. Assume we have already defined a word w_{j-1} feasible from position 1 and ending in position c_{j-1}. Let $\delta(q_1, w_{j-1}) = p_j$. As both p_j and q_{i_j} are reachable from q_1 by a path of length c_{j-1}, according to Lemma 2.3 we can change the last m bits in w_{j-1} so that we get a word u_j for which $\delta(q_1, u_j) = q_{i_j}$. We now define w_j as a concatenation of u_j and R'_j. Let w_h be the final word that is defined in this way, ending at place c_h. There are two possible reasons we have stopped with R_h. One possible reason is that there is no infeasible run starting at $c_h + 1$, in which case, changing the last m bits of w_h and concatenating to it the remaining suffix of w (that starts at position $c_h + 1$), exactly as in the case of adding R'_j, yields the required w^*. The other possible reason for stopping the growth of R_h is when there is a minimal infeasible run that starts at $c_h + 1$ and ends after position $n - m + 1$. Let R be that run, and let R' be the run obtained by flipping the last bit of R. As was the case with any R'_j, R' is feasible from position $c_h + 1$. Hence there is a feasible word u of which R' is a prefix and u is of length $n - c_h$ so that $\delta(q_{i_h}, u) = q_{acc}$. We can construct w^* from w_h and u exactly as we have constructed w^* from w_h and the suffix of w in the previous case.

By the definition of w^*, $w^* \in L$. Following the inductive construction of w^*, we have that for $1 \leq j \leq h$, $dist(w_j, w[1, c_j]) \leq jm + 1$. Then to get from w_h to w^* we start with R', which is either a subword of w (as in the first case previously discussed) or a subword of w where one bit was changed (as in the second case), and we continue by changing m bits at the end of w_h and possibly additional m bits at the end of u. Therefore, $dist(w, w^*) \leq hm + 2m + 2$, as we claimed.

Recalling that $dist(w, L) \geq \epsilon n$, we conclude that $h \geq \frac{\epsilon n - 2}{m} - 2 \geq \epsilon n/(2m)$. (The last inequality is by our assumptions that $\epsilon n \geq 64m \log(4m/\epsilon)$.) This already shows that if $dist(w, L) \geq \epsilon n$, then there are $\Omega(\epsilon n)$ many disjoint infeasible runs in w. However, we need a stronger dependence, as stated in the lemma. We achieve this in the following way.

Let $a = \log(4m/\epsilon)$. For $1 \leq i \leq a$, denote by s_i the number of runs in $\{R_j\}_{j=1}^h$ whose length falls in the interval $[2^{i-1} + 1, 2^i]$. As $|\{R_j : 1 \leq j \leq h, |R_j| > 4m/\epsilon\}| < \epsilon n/(4m)$, we get $\sum_{i=1}^a s_i \geq h - \epsilon n/(4m) \geq \epsilon n/(4m)$. Therefore, there exists an index i for which $s_i \geq \epsilon n/(4am)$. Consider all infeasible runs R_j with $|R_j| \in [2^{i-1} + 1, 2^i]$. Note that if a run contains an infeasible subrun, then it is infeasible by itself. Now, each infeasible run of length between $2^{i-1} + 1$ and 2^i is contained in at least 2^i infeasible runs of length 2^{i+1}, except maybe for the first two and the last two runs (those with the two smallest j's and those with the two largest j's). As R_j are

disjoint, each infeasible run of length 2^{i+1} contains at most three of the R_j's of length at least $2^{i-1}+1$. Thus, we a get a total of at least $(2^i/3)(s_i-4)$ infeasible runs of length at most 2^{i+1}. By our assumption on the parameters, this number is $(2^i/3)(s_i-4) \geq \frac{2^i}{3}(\frac{\epsilon n}{4am}-4) = 2^{i-4}(\frac{\epsilon n}{am}+\frac{1}{3}\frac{\epsilon n-64am}{am}) \geq \frac{2^{i-4}\epsilon n}{m\log(4m/\epsilon)}$, as claimed. ∎

Now our aim is to reduce the general case to the above described case. For a given DFA M with a graph $G=G(M)$, we denote by $\mathcal{C}(G)$ the *graph of components* of G, whose vertices correspond to maximal by inclusion strongly connected components of G and whose directed edges connect components of G, which are connected by some edge in G. Note that some of the vertices of $\mathcal{C}(G)$ may represent single vertices of G with no self loops that do not belong to any strongly connected subgraph of G with at least two vertices. All other components have nonempty paths inside them and will be called *truly connected*. From now on we reserve k for the number of vertices of $\mathcal{C}(G)$ and set $V=V(G)$. We may assume that all vertices of G are reachable from the initial state q_1. Then $\mathcal{C}(G)$ is an acyclic graph in which there exists a directed path from a component C_1, containing q_1, to every other component. Denote $m=\max_j(m(C_j))$, $l=lcm(\{g(G[C_j])\})$, where j runs over all truly connected components of G, corresponding to vertices of $\mathcal{C}(G)$. We will assume in what follows that the following relations are satisfied between the parameters.

Condition (*).

- $\frac{\epsilon n}{2k} \geq 64m\log\frac{8mk}{\epsilon}$.
- $\epsilon n > 8km$.
- $\epsilon\log(1/\epsilon) < \frac{1}{258k^2|V|^2m(l+m)}$.

Clearly, for any fixed k,m,l for ϵ small enough, and n large enough, condition (*) holds.

Our next step is to describe how a word $w \in L_M$ of length n can move along the automaton. If a word w belongs to L, it traverses G, starting from q_1 and ending in one of the accepting states. Accordingly, w traverses $\mathcal{C}(G)$, starting from C_1 and ending in a component containing an accepting state. For this reason, we call a path A in $\mathcal{C}(G)$ *admissible* if it starts at C_1 and ends at a component with an accepting state. Given an admissible path $A=(C_{i_1},\dots,C_{i_t})$ in $\mathcal{C}(G)$, a sequence $P=(p_j^1,p_j^2)_{j=1}^t$ of pairs of vertices of G (states of M) is called an *admissible sequence of portals* if it satisfies the following restrictions.

1. $p_j^1,p_j^2 \in C_{i_j}$ for every $1 \leq j \leq t$.
2. $p_1^1 = q_1$.
3. $p_t^2 \in F$ (i.e., p_t^2 is an accepting state of M).
4. For every $2 \leq j \leq t$ one has $(p_{j-1}^2,p_j^1) \in E(G)$.

The idea behind the above definition of admissible portals is simple: Given an admissible path A, an admissible sequence P of portals defines how a word $w \in L$ moves from one strongly connected component of A to the next one, starting from the initial state q_1 and ending in an accepting state. The pair (p_j^1,p_j^2) is the first and last states that are traversed in C_{i_j}.

Now, given an admissible path A and a corresponding admissible sequence P of portals, we say that an increasing sequence of integers $\Pi=(n_j)_{j=1}^{t+1}$ forms an *admissible partition* with respect to (A,P) if the following hold.

1. $n_1=0$.
2. For every $1 \leq j \leq t$, there exists a path from p_j^1 to p_j^2 in C_{i_j} of length $n_{j+1}-n_j-1$.
3. $n_{t+1}=n+1$.

The meaning of the partition $\Pi=(n_j)_{j=1}^{t+1}$ is as follows. If $w \in L$ and w traverses M

in accordance with (A, P), then, for each $1 \le j \le t$, the value of n_j indicates that w arrives to component C_{i_j} for the first time after n_j bits. For convenience we also set $n_{t+1} = n + 1$. Thus, for each $1 \le j \le t$, the word w stays in C_{i_j} in the interval $[n_j + 1, n_{j+1} - 1]$. Note that it is possible in principle that for a given admissible path A and a corresponding admissible sequence of portals P there is no corresponding admissible partition Π. (This could happen if the path A and the set of portals P correspond to no word of length n.)

A triplet (A, P, Π), where A is an admissible path, P is a corresponding admissible sequence of portals, and Π is a corresponding admissible partition, will be called an *admissible triplet*. It is clear from the definition of an admissible triplet that a word $w \in L$ traverses G in accordance with a scenario suggested by one of the admissible triplets. Therefore, in order to get convinced that $w \notin L$, it is enough to check that w does not fit any admissible triplet.

Fix an admissible triplet (A, P, Π), where $A = (C_{i_1}, \dots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, and $\Pi = (n_j)_{j=1}^{t+1}$. For all $1 \le j \le t$, we define a language L_j that contains all words that traverse in M from p_j^1 to p_j^2. This is done formally by defining an automaton M_j as follows. The set of states of M_j is obtained by adding to C_{i_j} a new state f_j. The initial state of M_j and its unique accepting state are p_j^1 and p_j^2, respectively. For each $q \in C_{i_j}$ and $\alpha \in \{0, 1\}$, if $\delta_M(q, \alpha) \in C_{i_j}$, we set $\delta_{M_j}(q, \alpha) = \delta_M(q, \alpha)$. Otherwise, $\delta_{M_j}(q, \alpha) = f_j$. We also define $\delta_{M_j}(f_j, 0) = \delta_{M_j}(f_j, 1) = f_j$. Namely, in M_j all transitions within C_{i_j} remain the same. All transitions going to other components now go to f_j, which has a loop to itself. Thus, M_j is essentially strongly connected, as in Definition 2.5, with $D = \{f_j\}$. Then L_j is the language accepted by M_j.

Given the fixed admissible triplet (A, P, Π) and a word w of length $|w| = n$, we define t subwords of it, $w^1, \dots, w^t$, by setting $w^j = w[n_j + 1] \dots w[n_{j+1} - 1]$, where $1 \le j \le t$. Note that $|w^j| = n_{j+1} - n_j - 1$. Namely, if w were to pass through M according to the partition Π, then the substring w_j would correspond to the portion of the traversal path of w that lies within the component C_{i_j}.

LEMMA 2.7. *Let (A, P, Π) be an admissible triplet , where $A = (C_{i_1}, \dots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, and $\Pi = (n_j)_{j=1}^{t+1}$. Let w be a word of length n satisfying $dist(w, L) \ge \epsilon n$. Define languages $(L_j)_{j=1}^t$ and words $(w_j)_{j=1}^t$ as described above. Then there exists an index j, $1 \le j \le t$, for which $dist(w^j, L_j) \ge \frac{\epsilon n - k}{k}$.*

Proof. Assume this is not the case. Let $\Pi = (n_j)_{j=1}^{t+1}$ be the partition, and recall that $t \le k$. For every $1 \le j \le t$, if $n_{j+1} - n_j \ge 2$, let $w^{j*} \in L_j$ be a word of length $n_{j+1} - n_j - 1$ for which $dist(w^j, w^{j*}) < (\epsilon n - k)/k$. If $n_{j+1} - n_j = 1$, we set $w^{j*} = \gamma$ (the empty word). Also, for $1 \le j \le t - 1$, choose $\alpha_j \in \{0, 1\}$ so that $\delta_M(p_j^2, \alpha_j) = p_{j+1}^1$. Then by construction the word $w^* = w^{1*}\alpha_1 w^{2*} \dots \alpha_{t-1} w^{t*}$ belongs to L and $dist(w, w^*) \le t - 1 + \sum_{j=1}^t dist(w^j, w^{j*}) \le t - 1 + t(\epsilon n - k)/k < \epsilon n$, which is a contradiction. ∎

Now we present a key idea of the proof. Ideally, we would like to test whether an input word w of length n fits any admissible triplet. In the positive case, i.e., when $w \in L_M$, the traversal path of w in M defines naturally an admissible triplet which w will obviously fit. In the negative case, i.e., when $dist(w, L) \ge \epsilon n$, Lemma 2.7 implies that for every admissible triplet (A, P, Π), at least one of the subwords w^j is very far from the corresponding language L_j. Then by Lemma 2.6 w^j contains many short infeasible runs, and thus sampling a small number of random runs will catch one of them with high probability. However, the problem is that the total number of admissible triplets clearly depends on n, which makes the task of applying directly

the union bound on the probability of not catching an infeasible run impossible.

We circumvent this difficulty in the following way. We place evenly in $1, \ldots, n$ a bounded number (depending only on ϵ and the parameters of M) of *transition intervals* T_s of a bounded length and postulate that a transition between components of $\mathcal{C}(G)$ should happen inside these transition intervals. Then we show that if $w \in L$, it can be modified slightly to meet this restriction, whereas if $dist(w, L) \geq \epsilon n$, for any choice of such an admissible triplet, w is far from fitting it. As the number of admissible triplets under consideration is bounded by a function of ϵ only, we can apply the union bound to estimate the probability of failure.

Recall that $m = \max_j(m(C_j))$, $l = lcm(\{g(G[C_j])\})$, where j runs over all truly connected components of G, corresponding to vertices of $\mathcal{C}(G)$. Let $S = 129km\log(1/\epsilon)/\epsilon$. We place S transition intervals $\{T_s = [a_s, b_s]\}_{s=1}^{S}$ evenly in $[n]$, where the length of each transition interval T_s is $|T_s| = (k-1)(l+m)$. For $1 \leq i \leq \log(8km/\epsilon)$ define $r_i = \frac{2^{8-i}k^2 m \log^2(\frac{1}{\epsilon})}{\epsilon}$.

ALGORITHM.

Input: a word w of length $|w| = n$;

1. For each $1 \leq i \leq \log(8km/\epsilon)$ choose r_i random runs in w of length 2^{i+1} each;
2. For each admissible triplet (A, P, Π) with $A = (C_{i_1}, \ldots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$ such that for all $2 \leq j \leq t$ one has $n_j \in T_s$ for some $1 \leq s \leq S$, do the following:
   - Form the automata M_j, $1 \leq j \leq t$, as described above.
   - Discard those chosen runs which end or begin at place p for which $|p - n_j| \leq \epsilon n/(128km\log(1/\epsilon))$. Namely, those runs which have one of their ends closer than $\epsilon n/(128km\log(1/\epsilon))$ from some $n_j \in \Pi$.
   - For each remaining run R, if R falls between n_j and n_{j+1}, check whether it is feasible for the automaton M_j starting at $b - n_j + 1$, where b is the first coordinate of R in w. Namely, $b - n_j + 1$ is the place where R starts relative to n_j, which is the place w "enters" M_j.
3. If for some admissible triplet all checked runs turned out to be feasible, output "YES." Otherwise (i.e., in the case where for all admissible triplets at least one infeasible run has been found) output "NO."

LEMMA 2.8. *If $dist(w, L) \geq \epsilon n$, then the above algorithm outputs "NO" with probability at least $3/4$. If $w \in L$, then the algorithm always outputs "YES."*

Proof. The proof contains two independent parts. In the first we consider the case of an input w with $dist(w, L) \geq \epsilon n$, on which the algorithm should answer "NO" (with high probability). The other part treats the case where $w \in L$, for which the algorithm should answer "YES."

Let us first assume that $dist(w, L) \geq \epsilon n$. The number of admissible triplets (A, P, Π) for which all partition points fall into the union of transition intervals $\bigcup_{s=1}^{S} T_s$ can be estimated from above by

$$2^k |V|^{2k} \left(S(k-1)(l+m)\right)^{k-1}.$$

(First choose an admissible path in $\mathcal{C}(G)$; the number of admissible paths is at most 2^k as any subset of vertices of $\mathcal{C}(G)$ defines at most one path spanning it. Then choose portals; the total number of chosen portals is at most $2k$, and therefore there are at most $|V|^{2k}$ possible choices for portals; then for a fixed pair (A, P) there are at most $S|T_s|$ choices for each n_j, where $2 \leq j \leq t$ and $t \leq k$.) For ϵ satisfying condition (*) and S as above, this expression is at most $(1/\epsilon)^{2k}$. Thus we need to check at most $(1/\epsilon)^{2k}$ admissible triplets.

Let (A, P, Π) be an admissible triplet satisfying the restriction formulated in step 2 of the above algorithm. Write $A = (C_{i_1}, \ldots, C_{i_t})$, $P = (p_j^1, p_j^2)_{j=1}^t$, $\Pi = (n_j)_{j=1}^{t+1}$. Then the triplet defines automata $(M_j)_{j=1}^t$ and languages $(L_j)_{j=1}^t$ as described before. By Lemma 2.7 for some $1 \le j \le t$ one has $dist(w^j, L_j) \ge (\epsilon n - k)/k > \epsilon n/(2k)$. Then by Lemma 2.6 there exists an i, $1 \le i \le \log(8km/\epsilon)$ so that w^j contains at least $2^{i-4}\epsilon n/(2km\log(8km/\epsilon)) \ge 2^{i-6}\epsilon n/(km\log(1/\epsilon))$ infeasible runs of length 2^{i+1}. At most $\alpha = \epsilon n/(128km\log(1/\epsilon))$ of them may touch the last α bits of the interval $[n_j, n_{j+1} - 1]$, and at most α of them may touch the first α bits of this interval. Hence there are at least $2^{i-6}\epsilon n/(km\log(1/\epsilon)) - 2\alpha \ge (2^{i-7}\epsilon n)/(km\log(1/\epsilon))$ of them that touch neither the first nor the last $\epsilon n/(128km\log(1/\epsilon))$ bits of the interval $[n_j, n_{j+1} - 1]$. Obviously, if a random sample contains one of these infeasible runs, then it provides a certificate for the fact that w does not fit this admissible triplet. A random sample of r_i runs of length 2^{i+1} misses all of these infeasible runs with probability at most

$$\left(1 - \frac{1}{n}\frac{2^{i-7}\epsilon n}{km\log(\frac{1}{\epsilon})}\right)^{r_i} < e^{-2k\log(\frac{1}{\epsilon})} < \frac{1}{4}\left(\frac{1}{\epsilon}\right)^{-2k}.$$

Thus by the union bound we conclude that in this case a random sample does not contain a "witness" for each feasible triplet with probability at most 1/4. This completes the proof for the case of $dist(w, L) \ge \epsilon n$.

We now address the case for which $w \in L$. We need to show that in this case the algorithm answers "YES." For this is enough to show that if $w \in L$, then there exists an admissible triplet which passes successfully the test of the above algorithm. A traversal of w in M naturally defines a triplet (A, P, Π) as follows: $A = (C_{i_1}, \ldots, C_{i_t})$, where $C_{i_1}, \ldots, C_{i_t}$ are components from $\mathcal{C}(G)$, ordered according to the order of their traversal by w; $P = (p_j^1, p_j^2)_{j=1}^t$, where p_j^1 (resp., p_j^2) is the first (resp., the last) state of C_{i_j} visited by w; $\Pi = (n_j)_{j=1}^{t+1}$, where $n_1 = 0$, $n_{t+1} = n + 1$, and for $2 \le j \le t$, n_j is set to be the first time w enters C_{i_j} while traversing M. However, this partition does not necessarily meet the requirement stated in step 2 of the algorithm. In the true traversal of w in M the transitions from C_{i_j} to $C_{i_{j+1}}$ might occur outside the transition intervals T_s. We show that the desired triplet can be obtained from the actual triplet, (A, P, Π), of w by modifying only the third component of it. This modified triplet would then correspond to a different word $w' \in L$ (which is quite close to w) that makes all the transitions inside the postulated transition intervals. In addition, we will take care that no query is made to bits in which w' differs from w. Hence, the algorithm will actually be consistent with both. This is in fact the reason for discarding the runs that are too close to some n_j in step 2 of the algorithm. Intuitively, this is done as follows. Assume n_j is not in a transition interval; then we either make the traversal in $C_{i_{j-1}}$ longer so as to end in p_{j-1}^2 in a transition interval, or we shorten the traversal in $C_{i_{j-1}}$ so as to enter a transition interval, depending on where the closest transition interval is. Formally, this is done as follows. Define a new partition $\Pi' = (n'_j)_{j=1}^{t+1}$, where $n'_1 = n_1 = 0$. For each $2 \le j \le t$ choose a transition interval T_s closest to n_j. If C_{i_j} is a truly connected component, we choose n'_j as the leftmost coordinate in T_s satisfying the following restrictions: (a) $n'_j \equiv n_j \bmod l$; (b) $n'_j - n'_{j-1} > m$. If C_{i_j} is a singleton without loops, we set $n'_j = n'_{j-1} + 1$. As $|T_s| = (k-1)(l+m)$, such an n'_j always exists. Finally, we set $n'_{t+1} = n_{t+1} = n + 1$.

Note that the obtained triplet (A, P, Π') is admissible. Indeed, for every $1 \le j \le t$ we have $n'_{j+1} - n'_j \equiv n_{j+1} - n_j \bmod l$, thus implying $n'_{j+1} - n'_j \equiv n_{j+1} - n_j \bmod g(G[C_{i_j}])$ if C_{i_j} is truly connected. As there exists a path from p_j^1 to p_j^2 in

C_{i_j} of length $n_{j+1} - n_j - 1$, there also exists a path of length $n'_{j+1} - n'_j - 1$. This implies the admissibility of Π' and hence the admissibility of (A, P, Π').

Now let R be a run of w inside $[n'_j + \epsilon n/(128km\log(1/\epsilon)), n'_{j+1} - \epsilon n/(128km\log(1/\epsilon))]$, and let b be its first coordinate. Since we placed S transition intervals $\{T_s\}$ evenly in $[n]$, we have $|n'_j - n_j| \le n/S + |T_s| = \epsilon n/(129km\log(1/\epsilon)) + (k-1)(l+m)$. Therefore, R falls also completely inside $[n_j + m, n_{j+1} - 1]$. (We remark at this point that the purpose of discarding marginal runs at step 2 of the algorithm is to achieve that each one of the remaining runs will fall completely not only within $[n'_j, n'_{j+1}]$ but also within $[n_j, n_{j+1}]$. As we will see immediately, this guarantees that R will be feasible for the corresponding automaton M_j. Without this deletion, with positive probability one of the sampled runs R may start in a place where w is in $C_{i_{j-1}}$ and end in a place where w is in C_{i_j}, thus making it impossible to attribute R to one particular automaton M_j. Therefore, with positive probability the algorithm would fail in the positive case. Discarding marginal runs allows us to get a one-sided error algorithm.)

As $w \in L$, there exists a state $q \in C_{i_j}$ so that $\delta(q, R) \in C_{i_j}$. Also, q is reachable from p^1_j (the initial state of C_{i_j}) in $b - n_j \ge m$ steps (b is the first coordinate of R). According to the choice of n'_j, we have $n'_j \equiv n_j \bmod g_j$, where g_j is the period of C_{i_j}. But then by Lemma 2.3 q is reachable from p^1_j in $b - n'_j$ ($\ge m$) steps. This shows that R is feasible for M_j, starting at $b - n'_j + 1$. Thus, if $w \in L$, the above algorithm always outputs "YES." □

Finally, the number of bits of w queried by our algorithm is at most

$$\sum_{i=1}^{\log(8km/\epsilon)} 2^{i+1} r_i = \sum_{i=1}^{\log(8km/\epsilon)} 2^{i+1} \frac{2^{8-i} k^2 m \log^2(\frac{1}{\epsilon})}{\epsilon} < \frac{2^{10} k^2 m \log^3(\frac{1}{\epsilon})}{\epsilon} .$$

We have thus proven the following theorem.

THEOREM 2.9. *For every regular language L, every integer n, and every small enough $\epsilon > 0$, there exists a one-sided error ϵ-testing algorithm for $L \cap \{0,1\}^n$, whose query complexity is $c\log^3(1/\epsilon)/\epsilon$, where the constant $c > 0$ depends only on L.*

A final note about the dependence of the complexity on the parameters is in place here. In the proof M is considered fixed, as the algorithm is tailored for a fixed given language. However, in the calculation above we have kept the dependence of the query complexity on the parameters of M explicit. One has to keep in mind, though, that the estimates hold only when condition (*) holds. In particular, we require (third item in (*)) that $1/(\epsilon\log(1/\epsilon)) = \Omega(k^2|V|^2 m(l+m))$.

Another note is about the running time of the algorithm (rather than just its query complexity). The dominating term in step 1 and the first two subsets of step 2 of the algorithm is the query complexity. In the last substeps, each run has to be checked against M_j. Each such check involves checking whether there is a word u and a word v (of suitable lengths) so that $uRv \in L$. Checking whether there are such u, v is done directly by Lemma 2.3 in case the length of u and v are longer than m, or by checking all 2^m words if one of them is shorter than m.

3. Lower bound for regular languages. In many testability questions, it is quite natural to expect a lower bound of order $1/\epsilon$ for the query complexity of testing. This is usually proven by taking a positive example of size n and perturbing it in randomly chosen ϵn places to create a negative instance which is hard to distinguish from the positive one. Regular languages are not an exception in this respect, as shown by the next proposition and its fairly simple proof.

PROPOSITION 3.1. *Let L be the regular language over the alphabet $\{0,1\}$ defined by $L = \{1^n \mid n \geq 1\}$. For any n an ϵ-test for $L \cap \{0,1\}^n$ has query complexity at least $\frac{1}{3\epsilon}$.*

Proof. Our proof is based on the following reformulation of the renowned principle of Yao [14], saying that if there exists a probability distribution on the union Ω of positive and negative examples such that any *deterministic* testing algorithm of query complexity d is correct with probability less than $2/3$ for an input randomly chosen from Ω according to this distribution, then d is a lower bound on the query complexity of any randomized testing algorithm.

Define a distribution on the set of positive and negative instances of length n as follows. The word 1^n gets probability $1/2$. Next we partition the index set $[1,n]$ into $t = 1/\epsilon$ parts $I_1, \ldots, I_t$, each of size ϵn, and for each $1 \leq i \leq t$ we give probability $1/(2t)$ to the vector y_i created from 1^n by flipping all bits in I_i from 1 to 0. Note that $dist(y_i, L) = \epsilon n$; hence all y_i are negative instances. Now we apply the above-mentioned principle of Yao. Let A be a deterministic ϵ-testing algorithm with query complexity d. If A is incorrect on the word 1^n, then it is already incorrect with probability at least $1/2$. Otherwise, it should accept the input if all d tested bits equal to 1. Therefore, it accepts as well at least $t - d$ of the inputs y_i. This shows that A gives an incorrect answer with probability at least $(t-d)/(2t) < 1/3$, implying that $d > t/3$. □

The main idea of the proof of the above proposition can be used to get an $\Omega(1/\epsilon)$ lower bound on the query complexity of testing any nontrivial regular language, with a natural definition of nontrivial. This is proven in the next proposition. A somewhat paradoxical feature of its proof is that our main positive result (Theorem 2.9) and its proof are used here to get a negative result.

For a language L let $L_n = L \cap \{0,1\}^n$.

DEFINITION 3.2. *A language L is nontrivial if there exists a constant $0 < \epsilon_0 < 1$, so that for infinitely many values of n the set L_n is nonempty, and there exists a word $w \in \{0,1\}^n$ so that $dist(w, L_n) \geq \epsilon_0 n$.*

PROPOSITION 3.3. *Let L be a nontrivial regular language. Then for all sufficiently small $\epsilon > 0$, any ϵ-testing algorithm for L requires $\Omega(1/\epsilon)$ queries.*

Proof. The proof here is essentially a generalization of the proof of Proposition 3.1. We thus present it in a somewhat abridged form.

Let n be large enough. Assume $L_n \neq \emptyset$, and $w \in \{0,1\}^n$ is such that $dist(w, L_n) \geq \epsilon_0 n$. We may clearly assume that the constant ϵ_0 is as small as needed for our purposes. Our main result, Theorem 2.9, and its proof imply that with probability at least $2/3$, a random choice of a set of runs, built as described at step 1 of the testing algorithm of Theorem 2.9, and having total length $\tilde{O}(1/\epsilon_0)$, will cause the algorithm to reject w. As we have noticed, the testing algorithm has a one-sided error, i.e., it always accepts a word from L. Thus, if we choose a random set of runs as above, it will cause a rejection of w with probability $2/3$ and it will not coincide with any word $u \in L_n$ (for otherwise, it would reject u too).

Each such random set of runs is just a random set of intervals in $\{1, \ldots, n\}$ (of length as defined in step 1 of the testing algorithm) of total length bounded by $\tilde{O}(1/\epsilon_0)$. Notice that two such random sets intersect with probability $\tilde{O}(1/(\epsilon_0^2 n))$. Therefore, if we choose $\tilde{\Omega}(\epsilon^2 n)$ such subsets at random, then we expect that $\tilde{O}(\epsilon_0^2 n)$ pairs of them will intersect, and that $2/3$ of the members will reject w. This implies that there exists a family $\mathcal{S}$ of $\tilde{\Omega}(\epsilon_0^2)n$ pairwise disjoint sets of runs so that for each member of $\mathcal{S}$, no word of L_n coincides with w on this set. Fix now ϵ_0 and let $\epsilon > 0$

be small enough compared to ϵ_0. We partition the family $\mathcal{S}$ into $t = (c/\epsilon)$ subfamilies $\mathcal{S}_1, \ldots, \mathcal{S}_t$, each of cardinality ϵn, where the constant c depends on ϵ_0 only and is thus independent of ϵ. Let u be a word in L_n. For each $1 \leq i \leq t$, the word w_i is obtained from u by changing the bits of u, corresponding to $\mathcal{S}_i$, to those from w. It follows then that $dist(w_i, L_n) \geq \epsilon n$. Indeed, to transform w_i into a word in L_n, at least one bit has to be changed in every member of $\mathcal{S}_i$.

Now, as in the proof of Proposition 3.1, we define a probability distribution on the union of positive and negative examples. The word u gets probability $1/2$, and each one of the t words $w_1, \ldots, w_t$ gets probability $1/(2t)$. A simple argument, essentially identical to that in the proof of Proposition 3.1, shows that any deterministic algorithm needs to query at least $\Omega(t) = \Omega(1/\epsilon)$ bits of the input word to be successful with probability at least $2/3$ on the defined probability distribution. Applying Yao's principle, we get the desired result. □

4. Testability of context-free languages. Having essentially completed the analysis of testability of regular languages, it is quite natural to try to go one step further and to address testability of the much more complex class of context-free languages (see, e.g., [8] for background information). It turns out that the general situation changes drastically here as compared to the case of regular languages. We show that there exist quite simple context-free languages which are *not* ϵ-testable. Then we turn our attention to one particular family of context-free languages—the so-called Dyck languages. We prove that the first language in this family, D_1, is testable in time polynomial in $1/\epsilon$, while all other languages in the family are already nontestable. All relevant definitions and proofs follow.

4.1. Some context-free languages are nontestable. As we have already mentioned, not all context-free languages are testable. This is proven in the following proposition.

THEOREM 4.1. *Any ϵ-testing algorithm for the context-free language $L = \{vv^R uu^R\}$, where w^R denotes the reversal of a word w, requires $\Omega(\sqrt{n})$ queries in order to have an error of at most $1/3$.*

Proof. Let n be divisible by 6. We again define a distribution D on the union of positive and negative inputs in the following way. A negative instance is chosen uniformly at random from among all negative instances (i.e., those words $w \in \{0,1\}^n$ which are at distance at least ϵn from L). We refer to this distribution as N. Positive instances are generated according to a distribution P defined as follows. We pick uniformly at random an integer k in the interval $[n/6+1, n/3]$ and then select a positive example uniformly among words $vv^R uu^R$ with $|v| = k$. Finally, the distribution D on all inputs is defined as follows: with probability $1/2$ we choose a positive input according to P and with probability $1/2$ we choose a negative input according to N. We note that a positive instance is actually a pair (k, w). (The same word w may be generated using different k's.)

We use the above-mentioned Yao's principle again. Let A be a deterministic ϵ-testing algorithm for L. We show that for any such A, if its maximum number of queries is $d = o(\sqrt{n})$, then its expected error with respect to D is at least $\frac{1}{2} - o(1)$. Indeed, let A be such an algorithm. We can view A as a binary decision tree, where each node represents a query to a certain place, and the two outgoing edges, labeled with 0 or 1, represent possible answers. Each leaf of A represents the end of a possible computation and is labeled "positive" or "negative" according to the decision of the algorithm. Tracing the path from the root to a node of A, we can associate with each node t of A a pair (Q_t, f_t), where $Q_t \subseteq \{1, \ldots, n\}$ is a set of queries to the input

word, and $f_t : Q_t \to \{0,1\}$ is a vector of answers received by the algorithm. We may obviously assume that A is a full binary tree of height d and has thus 2^d leaves. Then $|Q_t| = d$ for each leaf t of A.

We will use the following notation. For a subset $Q \subseteq \{1,\dots,n\}$ and a function $f : Q \to \{0,1\}$, let

$$E^-(Q,f) = \{w \in \{0,1\}^n, dist(w,L) \geq \epsilon n,\ w \text{ coincides with } f \text{ on } Q\},$$
$$E^+(Q,f) = \{w \in \{0,1\}^n \cap L :\ w \text{ coincides with } f \text{ on } Q\},$$

i.e., $E^-(Q,f)$ ($E^+(Q,f)$) is the set of all negative (resp., positive) instances of length n consistent with the pair (Q,f). Also, if D is a probability distribution on the set of binary strings of length n and $E \subseteq \{0,1\}^n$ is a subset, we define $\Pr^D[E] = \sum_{w \in E} \Pr^D[w]$.

Let T_1 be the set of all leaves of A labeled "positive," and let T_0 be the set of all leaves of T labeled "negative." Then the total error of the algorithm A on the distribution D is

$$\sum_{t \in T_1} \Pr^D[E^-(Q_t, f_t)] + \sum_{t \in T_0} \Pr^D[E^+(Q_t, f_t)] \ .$$

The theorem follows from the following two claims.

CLAIM 4.2. *For every subset $Q \subset \{1,\dots,n\}$ of cardinality $|Q| = o(n)$ and for every function $f : Q \to \{0,1\}$,*

$$\Pr^D[E^-(Q,f)] \geq \left(\frac{1}{2} - o(1)\right) 2^{-|Q|} \ .$$

CLAIM 4.3. *For every subset $Q \subset \{1,\dots,n\}$ of cardinality $|Q| = o(\sqrt{n})$ and for every function $f : Q \to \{0,1\}$,*

$$\Pr^D[E^+(Q,f)] \geq \left(\frac{1}{2} - o(1)\right) 2^{-|Q|} \ .$$

Based on Claims 4.2 and 4.3, we can estimate the error of the algorithm A by

$$\sum_{t \in T_1} \Pr^D[E^-(Q_t, f_t)] + \sum_{t \in T_0} \Pr^D[E^+(Q_t, f_t)] \geq \sum_{t \in T_1} \left(\frac{1}{2} - o(1)\right) 2^{-|Q_t|}$$
$$+ \sum_{t \in T_0} \left(\frac{1}{2} - o(1)\right) 2^{-|Q_t|} = \left(\frac{1}{2} - o(1)\right) \frac{|T_1| + |T_0|}{2^d} \geq \frac{1}{2} - o(1) \ .$$

The theorem follows. □

We now present the proofs of Claims 4.2 and 4.3.

Proof of Claim 4.2. Notice first that L has at most $2^{n/2} n/2$ words of length n. (First choose a word of length $n/2$, and then cut it into two parts v and u, thus getting a word $w = vv^R uu^R \in L$.) Therefore, the number of words of length n at a distance less than ϵn from L is at most $|L \cap \{0,1\}^n| \sum_{i=0}^{\epsilon n} \binom{n}{i} \leq 2^{n/2 + 2\epsilon \log(1/\epsilon) n}$. We get

$$|E^-(Q,f)| \geq |\{w \in \{0,1\}^n : w(Q) = f\}| - |\{w \in \{0,1\}^n : dist(w,L) < \epsilon n\}|$$
$$\geq 2^{n-|Q|} - 2^{n/2 + 2\epsilon \log(1/\epsilon) n} = (1 - o(1)) 2^{n-|Q|} \ .$$

It follows then from the definition of D that

$$\Pr^D[E^-(Q,f)] = \frac{1}{2} \Pr^N[E^-(Q,f)] \geq \frac{1}{2} \frac{|E^-(Q,f)|}{2^n} \geq \left(\frac{1}{2} - o(1)\right) 2^{-|Q|} \ . \qquad \square$$

Proof of Claim 4.3. It follows from the definition of the distribution D that, for a word $w \in L \cap \{0,1\}^n$,

$$\Pr^D(w) = \frac{1}{2}\Pr^P(w) = \frac{1}{2}\,\frac{|\{w = vv^Ruu^R : \ |v| = k, \frac{n}{6}+1 \le k \le \frac{n}{3}\}|}{\frac{n}{6}2^{n/2}} \ .$$

Recall that $E^+(Q,f)$ is the set of words in L which is consistent with f on the set of queries Q. Hence,

$$\Pr^D[E^+(Q,f)] = \frac{1}{\frac{n}{6}2^{n/2+1}} \sum_{k=n/6+1}^{n/3} |\{w \in \{0,1\}^n : w(Q) = f, w = vv^Ruu^R, |v| = k\}| \ .$$

Now observe that for each of the $\binom{d}{2}$ pairs of places in Q there are at most two choices of k for which the pair is symmetric with respect to k or to $n/2+k$. This implies that for $n/6 - 2\binom{d}{2} = (1-o(1))n/6$ choices of k, the set Q does not contain a pair symmetric with respect to k or $n/2+k$. For each such k,

$$|\{w \in \{0,1\}^n : w(Q) = f, w = vv^Ruu^R, |v| = k\}| = 2^{n/2-|Q|} \ .$$

Therefore,

$$\Pr^D[E^+(Q,f)] \ge \frac{(1-o(1))\frac{n}{6}2^{n/2-|Q|}}{\frac{n}{6}2^{n/2+1}} = \left(\frac{1}{2} - o(1)\right)2^{-|Q|} \ . \qquad \square$$

As a concluding remark to this subsection we would like to note that in the next subsection (Theorem 4.10) we will give another proof to the fact that not all context-free languages are testable by showing the nontestability of the Dyck language D_2. However, we preferred to give Theorem 4.1 as well due to the following reasons. First, the language discussed in Theorem 4.1 is simpler and more natural than the Dyck language D_2. Second, the lower bound of Theorem 4.1 is better than that of Theorem 4.10. The proofs of these two theorems have many common points, so the reader may view Theorem 4.1 as a "warm-up" for Theorem 4.10.

4.2. Testability of the Dyck languages. It would be extremely nice to determine exactly which context-free languages are testable. At present we seem to be very far from fulfilling this task. However, we are able to solve this question completely for one family of context-free languages—the so-called Dyck languages.

For an integer $n \ge 1$, the *Dyck language of order* n, denoted by D_n, is the language over the alphabet of $2n$ symbols $\{a_1, b_1, \ldots, a_n, b_n\}$, grouped into n ordered pairs $(a_1,b_1), \ldots, (a_n,b_n)$. The language D_n is defined by the following productions:

1. $S \to a_i S b_i$ for $i = 1, \ldots, n$,
2. $S \to SS$, and
3. $S \to \gamma$,

where γ denotes the empty word. Though the words of D_n are not binary according to the above definition, we can easily encode them and the grammar describing them using only 0's and 1's. Thus we may still assume that we are in the framework of languages over the binary alphabet. We can interpret D_n as the language with n distinct pairs of brackets, where a word w belongs to D_n iff it forms a balanced bracket expression. The most basic and well-known language in this family is D_1, where we have only one pair of brackets. Dyck languages play an important role in the theory

of context-free languages (see, e.g., [4] for a relevant discussion), and therefore the task of exploring their testability is interesting.

Our first goal in this subsection is to show that the language D_1 is testable. Let us introduce a suitable notation. First, for the sake of simplicity we denote the brackets a_1, b_1 of D_1 by $0, 1$, respectively. Assume that n is a large enough even number. (Obviously, for odd n we have $D_1 \cap \{0,1\}^n = \emptyset$, and thus there is nothing to test in this case.) Let w be a binary word of length n. For $1 \le i \le n$, we denote by $x(w,i)$ the number of 0's in the first i positions of w. Also, $y(w,i)$ stands for the number of 1's in the first i positions of w. We have the following claims.

CLAIM 4.4. *The word w belongs to D_1 iff the following two conditions hold:* (a) $x(w,i) \ge y(w,i)$ *for every* $1 \le i \le n$; (b) $x(w,n) = y(w,n)$.

Proof. The proof follows easily from the definition of D_1—for example, by induction on the length of w. We omit a detailed proof. □

CLAIM 4.5. *If w satisfies* (a) $y(w,i) - x(w,i) \le s_1$ *for every* $1 \le i \le n$, *and* (b) $x(w,n) - y(w,n) \le s_2$, *then* $dist(w, D_1) \le s_1 + s_2/2 + 1$.

Proof. Observe first that by Claim 4.4 a word w is in D_1 iff we can partition its letters into pairwise disjoint pairs, so that the left letter in each pair is a 0, and the right letter is a 1. Consider the bipartite graph, whose two classes of vertices are the set of indices i for which $w[i] = 0$ and the set of indices i for which $w[i] = 1$, respectively, where each i with $w[i] = 1$ is connected to all $1 \le j < i$ for which $w[j] = 0$. By assumption (a) and the defect form of Hall's theorem, this graph contains a matching of size at least $y(w,n) - s_1$. By assumption (b), $y(w,n) \ge n/2 - s_2/2$. Therefore, there are at least $n/2 - s_2/2 - s_1$ disjoint pairs of letters in w, where in each pair there is a 0 on the left and a 1 on the right. Let us pair the remaining elements of w arbitrarily, where all pairs but at most one consist of either two 0's or two 1's. By changing, now, when needed, the left entry of each such pair to 0 and its right entry to 1 we obtain a word in D_1, and the total number of changes performed is at most $(s_2 + 2s_1 - 2)/2 + 2 = s_1 + s_2/2 + 1$, completing the proof. □

CLAIM 4.6. (a) *If for some $1 \le i \le n$ one has $y(w,i) - x(w,i) \ge s$, then $dist(w, D_1) \ge s/2$.* (b) *If $x(w,n) - y(w,n) \ge s$, then $dist(w, D_1) \ge s/2$.*

Proof. The proof follows immediately from Claim 4.4. □

We conclude from the above three claims that a word w is far from D_1 iff for some coordinate i it deviates significantly from the necessary and sufficient conditions provided by Claim 4.5. This observation is used in the analysis of an algorithm for testing D_1, proposed below.

Set

$$d = \frac{C \log\left(\frac{1}{\epsilon}\right)}{\epsilon^2},$$

$$\Delta = \frac{C \log\left(\frac{1}{\epsilon}\right)}{8\epsilon},$$

where $C > 0$ is a sufficiently large constant, whose value will be chosen later, and assume that d is an even integer. In what follows we omit all floor and ceiling signs to simplify the presentation.

ALGORITHM.

Input: a word w of length $|w| = n$;

1. Choose a sample S of bits in the following way. For each bit of w, independently and with probability $p = d/n$ choose it to be in S. Then, if S contains more than $d + \Delta/4$ bits, answer "YES" without querying any bit. Else,

2. If $dist(S, D_1 \cap \{0,1\}^{d'}) < \Delta$, where $d' = |S|$, output "YES"; otherwise, output "NO."

LEMMA 4.7. *The above algorithm outputs a correct answer with probability at least* 2/3.

Proof. As we have already mentioned, we set

$$p = \frac{d}{n} = \frac{C \log\left(\frac{1}{\epsilon}\right)}{\epsilon^2 n} .$$

The proof contains two independent parts; in the first we prove that the algorithm is correct (with probability 2/3) for $w \in D_1$, and in the second part we prove that the algorithm has a bounded error for words w for which $dist(w, D_1) \geq \epsilon n$.

Consider first the positive case $w \in D_1$. Set $t = C/\epsilon$, and assume for simplicity that t as well as n/t are integers. For $1 \leq j \leq t$, let X_j be the number of 0's in S, sampled from the interval $[1, nj/t]$. Let also Y_j denote the number of 1's in S, sampled from the same interval. Both X_j and Y_j are binomial random variables with parameters $x(w, nj/t)$ and p, and $y(w, nj/t)$ and p, respectively. As $w \in D_1$, we get by Claim 4.4 that $x(w, nj/t) \geq y(w, nj/t)$, implying $EX_j \geq EY_j$. Applying standard bounds on the tails of binomial distribution, we obtain

$$\Pr\left[Y_j \geq X_j + \frac{\Delta}{2}\right] \leq \Pr\left[X_j \leq EX_j - \frac{\Delta}{4}\right] + \Pr\left[Y_j \geq EY_j + \frac{\Delta}{4}\right] \leq 2^{-\Omega(\Delta^2/np)}$$

$$= 2^{-\Omega(C \log(1/\epsilon))}. \tag{4.1}$$

For $1 \leq j \leq t-1$, set $Z_j = Y_{j+1} - Y_j$. Note that $EZ_j \leq np/t$. Using similar argumentation as above, we get

$$\Pr\left[Z_j \geq \frac{2np}{t}\right] \leq 2^{-\Omega(np/t)} = 2^{-\Omega(\log(1/\epsilon)/\epsilon)}. \tag{4.2}$$

As $w \in D_1$, we have by Claim 4.4 $x(w, n) = y(w, n) = n/2$. Hence

$$\Pr\left[X_t \geq \frac{np}{2} + \frac{\Delta}{8}\right] \leq 2^{-\Omega(\Delta^2/np)} = 2^{-\Omega(C \log(1/\epsilon))}. \tag{4.3}$$

Finally, we have the following estimate on the distribution of the sample size $|S|$:

$$\Pr\left[||S| - np| \geq \frac{\Delta}{4}\right] \leq 2^{-\Omega(\Delta^2/np)} = 2^{-\Omega(C \log(1/\epsilon))}. \tag{4.4}$$

Choosing C large enough and recalling the definition of t, we derive from (4.1)–(4.4) that with probability at least 2/3 the following events hold simultaneously:

1. $\max_{1 \leq j \leq t}(Y_j - X_j) \leq \frac{\Delta}{2}$,
2. $\max_{1 \leq j \leq t} Z_j \leq \frac{2np}{t}$,
3. $X_t \leq \frac{np}{2} + \frac{\Delta}{8}$,
4. $|S| \geq np - \frac{\Delta}{4}$.

Assume that the above four conditions are satisfied. Then we claim that $dist(S, D_1) < \Delta$. Indeed, the first two conditions guarantee that for all $1 \leq i \leq |S|$ we have $y(S, i) - x(S, i) \leq \Delta/2 + 2np/t \leq 2\Delta/3$. The last two conditions provide $x(S, |S|) - y(S, |S|) = X_t - Y_t = 2X_t - |S| \leq \Delta/2$. Therefore, by Claim 4.5 $dist(S, D_1) < \Delta$.

Thus, if $w \in D_1$, our algorithm will accept w with probability at least $2/3$, as required. This ends the first part of the proof.

Let us now consider the negative case. Assume that $dist(w, D_1 \cap \{0,1\}^n) \geq \epsilon n$. By Claim 4.5 we have then that at least one of the following two conditions holds: (a) there exists an index $1 \leq i \leq n$ for which $y(w,i) - x(w,i) \geq \epsilon n/2$; (b) $x(w,n) - y(w,n) \geq \epsilon n/2$. In the former case, let X, Y be the number of 0's, 1's, respectively, of S, sampled from the interval $[1,i]$. Let also k be the number of elements from $[1,i]$ chosen to S. Then $X = x(S,k)$, $Y = y(S,k)$. Both X and Y are binomially distributed with parameters $x(w,i)$ and p, and $y(w,i)$ and p, respectively. It follows from the definition of i that $EY - EX \geq \epsilon np/2$. But then we have

$$\begin{aligned}\Pr[y(S,k) - x(S,k) \leq 2\Delta] &= \Pr[Y - X \leq 2\Delta] \\ &\leq \Pr\left[X \geq EX + \left(\frac{\epsilon np}{4} - \Delta\right)\right] \\ &\quad + \Pr\left[Y \leq EY - \left(\frac{\epsilon np}{4} - \Delta\right)\right] \\ &= 2^{-\Omega((\epsilon np/4 - \Delta)^2/(np))}.\end{aligned}$$

Choosing the constant C to be sufficiently large and recalling the definitions of p and Δ, we see that the above probability is at most $1/6$. But if $y(S,k) - x(S,k) \geq 2\Delta$, it follows from Claim 4.6 that $dist(S, D_1) \geq \Delta$.

If $x(w,n) - y(w,n) \geq \epsilon n/2$, we obtain, using similar arguments,

$$\Pr[x(S,|S|) - y(S,|S|) \leq 2\Delta] = 2^{-\Omega((\epsilon np/4 - \Delta)^2/(np))}.$$

The above probability can be made at most $1/6$ by the choice of C. But if $x(S,|S|) - y(S,|S|) \geq 2\Delta$, it follows from Claim 4.6 that $dist(S, D_1) \geq \Delta$. Thus in both cases we obtain that our algorithm accepts w with probability at most $1/6$. In addition, the algorithm may accept w (in each of the cases) when $|S| > d + \Delta/4$ (first item in the algorithm). However, by (4.4) this may be bounded by $1/6$ (choosing C as in the first part). Hence the algorithm rejects w with probability at least $2/3$. This completes the proof of Lemma 4.7. □

By Lemma 4.7 we have the following result about the testability of the Dyck language D_1.

THEOREM 4.8. *For every integer n and every small enough $\epsilon > 0$, there exists an ϵ-testing algorithm for $D_1 \cap \{0,1\}^n$, whose query complexity is $C \log(1/\epsilon)/\epsilon^2$ for some absolute constant $C > 0$.*

The reader has possibly noticed one significant difference between the algorithm of section 2 for testing regular languages and our algorithm for testing D_1. While the algorithm for testing regular languages has a one-sided error, the algorithm of this section has a two-sided error. This is not a coincidence. We can show that there is *no* one-sided error algorithm for testing membership in D_1, whose number of queries is bounded by a function of ϵ only. Indeed, assume that A is a one-sided error algorithm for testing D_1. Consider its execution on the input word $u = 0^{n/2+\epsilon n}1^{n/2-\epsilon n}$. It is easy to see that $dist(u, D_1) \geq \epsilon n$. Therefore, A must reject u with probability at least $2/3$. Fix any sequence of coin tosses which makes A reject u, and denote by Q the corresponding set of queried bits of u. We claim that if $|Q \cap [1, n/2 + \epsilon n]| \leq n/2 - \epsilon n$, then there exists a word w of length n from D_1 for which $w[i] = u[i]$ for all $i \in Q$. To prove this claim, we may clearly assume that $|Q \cap [1, n/2 + \epsilon n]| = n/2 - \epsilon n$. Define w as follows. For all $i > n/2 + \epsilon n$ we set $w[i] = 1$. Now, we take the first ϵn indices i in

$[1, n/2+\epsilon n] \setminus Q$ and set $w[i] = 0$. For the last ϵn indices i in $[1, n/2+\epsilon n] \setminus Q$ we set $w[i] = 1$. Also, $w[i] = u[i]$ for all $i \in Q$. Now, w satisfies the sufficient condition for the membership in D_1, given by Claim 4.4. Indeed, at any point j in $[1, n/2+\epsilon n]$, the number of 0's in the first j bits of w is at least as large as the number of 1's. Also, for $j \geq n/2+\epsilon n$ we have $x(w, j) = n/2$ and $y(w, j) = \epsilon n + (j - n/2 - \epsilon n) = j - n/2$. Therefore, $w \in D_1$. As A is assumed to be a one-sided error algorithm, it should always accept every $w \in D_1$. But then we must have $|Q \cap [1, n/2+\epsilon n]| > n/2 - \epsilon n$, implying that A queries a number of bits linear in n. We have proven the following statement.

PROPOSITION 4.9. *Any one-sided error ϵ-test for membership in D_1 queries $\Omega(n)$ bits on words of length n.*

Our next goal is to prove that all other Dyck languages, namely, D_k for all $k \geq 2$, are nontestable. We will present a detailed proof of this statement only for $k = 2$, but this clearly implies the result for all $k \geq 3$.

For the sake of clarity of exposition we replace the symbols a_1, b_1, a_2, b_2 in the definition of D_2 by $0, 1, 2, 3$, respectively. Then D_2 is defined by the following context-free grammar: $S \to 0S1 \mid 2S3 \mid SS \mid \gamma$, where γ is the empty word. Having in mind the above-mentioned bracket interpretation of the Dyck languages, we will sometimes refer to $0, 2$ as left brackets and to $1, 3$ as right brackets. Note that we do not use an encoding of D_2 as a language over $\{0, 1\}$ but rather as a language over an alphabet of size 4. Clearly, nontestability of D_2 as defined above will imply nontestability of any binary encoding of D_2 that is obtained by a fixed binary encoding of $\{0, 1, 2, 3\}$.

THEOREM 4.10. *The language D_2 is not ϵ-testable.*

Proof. Let n be a large enough integer, divisible by 8. We denote $L_n = D_2 \cap \{0, 1, 2, 3\}^n$. Using Yao's principle, we assign a probability distribution on inputs of length n and show that any deterministic algorithm probing $d = O(1)$ bits outputs an incorrect answer with probability $0.5 \pm o(1)$. Both positive and negative words will be composed of three parts. The first is a sequence of matching 0/1 (brackets of the first kind) followed by a sequence of 0/2 (left brackets) and a sequence of 1/3 (right brackets).

Positive instances are generated according to the distribution P as follows: choose k uniformly at random in the range $n/8, \ldots, n/4$. Given k, the word of length n is $w = 0^k 1^k v$, where v is of length $n-2k$ generated by the following. For $i = 1, \ldots, (n-2k)/2$ choose $v[i]$ at random from $0, 2$, and then set $v[n-2k+1-i] = v[i]+1$.

Negative instances are chosen as follows: the process is very similar to the positive case except that we do not have the restriction on $v[n-2k+1-i] = v[i]+1$. Namely, we choose k at random in the range $n/8, \ldots, n/4$. Given k, a word of length n is $w = 0^k 1^k v$, where v is of length $n-2k$ generated by the following. For $i = 1, \ldots, (n-2k)/2$ choose $v[i]$ at random from $0, 2$, and for $i = 1, \ldots, (n-2k)/2$ choose $v[n-2k+1-i]$ at random from $1, 3$. Let us denote by N the distribution at this stage. Note that the words that are generated may be of distance less than ϵn from L_n. (In fact, some words in L_n are generated too.) Hence we further condition N on the event that the word is of distance at least ϵn from L_n.

The probability distribution over all inputs of length n is now defined by choosing with probability 1/2 a positive instance, generated as above, and with probability 1/2 a negative instance, chosen according to the above-described process. □

CLAIM 4.11. *The probability that an instance generated according to N is ϵn-close to some word in L_n is exponentially small in n.*

Proof. Fix k, and let $w = 0^k 1^k v$ be a word of length n generated by N. For such

fixed k the three parts of w are the first part of matching 0/1 of length $2k$, the second part of which is a random sequence of 0/2 of length $\frac{n-2k}{2}$ and the third part of which is a random sequence of 1/3 of length $\frac{n-2k}{2}$. Let us denote by N_1, N_2, N_3 these three disjoint sets of indices of w.

We will bound from above the number of words w of length n of the form $w = 0^k 1^k (0/2)^{\frac{n-2k}{2}} (1/3)^{\frac{n-2k}{2}}$ which are at distance at most ϵn from L_n. First, we choose the value of w on N_2, which gives $2^{\frac{n-2k}{2}}$ possibilities. Then we choose (at most) ϵn bits of w to be changed to get a word from L_n ($\binom{n}{\epsilon n}$ choices) and set those bits ($4^{\epsilon n}$ possibilities). At this point, the only part of w still to be set is its value of N_3, where we are allowed to use only right brackets $1, 3$. The word to be obtained should belong to L_n. It is easy to see that there is at most one way to complete the current word to a word in L_n using right brackets only. Hence the number of such words altogether is at most $2^{\frac{n-2k}{2}} \binom{n}{\epsilon n} 4^{\epsilon n}$. The total number of words w of the form $0^k 1^k (0/2)^{\frac{n-2k}{2}} (1/3)^{\frac{n-2k}{2}}$ is 2^{n-2k}, and each such word gets the same probability in the distribution N. Therefore, the probability that a word chosen according to N is ϵn-close to L_n can be estimated from above by

$$\sum_{k=n/8}^{n/4} \frac{8}{n} \cdot \frac{2^{\frac{n-2k}{2}} \binom{n}{\epsilon n} 4^{\epsilon n}}{2^{n-2k}} \leq \max_k (2^{O(\epsilon \log(\frac{1}{\epsilon}))n + 2\epsilon n - \frac{n}{2} + k}) \leq 2^{-n/5}$$

for small enough $\epsilon > 0$ as promised. □

CLAIM 4.12. *Let $S \subseteq [n], |S| = d$, be a fixed set of places, and let k be chosen uniformly at random in the range $n/8, \ldots, n/4$. Then S contains a pair $i < j$ symmetric with respect to $(n-2k)/2$ with probability at most $\binom{d}{2}\frac{8}{n}$.*

Proof. For each distinct pair $i, j \in S$ there is a unique k for which i, j are symmetric with respect to the above point. Hence the above probability is bounded by $\binom{d}{2}\frac{8}{n}$. □

Proof. We now return to the proof of Theorem 4.10. Let A be an algorithm for testing L_n that queries at most $d = O(1)$ queries. As $d = O(1)$, we may assume that A is nonadaptive; namely, it queries some fixed set of places S of size d (as every adaptive A can be made nonadaptive by querying ahead at most 2^d possible queries defined by two possible branchings after each adaptive query. We then look at these $2^d = O(1)$ queries as our S). For any possible set of answers $f : S \longrightarrow \{0, 1, 2, 3\}$ and an input w let f_w denote the event that w is consistent with f on S. Let $NoSym$ be the event in which S contains no symmetric pair with respect to $(n-2k)/2$. Also, let F_0 denote all these f's on which the algorithm answers "NO," and let F_1 be all these f's on which it answers "YES." Finally, denote by (w *positive*) and (w *negative*) the events that a random w is a positive instance and a negative instance, respectively.

The total error of the algorithm is

$$\begin{aligned}
&\sum_{f \in F_0} \Pr[f_w \wedge (w \text{ positive})] + \sum_{f \in F_1} \Pr[f_w \wedge (w \text{ negative})] \\
&\geq \Pr[NoSym] \Bigg(\sum_{f \in F_0} \Pr[f_w \wedge (w \text{ positive}) | NoSym] \\
&\qquad + \sum_{f \in F_1} \Pr[f_w \wedge (w \text{ negative}) | NoSym] \Bigg).
\end{aligned}$$

However, given that S contains no symmetric pairs, for a fixed f, $\Pr[f_w \wedge (w \textit{ is negative})]$ is essentially equal to $\Pr[f_w \wedge (w \textit{ is positive})]$. (These probabilities would be exactly equal if negative w would be generated according to N. Claim 4.11 asserts that N is exponentially close to the real distribution on negative instances.) Hence each is of these probabilities is $0.5 \Pr[f_w|NoSym] \pm o(1)$.

Plugging this into the sum above and using Claim 4.12, we get that the error probability is bounded from below by $\Pr(NoSym) \sum_f (0.5 \pm o(1)) \Pr[f_w|NoSym] \geq (1 - \binom{d}{2}\frac{8}{n})(0.5 \pm o(1)) \geq 0.5 - o(1)$. □

5. Concluding remarks. The main technical achievement of this paper is a proof of testability of regular languages. A possible continuation of the research is to describe other classes of testable languages and to formulate sufficient conditions for a context-free language to be testable. (Recall that in Theorem 4.1 we have shown that not all context-free languages are testable.)

One of the most natural ways to describe large classes of testable combinatorial properties is by putting some restrictions on the logical formulas that define them. In particular, we can restrict the arity of the participating relations, the number of quantifier alternations, the order of the logical expression (first order, second order), etc.

The result of the present paper is an example to this approach, since regular languages are exactly those that can be expressed in second order monadic logic with a unary predicate and an embedded linear order. Another example can be found in a sequel of this paper [1], which addresses testability of graph properties defined by sentences in first order logic with binary predicates and which complements the class of graph properties shown to be testable by Goldreich, Goldwasser, and Ron [7]. Analogous results for predicates of higher arities would be desirable to obtain, but technical difficulties arise when the arity is greater than two.

As a long-term goal we propose a systematic study of the testability of logically defined classes. Since many different types of logical frameworks are known, to find out which one is suited for this study is a challenge. Virtually all single problems that have been looked at so far have the perspective of being captured by a more general logically defined class with members that have the same testability properties.

A very different avenue is to try to develop general *combinatorial* techniques for proving lower bounds for the query complexity of testing arbitrary properties, possibly by finding analogues to the block sensitivity [12] and the Fourier analysis [11] approaches for decision tree complexity. At present we have no candidates for combinatorial conditions that would be both necessary and sufficient for ϵ-testability.

Acknowledgments. We would like to thank Oded Goldreich for helpful comments. We are also grateful to the anonymous referees for their careful reading.

REFERENCES

[1] N. Alon, E. Fischer, M. Krivelevich, and M. Szegedy, Combinatorica, 20 (2000), pp. 451–476.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and intractability of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[3] M. Blum, M. Luby, and R. Rubinfeld, *Self-testing/correcting with applications to numerical problems*, J. Comput. System Sci., 47 (1994), pp. 549–595.

[4] M. D. Davis, R. Sigal, and E. Weyuker, *Computability, Complexity and Languages: Fundamentals of Theoretical Computer Science*, Academic Press, New York, 1994.

[5] J. Dixmier, *Proof of a conjecture by Erdős and Graham concerning the problem of Frobenius*, J. Number Theory, 34 (1990), pp. 198–209.

[6] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson, *Self-testing/correcting for polynomials and for approximate functions*, in Proceedings of the 23rd ACM Symposium on Theory of Computer Science, ACM, New York, 1991, pp. 33–42.

[7] O. Goldreich, S. Goldwasser, and D. Ron, *Property testing and its connections to learning and approximation*, J. ACM, 45 (1998), pp. 653–750. Also in Proceedings of the 37th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1996, pp. 339–348.

[8] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison-Wesley, Reading, MA, 1979.

[9] M. Lewin, *A bound for a solution of a linear Diophantine problem*, J. London Math. Soc. (2), 6 (1972), pp. 61–69.

[10] R. Lipton, *New Directions in Testing*, unpublished manuscript, 1989.

[11] N. Nisan and M. Szegedy, *On the degree of Boolean functions as real polynomials*, Proceedings of the 24th ACM Symposium on Theory of Computer Science, ACM, New York, 1992, pp. 462–467.

[12] N. Nisan, *CREW PRAMs and decision trees*, in Proceedings of the 21st ACM Symposium on Theory of Computer Science, ACM, New York, 1989, pp. 327–335.

[13] R. Rubinfeld and M. Sudan, *Robust characterizations of polynomials with applications to program testing*, SIAM J. Comput., 25 (1996), pp. 252–271.

[14] A. C. Yao, *Probabilistic computation, towards a unified measure of complexity*, in Proceedings of the 18th IEEE Symposium on Foundations of Computer Science, IEEE Computer Society, Los Alamitos, CA, 1977, pp. 222–227.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1863–1920

THE APPROXIMABILITY OF CONSTRAINT SATISFACTION PROBLEMS*

SANJEEV KHANNA†, MADHU SUDAN‡, LUCA TREVISAN§, AND DAVID P. WILLIAMSON¶

Abstract. We study optimization problems that may be expressed as "Boolean constraint satisfaction problems." An instance of a Boolean constraint satisfaction problem is given by m constraints applied to n Boolean variables. Different computational problems arise from constraint satisfaction problems depending on the nature of the "underlying" constraints as well as on the goal of the optimization task. Here we consider four possible goals: MAX CSP (MIN CSP) is the class of problems where the goal is to find an assignment maximizing the number of satisfied constraints (minimizing the number of unsatisfied constraints). MAX ONES (MIN ONES) is the class of optimization problems where the goal is to find an assignment satisfying all constraints with maximum (minimum) number of variables set to 1. Each class consists of infinitely many problems and a problem within a class is specified by a finite collection of finite Boolean functions that describe the possible constraints that may be used.

Tight bounds on the approximability of every problem in MAX CSP were obtained by Creignou [*J. Comput. System Sci.,* 51 (1995), pp. 511–522]. In this work we determine tight bounds on the "approximability" (i.e., the ratio to within which each problem may be approximated in polynomial time) of every problem in MAX ONES, MIN CSP, and MIN ONES. Combined with the result of Creignou, this completely classifies all optimization problems derived from Boolean constraint satisfaction. Our results capture a diverse collection of optimization problems such as MAX 3-SAT, MAX CUT, MAX CLIQUE, MIN CUT, NEAREST CODEWORD, etc. Our results unify recent results on the (in-)approximability of these optimization problems and yield a compact presentation of most known results. Moreover, these results provide a formal basis to many statements on the behavior of natural optimization problems that have so far been observed only empirically.

Key words. approximation algorithms, approximation classes, approximation-preserving reductions, complete problems, Boolean constraint satisfaction problems, hardness of approximation

AMS subject classifications. 68Q15, 68Q17, 68W25

PII. S0097539799349948

1. Introduction.

The approximability of an optimization problem is the best possible performance ratio that is achieved by a polynomial time approximation algorithm for the problem. The approximability is studied as a function of the input size and is always a function bounded from below by 1. Research in the 1990s has led to dramatic progress in our understanding of the approximability of many central optimization problems. The results cover a large number of optimization problems,

*Received by the editors January 18, 1999; accepted for publication (in revised form) October 31, 2000; published electronically March 13, 2001. Preliminary versions of parts of this paper appeared in *Proceedings of the* 29*th Annual ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 11–20, and in *Proceedings of the* 12*th Annual IEEE Conference on Computational Complexity*, Ulm, Germany, IEEE Computer Society Press, 1997, pp. 282–296.

http://www.siam.org/journals/sicomp/30-6/34994.html

†Department of CIS, University of Pennsylvania, Philadelphia, PA 19104 (sanjeev@cis.upenn.edu). Part of this work was done when this author was a graduate student at Stanford University, and another part was done when this author was at Bell Laboratories.

‡Laboratory for Computer Science, MIT, 545 Technology Square, Cambridge, MA 02139 (madhu@lcs.mit.edu). Part of this work was done when this author was at the IBM Thomas J. Watson Research Center.

§Computer Science Division, University of California, Berkeley, CA 94720 (luca@eecs.berkeley.edu). This work was done at the University of Rome "La Sapienza."

¶IBM Almaden Research Center, K53/B1, 650 Harry Road, San Jose, CA 95120 (dpw@almaden.ibm.com).

deriving tight bounds on the approximability of some, while deriving "asymptotically" tight bounds on many more.[1]

In this paper we study optimization problems derived from "Boolean constraint satisfaction problems" and present a complete classification of these problems based on their approximability. Our work is motivated by an attempt to unify this recent progress on the (in-)approximability of combinatorial optimization problems. In the case of positive results, i.e., bounding the approximability from above, a few paradigms have been used repeatedly and these serve to unify the results nicely. In contrast, there is a lack of similar unification among negative or inapproximability results. Inapproximability results are established by approximation-preserving reductions from hard problems, and such reductions tend to exploit every feature of the problem whose hardness is being shown, rather than isolating the "minimal" features that would suffice to obtain the hardness result. As a result inapproximability results are typically isolated and are not immediately suited for unification.

The need for a unified study is, however, quite essential at this stage. The progress in the understanding of optimization problems has shown large amounts of diversity in their approximability. Despite this diversity, natural optimization problems do seem to exhibit some noticeable trends in their behavior. However, in the absence of a terse description of known results, it is hard to extract the trends, let alone trying to provide them with a formal basis. Some such trends are described below:

- There exist optimization problems that are solvable exactly, that admit polynomial time approximation schemes (PTAS) (i.e., for every constant $\alpha > 1$, there exists a polynomial time α-approximation algorithm), that admit constant factor approximation algorithms, that admit logarithmic factor approximation algorithms, and that admit polynomial factor approximation algorithms. However, this list appears to be nearly exhaustive, raising the question, "Are there 'natural' optimization problems with intermediate approximability?"[2]
- A number of minimization problems have an approximability of logarithmic factors. However, so far no natural maximization problem has been shown to have a similar approximability, raising the question, "Are there any 'natural' maximization problems which are approximable to within polylogarithmic factors, but no better?"
- Papadimitriou and Yannakakis [39] define a class of optimization problems called MAX SNP. This class has played a central role in many of the recent inapproximability results, and yet even now the class does not appear to be fully understood. The class contains a number of NP-hard problems, and for all such known problems it turns out to be the case that the approximability is bounded away from 1! This raises the natural question, "Are there any NP-hard problems in MAX SNP that admit polynomial time approximation schemes?"

In order to study such questions, or even to place them under a formal setting,

[1] We say that the approximability of an optimization is known asymptotically if we can determine a function $f : \mathcal{Z} \to \mathcal{Z}$ and constants c_1, c_2 such that the approximability is between $1 + f(n)$ and $1 + c_1 f(n^{c_2})$. This choice is based on the common choice of an approximation preserving reduction. See Definition 2.7.

[2] There are problems such as the *minimum feedback arc set* for which the best known approximation factor is $O(\log n \log\log n)$ [16] and the *asymmetric p-center problem,* where the best known approximation factor is $O(\log^* n)$ [38]. However, no matching inapproximability results are known for such problems.

one needs to first specify the optimization problems in some uniform framework. Furthermore, one has to be careful to ensure that the task of determining whether the optimization problem studied is easy or hard (to, say, compute exactly) is decidable. Unfortunately, barriers such as Rice's theorem (which says this question may not in general be decidable) or Ladner's theorem (which says problems may not be just easy or hard [35]) force us to severely restrict the class of problems which can be studied in such a manner.

Schaefer [42] isolates one class of decision problems which can actually be classified completely. He obtains this classification by restricting his attention to "Boolean constraint satisfaction problems." A problem in this class is specified by a finite set $\mathcal{F}$ of Boolean functions on finitely many variables, referred to as the constraints. (These functions are specified by, say, a truth table.) A function $f : \{0,1\}^k \to \{0,1\}$, when applied to k variables $x_1, \ldots, x_k$, represents the constraint $f(x_1, \ldots, x_k) = 1$. An instance of a constraint satisfaction problem specified by $\mathcal{F}$ consists of m "constraint applications" on n Boolean variables, where each constraint application is the application of one of the constraints from $\mathcal{F}$ to some ordered subset of the n variables. The language $\textsc{Sat}(\mathcal{F})$ consists of all instances which have an assignment satisfying all m constraints. Schaefer describes six classes of function families such that if $\mathcal{F}$ is a subset of one of these classes, then the decision problem is in P; otherwise he shows that the decision problem is NP-hard.

Our setup. In this paper we consider four different optimization versions of Boolean constraint satisfaction problems. In each case the problem is specified by a family $\mathcal{F}$ and the instance by m constraints from $\mathcal{F}$ applied to n Boolean variables. The goals for the four versions vary as follows: In the problem $\textsc{Max CSP}(\mathcal{F})$ the goal is to find an assignment that maximizes the number of satisfied constraints. Analogously in the problem $\textsc{Min CSP}(\mathcal{F})$ the goal is to find an assignment that minimizes the number of unsatisfied constraints. Notice that while the problems are equivalent w.r.t. exact computation, their approximability may be (and often is) very different. In the problem $\textsc{Max Ones}(\mathcal{F})$ ($\textsc{Min Ones}(\mathcal{F})$) the goal is to find an assignment satisfying all constraints while maximizing (minimizing) the number of variables set to 1. We also consider the weighted version of all the above problems. In the case of $\textsc{Weighted Max CSP}(\mathcal{F})$ ($\textsc{Weighted Min CSP}(\mathcal{F})$) the instance includes a nonnegative weight for every constraint and the goal is to maximize (minimize) the sum of the weights of the satisfied (unsatisfied) constraints. In the case of $\textsc{Weighted Max Ones}(\mathcal{F})$ ($\textsc{Weighted Min Ones}(\mathcal{F})$) the instance includes a nonnegative weight for every variable and the goal is to find an assignment satisfying all constraint maximizing (minimizing) the weight of the variables set to 1. The collection of problems $\{\textsc{Max CSP}(\mathcal{F}) \mid \mathcal{F} \text{ finite}\}$ yields the class Max CSP, and similarly we get the classes (Weighted) Min CSP, Max Ones, Min Ones.

Together these classes capture a host of interesting optimization problems. Max CSP is a subset of MAX SNP and forms a combinatorial core of the problems in MAX SNP. It also includes a number of well-studied MAX SNP-complete problems, including MAX 3-SAT, MAX 2-SAT, and Max Cut. Max Ones shows more varied behavior among maximization problems and includes Max Clique and a problem equivalent to Max Cut. Min CSP and Min Ones are closely related to each other capturing very similar problems. The list of problems expressible as one of these includes the *s-t* Min Cut problem, Vertex Cover, Hitting Set with bounded size sets, integer programs with two variables per inequality [25], Min UnCut [20], Min 2CNF Deletion [33], and Nearest Codeword [2]. The ability to study

all these different problems in a uniform framework and extract the features that make the problems easier/harder than the others shows the advantage of studying optimization problems under the constraint satisfaction framework.

We provide a complete characterization of the asymptotic approximability of every optimization problem in the classes mentioned above. For the class MAX CSP such a classification was obtained by Creignou [11], who shows that every problem in the class is either solvable to optimality in polynomial time or has a constant approximability bounded away from 1. For the remaining classes we provide complete characterizations. The detailed statement of our results, comprising 22 cases, appear in Theorems 2.11–2.14. (This includes a technical strengthening of the results of Creignou [11].) In short the results show that every MAX ONES problem either is solvable optimally in P, or has constant factor approximability or polynomial approximability, or it is hard to find feasible solutions. For the minimization problems, the results show that the approximability of every problem lies in one of at most seven levels. However, it does not pin down the approximability of every problem but rather highlights a number of open problems in the area of minimization that deserve further attention. In particular, it exposes a class of problems for which MIN UNCUT is complete, a class for which MIN 2CNF DELETION is complete, and a class for which NEAREST CODEWORD is complete. The approximability of these problems is not yet resolved.

Our results do indeed validate some of the observations about trends exhibited by optimization problems. We find that when restricted to constraint satisfaction problems, the following can be formally established. The approximability of optimization problems does come from a small number of levels; maximization problems do not have a log-approximable representative while minimization problems may have such representatives (e.g., MIN UNCUT). NP-hard MAX CSP problems are also MAX SNP-hard. We also find that weights do not play any significant role in the approximability of combinatorial optimization problems, a thesis in the work of Crescenzi, Silvestri, and Trevisan [15].[3]

Finally, we conclude with some thoughts on directions for further work. We stress that while constraint satisfaction problems provide a good collection of core problems to work with, they are by no means an exhaustive or even near-exhaustive collection of optimization problems. Our framework lacks such phenomena as PTAS; it does not capture several important optimization problems such as the traveling salesman problem and numerous scheduling, sequencing, and graph partitioning problems. One possible reason for the nonexistence of PTAS is that in our problems the input instances have no restrictions in the manner in which constraints may be imposed on the input variables. Significant insight may be gleaned by restricting the problem instances. A widely prescribed condition is that the incidence graph on the variables and the constraints should form a planar graph. This restriction has been studied by Khanna and Motwani [28] and they show that it leads to PTAS for a general class of constraint satisfaction problems. Another input restriction of interest could be that variables are allowed to participate only in a bounded number of constraints. We are unaware of any work on this front. An important extension of our work would be to consider constraint families which contain constraints of unbounded arity (such as

[3]Our definition of an unweighted problem is more loose than that of Crescenzi, Silvestri, and Trevisan. In their definition they disallow instances with repeated constraints, while we do allow them. We believe that it may be possible to remove this discrepancy from our work by a careful analysis of all proofs. We do not carry out this exercise here.

those included in the class MIN $F^+\Pi_1$ studied by Kolaitis and Thakur [34]). Such an extension would allow us to capture problems such as SET COVER. Other directions include working with larger domain sizes (rather than Boolean domains for the variables) and working over spaces where the solution space is the set of all permutations of $[n]$ rather than $\{0,1\}^n$.

Related work. The works of Schaefer [42] and Creignou [11] have already been mentioned above. We reproduce some of the results of Creignou in Theorem 2.11 with some technical strengthenings. This strengthening is described in section 2.5. Another point of difference with the result of Creignou is that our techniques allow us to directly work with the approximability of optimization problems, while in her case the results formally establish NP-hardness and the hardness of approximation can in turn be derived from them. A description of these techniques appear in section 2.6. Among other works focusing on classes showing dichotomy is that of Feder and Vardi [17], who consider the "largest" possible class of natural problems in NP that may exhibit a dichotomy. They motivate constraint satisfaction problems over larger domains and highlight a number of central open questions that lie on the path to the resolution of the complexity of deciding them. Creignou and Hermann [12] show a dichotomy result analogous to Schaefer's for counting versions of constraint satisfaction problems. In the area of approximability, the works of Lund and Yannakakis [37] and Zuckerman [45] provide two instances where large classes of problems are shown to be hard to approximate simultaneously—to the best of our knowledge these are the only cases where the results provide hardness for many problems simultaneously. Finally we mention a few results that are directly related to the optimization problems considered here. Trevisan et al. [43] provide an algorithm for finding optimal implementations (or "gadgets" in their terminology) reducing between MAX CSP problems. Karloff and Zwick [27] describe generic methods for finding "semidefinite relaxations" of MAX CSP problems and use these to provide approximation algorithms for these problems. These results further highlight the appeal of the "constraint satisfaction" framework for studying optimization problems.

2. Definitions and results.

2.1. Constraints, constraint applications, and constraint families. We start by formally defining constraints and constraint satisfaction problems. Schaefer's work [42] proposes the study of such problems as a generalization of 3-satisfiability (3-SAT). We will use the same example to illustrate the definitions below.

A constraint is a function $f : \{0,1\}^k \to \{0,1\}$. A constraint f is satisfied by an input $s \in \{0,1\}^k$ if $f(s) = 1$. A constraint family $\mathcal{F}$ is a finite collection of constraints $\{f_1, \ldots, f_l\}$. For example, constraints of interest for 3-SAT are described by the constraint family $\mathcal{F}_{3\text{SAT}} = \{\text{OR}_{k,j} : 1 \le k \le 3, 0 \le j \le k\}$, where $\text{OR}_{k,j} : \{0,1\}^k \to \{0,1\}$ denotes the constraint $\neg x_1 \bigvee \cdots \bigvee \neg x_j \bigvee x_{j+1} \bigvee \cdots \bigvee x_k$. A constraint application, of a constraint f to n Boolean variables, is a pair $\langle f, (i_1, \ldots, i_k)\rangle$, where the indices $i_j \in [n]$ select k of the n Boolean variables to whom the constraint is applied. (Here and throughout the paper we use the notation $[n]$ to denote the set $\{1, \ldots, n\}$.) For example, to generate the clause $(x_5 \bigvee \neg x_3 \bigvee x_2)$, we could use the constraint application $\langle \text{OR}_{3,1}, (3,5,2)\rangle$ or $\langle \text{OR}_{3,1}, (3,2,5)\rangle$. Note that the applications allow the constraint to be applied to different ordered sets of variables but *not literals*. This distinction is an important one and is the reason that we need all the constraints $\text{OR}_{3,0}, \text{OR}_{3,1}$, etc. to describe 3-SAT. In a constraint application $\langle f, (i_1, \ldots, i_k)\rangle$, we require that $i_j \neq i_{j'}$ for $j \neq j'$, i.e., the variables are not allowed to be replicated

within a constraint application. This is why we need both the functions $\mathrm{OR}_{2,0}$ as well as $\mathrm{OR}_{3,0}$ in 3-SAT.

Constraints and constraint families are the ingredients that specify an optimization *problem*. Thus it is necessary that their description be finite. (Notice that the description of $\mathcal{F}_{3\mathrm{SAT}}$ is finite.) Constraint applications are used to specify *instances* of optimization problems (as well as instances of Schaefer's generalized satisfiability problems) and the fact that their description lengths grow with the instance size is crucially exploited here. (Notice that the description size of a constraint application used to describe a 3-SAT clause will be $\Omega(\log n)$.) While this distinction between constraints and constraint applications is important, we will often blur this distinction in the rest of this paper. In particular we may often let the constraint application $C = \langle f, (i_1, \ldots, i_k)\rangle$ refer only to the constraint f. In particular, we will often use the expression "$C \in \mathcal{F}$" when we mean "$f \in \mathcal{F}$, where f is the first component of C." We now describe Schaefer's class of satisfiability problems and the optimization problems considered in this paper.

Definition 2.1 (Sat($\mathcal{F}$)).

Instance. *A collection of m constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^{m}$ on Boolean variables $x_1, x_2, \ldots, x_n$, where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*

Objective. *Decide if there exists a Boolean assignment to the x_i's which satisfies all the constraints.*

For example, the problem Sat($\mathcal{F}_{3\mathrm{SAT}}$) is the classical 3-SAT problem.

Definition 2.2 (Max CSP($\mathcal{F}$) (Min CSP($\mathcal{F}$))).

Instance. *A collection of m constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^{m}$ on Boolean variables $x_1, x_2, \ldots, x_n$, where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*

Objective. *Find a Boolean assignment to x_i's so as to maximize (minimize) the number of satisfied (unsatisfied) constraints.*

In the weighted problem Weighted Max CSP($\mathcal{F}$) *(*Weighted Min CSP($\mathcal{F}$)*) the input instance includes m nonnegative weights $w_1, \ldots, w_m$, and the objective is to find an assignment which maximizes (minimizes) the sum of the weights of the satisfied (unsatisfied) constraints.*

Definition 2.3 (Max Ones($\mathcal{F}$) (Min Ones($\mathcal{F}$))).

Instance. *A collection of m constraint applications of the form $\{\langle f_j, (i_1(j), \ldots, i_{k_j}(j))\rangle\}_{j=1}^{m}$ on Boolean variables $x_1, x_2, \ldots, x_n$, where $f_j \in \mathcal{F}$ and k_j is the arity of f_j.*

Objective. *Find a Boolean assignment to x_i's which satisfies all the constraints and maximizes (minimizes) the total number of variables assigned true.*

In the weighted problem Weighted Max Ones($\mathcal{F}$) *(*Weighted Min Ones($\mathcal{F}$)*) the input instance includes n nonnegative weights $w_1, \ldots, w_n$, and the objective is to find an assignment which satisfies all constraints and maximizes (minimizes) the sum of the weights of variables assigned true.*

The class (Weighted) Max CSP is the set of all optimization problems (Weighted) Max CSP($\mathcal{F}$) for every constraint family $\mathcal{F}$. The classes (Weighted) Max Ones, Min CSP, Min Ones are defined similarly.

The optimization problem Max 3 Sat is easily seen to be equivalent to Max CSP($\mathcal{F}_{3\mathrm{SAT}}$). This and the other problems Max Ones($\mathcal{F}_{3\mathrm{SAT}}$), Min CSP($\mathcal{F}_{3\mathrm{SAT}}$), and Min Ones($\mathcal{F}_{3\mathrm{SAT}}$) are considered in the rest of this paper. More interesting examples of Max Ones, Min CSP, and Min Ones problems are described in sec-

tion 2.3. We start with some preliminaries on approximability that we need to state our results.

2.2. Approximability, reductions, and completeness. A *combinatorial optimization* problem is defined over a set of *instances* (admissible input data); a finite set $\mathsf{sol}(x)$ of *feasible solutions* is associated with any instance. An *objective function* attributes an integer value to any solution. The *goal* of an optimization problem is, given an instance x, to find a solution $y \in \mathsf{sol}(x)$ of *optimum* value. The optimum value is the largest one for *maximization* problems and the smallest one for *minimization* problems. A combinatorial optimization problem is said to be an NP optimization (NPO) problem if instances and solutions can be recognized in polynomial time, solutions are polynomial-bounded in input size, and the objective function can be computed in polynomial time (see, e.g., [10]).

DEFINITION 2.4 (performance ratio). *A solution s to an instance $\mathcal{I}$ of an* NPO *problem A is r-approximate if it has a value V satisfying*

$$\max\left\{\frac{V}{\mathrm{OPT}(\mathcal{I})}, \frac{\mathrm{OPT}(\mathcal{I})}{V}\right\} \leq r.$$

An approximation algorithm for an NPO *problem A has* performance ratio $\mathcal{R}(n)$ *if, given any instance $\mathcal{I}$ of A with $|\mathcal{I}| = n$, it outputs an $\mathcal{R}(n)$-approximate solution.*

We say that an NPO problem is approximable to within a factor $\mathcal{R}(n)$ if it has a polynomial time approximation algorithm with performance ratio $\mathcal{R}(n)$.

DEFINITION 2.5 (approximation classes). *An* NPO *problem A is in the class* PO *if it is solvable to optimality in polynomial time. A is in the class* APX *(resp.,* log-APX*/*poly-APX*) if there exists a polynomial time algorithm for A whose performance ratio is bounded by a constant (resp., logarithmic/polynomial factor in the size of the input).*

Completeness in approximation classes can be defined using the appropriate approximation-preserving reducibilities. In this paper, we use two notions of reducibility: A-reducibility and AP-reducibility. We discuss the difference between the two and the need for having two different notions after the definitions.

DEFINITION 2.6 (A-reducibility [14]). *An* NPO *problem A is said to be A-reducible to an* NPO *problem B, denoted $A \leq_{\mathrm{A}} B$, if two polynomial time computable functions F and G and a constant α exist such that*

1. *for any instance $\mathcal{I}$ of A, $F(\mathcal{I})$ is an instance of B;*
2. *for any instance $\mathcal{I}$ of A and any feasible solution $\mathcal{S}'$ for $F(\mathcal{I})$, $G(\mathcal{I}, \mathcal{S}')$ is a feasible solution for $\mathcal{I}$;*
3. *for any instance $\mathcal{I}$ of A and any $r \geq 1$, if $\mathcal{S}'$ is an r-approximate solution for $F(\mathcal{I})$, then $G(\mathcal{I}, \mathcal{S}')$ is an (αr)-approximate solution for $\mathcal{I}$.*

DEFINITION 2.7 (AP-reducibility [13]). *For a constant $\alpha > 0$ and two* NPO *problems A and B, we say that A is α-AP-reducible to B, denoted $A \leq_{\mathrm{AP}} B$, if two polynomial time computable functions F and G exist such that the following holds:*

1. *For any instance $\mathcal{I}$ of A, $F(\mathcal{I})$ is an instance of B.*
2. *For any instance $\mathcal{I}$ of A, and any feasible solution $\mathcal{S}'$ for $F(\mathcal{I})$, $G(\mathcal{I}, \mathcal{S}')$ is a feasible solution for $\mathcal{I}$.*
3. *For any instance $\mathcal{I}$ of A and any $r \geq 1$, if $\mathcal{S}'$ is an r-approximate solution for $F(\mathcal{I})$, then $G(\mathcal{I}, \mathcal{S}')$ is an $(1 + (r-1)\alpha + o(1))$-approximate solution for $\mathcal{I}$, where the $o()$-notation is with respect to $|\mathcal{I}|$.*

We say that A is AP-reducible to B if a constant $\alpha > 0$ exists such that A is α-AP-reducible to B.

Remark.

(1) Notice that conditions (3) of both reductions preserve only the quality of an approximate solution in absolute terms (to within the specified limits) and not as functions of the instance size. For example, an A-reduction from Π to Π' which blows up instance size by quadratic factor and an $O(n^{1/3})$ approximation algorithm for Π' combine to give only an $O(n^{2/3})$ approximation algorithm for Π.

(2) The difference between the two reductions is the level of approximability that is preserved by them (conditions (3) in the definitions). A-reductions preserve constant factor approximability or higher, i.e., if Π is A-reducible to Π' and Π' is approximable to within a factor of $r(n)$, then Π is approximable to within $\alpha r(n^c)$ for some constants α, c. This property suffices to preserve membership in APX (log-APX, poly-APX), i.e., if Π is in APX (log-APX, poly-APX), then Π' is also in APX (resp., log-APX, poly-APX). However, it does not preserve membership in PO or PTAS, as can be observed by setting $r = 1$.

(3) AP-reductions are more sensitive than A-reductions. Thus if Π is AP-reducible to Π, then an r-approximate solution is mapped to an $h(r)$-approximate solution where $h(r) \to 1$ as $r \to 1$. Thus AP-reductions preserve membership in PTAS as well. However, they need not preserve membership in PO (due to the $o(1)$-term in their preservation of approximability).

(4) Condition (3) of the definition of AP-reductions is strictly stronger than the corresponding condition in the definition of A-reductions. Thus, every AP-reduction is also an A-reduction. Unfortunately, neither one of these reductions on their own suffice for our purposes. We need AP-reductions to show APX-hardness of problems, but we need the added flexibility of A-reductions for other hardness results.

(5) The original definitions of AP-reducibility and A-reducibility of [14] and [13] were more general. Under the original definitions, the A-reducibility does not preserve membership in log-APX, and it is not clear whether every AP-reduction is also an A-reduction. The restricted versions defined here are more suitable for our purposes. In particular, it is true that the VERTEX COVER problem is APX-complete under our definition of AP-reducibility.

DEFINITION 2.8 (APX, log-APX, and poly-APX-completeness). *An* NPO *problem* Π *is* APX*-hard if every* APX *problem is AP-reducible to* Π. *An* NPO *problem* Π *is* log-APX*-hard (*poly-APX*-hard) if every* log-APX *(*poly-APX*) problem is A-reducible to* Π. *A problem* Π *is* APX*(*log-APX*,* poly-APX*)-complete if it is in* APX *(resp.,* log-APX*,* poly-APX*) and it is* APX *(resp.,* log-APX*,* poly-APX*)-hard.*

The class APX contains the class MAX SNP as defined by Papadimitriou and Yannakakis [39]. The containment is strict in a syntactic sense (e.g., MAX SNP does not contain any minimization problems); however, when one takes the closure of APX under AP-reductions, one obtains the class MAX SNP [29]. The notion of reductions used here is also less stringent than the notion of reduction used in [39]. Thus APX, APX-hardness, and APX-completeness are (mild) generalizations of the notions of MAX SNP, MAX SNP-hardness, and MAX SNP-completeness.

Most problems we consider are known/shown to be in PO or else are APX-complete or poly-APX-complete. However, in some cases, we will not be able to establish the exact approximability of a given problem. However, we will nevertheless be able to compile all problems into a finite number of equivalence classes with some equivalence classes being defined as "problems equivalent to Π" for some problem Π

of unknown approximability. The following definition captures this concept.

DEFINITION 2.9 (Π-completeness). *For* NPO *problems* Π *and* Π′, Π′ *is said to be* Π*-complete if* $\Pi \leq_{\mathrm{A}} \Pi'$ *and* $\Pi' \leq_{\mathrm{A}} \Pi$.

2.3. Problems captured by MAX CSP, MAX ONES, MIN CSP, and MIN ONES. We first specify our notation for commonly used functions.

- **0** and **1** are the functions which are always satisfied and never satisfied, respectively. Together these are the trivial functions. We will assume that all our function families do not have any trivial functions.
- T and F are unary functions given by $T(x) = x$ and $F(x) = \neg x$.
- For a positive integer i and nonnegative integer $j \leq i$, $\mathrm{OR}_{i,j}$ is the function on i variables given by $\mathrm{OR}_{i,j}(x_1, \ldots, x_i) = \neg x_1 \bigvee \cdots \bigvee \neg x_j \bigvee x_{j+1} \bigvee \cdots \bigvee x_i$. $\mathrm{OR}_i = \mathrm{OR}_{i,0}$; $\mathrm{NAND}_i = \mathrm{OR}_{i,i}$; $\mathrm{OR} = \mathrm{OR}_2$; $\mathrm{NAND} = \mathrm{NAND}_2$.
- Similarly, $\mathrm{AND}_{i,j}$ is given by $\mathrm{AND}_{i,j}(x_1, \ldots, x_i) = \neg x_1 \bigwedge \cdots \bigwedge \neg x_j \bigwedge x_{j+1} \bigwedge \cdots \bigwedge x_i$. $\mathrm{AND}_i = \mathrm{AND}_{i,0}$; $\mathrm{NOR}_i = \mathrm{AND}_{i,i}$; $\mathrm{AND} = \mathrm{AND}_2$; $\mathrm{NOR} = \mathrm{NOR}_2$.
- The function XOR_i is given by $\mathrm{XOR}(x_1, \ldots, x_i) = x_1 \oplus \cdots \oplus x_i$. $\mathrm{XOR} = \mathrm{XOR}_2$.
- The function XNOR_i is given by $\mathrm{XNOR}(x_1, \ldots, x_i) = \neg(x_1 \oplus \cdots \oplus x_i)$. $\mathrm{XNOR} = \mathrm{XNOR}_2$.

Now we enumerate some interesting maximization and minimization problems which are "captured" by (i.e., are equivalent to some problem in) MAX CSP, MAX ONES, MIN CSP, and MIN ONES. The following list is interesting for several reasons. First, it highlights the importance of these classes as ones that contain interesting optimization problems and shows the diversity of the problems captured by these classes. Furthermore, each of these problems turn out to be "complete" problems for the partitions they belong to. Some are even necessary for a full statement of our results. Finally, for several of the minimization problems listed below, their approximability is not yet fully resolved. We feel that these problems are somehow representative of the lack of our understanding of the approximability of minimization problems. We start with the maximization problems.

- For any positive integer k, MAX kSAT = MAX CSP($\{\mathrm{OR}_{i,j} | i \in [k], 0 \leq j \leq i\}$). MAX kSAT is a well-studied problem and known to be MAX SNP-complete [39] for $k \geq 2$. Every MAX SNP-complete problem is in APX (i.e., approximable to within a constant factor in polynomial time) [39] (see also [6, 21, 44]). Also for MAX SNP-complete problem there exists a constant α greater than 1 such that the problem is not α-approximable unless NP = P [3, 4, 24] .
- For any positive integer k, MAX EkSAT = MAX CSP($\{\mathrm{OR}_{k,j} | 0 \leq j \leq k\}$). The problem MAX E$k$SAT is a variant of MAX kSAT restricted to have clauses of length exactly k.
- MAX CUT = MAX CSP($\{\mathrm{XOR}\}$). MAX CUT is also MAX SNP-complete [39], and the best known approximation algorithm for this problem, due to [22], achieves a performance ratio of $1.14 \approx 1/.878$
- MAX CLIQUE = MAX ONES(NAND). MAX CLIQUE is known to be approximable to within a factor of $O(n/\log^2 n)$ in an n-vertex graph [9] and is known to be hard to approximate to within a factor of $\Omega(n^{1-\epsilon})$ for any $\epsilon > 0$ unless NP = RP [18, 23].

We now go on to the minimization problems.

- The well-known minimum s-t cut problem in directed graphs is equivalent to WEIGHTED MIN CSP($\mathcal{F}$) for $\mathcal{F} = \{\mathrm{OR}_{2,1}, T, F\}$. This is shown in section 5.1.

This problem is well known to be solvable exactly in polynomial time.

- The HITTING SET problem, restricted to instances in which all sets are of size at most B, can be captured as MIN ONES($\mathcal{F}$) for $\mathcal{F} = \{\mathrm{OR}_k | k \leq B\}$. Also, of interest to our paper is a slight generalization of this problem which we call the implicative HITTING SET-B problem (MIN IHS-B) which is MIN CSP($\{\mathrm{OR}_k : k \leq B\} \cup \{\mathrm{OR}_{2,1}, F\}$). The MIN ONES version of this problem will be of interest to us as well. The HITTING SET-B problem is well known to be approximable to within a factor of B. We show that MIN IHS-B is approximable to within a factor of $B + 1$.
- MIN UNCUT = MIN CSP($\{\mathrm{XOR}\}$). This problem has been studied previously by Klein et al. [32] and Garg, Vazirani, and Yannakakis [20]. The problem is known to be MAX SNP-hard and hence not approximable to within some constant factor greater than 1. On the other hand, the problem is known to be approximable to within a factor of $O(\log n)$ [20].
- MIN 2CNF DELETION = MIN CSP($\{\mathrm{OR}, \mathrm{NAND}\}$). This problem has been studied by Klein et al. [33]. They show that the problem is MAX SNP-hard and that it is approximable to within a factor of $O(\log n \log \log n)$.
- NEAREST CODEWORD = MIN CSP($\{\mathrm{XOR}_3, \mathrm{XNOR}_3\}$). This is a classical problem for which hardness of approximation results have been shown by Arora et al. [2]. The MIN ONES version of this problem is essentially identical to this problem. For both problems, the hardness result of Arora et al. [2] shows that approximating this problem to within a factor of $\Omega(2^{\log^{1-\epsilon} n})$ is hard for every $\epsilon > 0$, unless NP $\subseteq$ QP. No nontrivial approximation guarantees are known for this problem (the trivial bound being a factor of m, which is easily achieved since deciding if all equations are satisfiable amounts to solving a linear system).
- Finally we also mention one more problem which is required to present our main theorem. MIN HORN DELETION = MIN CSP($\{\mathrm{OR}_{3,1}, T, F\}$). The currently known bounds on the approximability of this problem are similar to those of the NEAREST CODEWORD, i.e., it is in poly-APX and hard to approximate to within a factor of $2^{\Omega(\log^{1-\epsilon} n)}$ (see Lemma 7.21).

2.4. Properties of function families. We start with the six properties defined by Schaefer:

- A constraint f is 0-*valid* (resp., 1-*valid*) if $f(0, \ldots, 0) = 1$ (resp., $f(1, \ldots, 1) = 1$).
- A constraint is *weakly positive* (resp., *weakly negative*) if it can be expressed as a CNF-formula having at most one negated variable (resp., at most one unnegated variable[4]) in each clause.
- A constraint is *affine* if it can be expressed as a conjunction of linear equalities over $\mathcal{Z}_2$.
- A constraint is 2*cnf* if it is expressible as a 2CNF-formula.

The above definitions extend to constraint families naturally. For instance, a constraint family $\mathcal{F}$ is 0-*valid* if *every* constraint $f \in \mathcal{F}$ is 0-valid. The above definitions are central to Schaefer's main theorem, restated below.

THEOREM 2.10 (Schaefer's theorem [42]). *For any constraint family $\mathcal{F}$,* SAT($\mathcal{F}$) *is in* P *if $\mathcal{F}$ is* 0*-valid or* 1*-valid or weakly positive or weakly negative or affine or* 2*cnf; otherwise deciding* SAT($\mathcal{F}$) *is* NP*-hard.*

[4]Such clauses are usually called Horn clauses.

We use the shorthand "$\mathcal{F}$ is (not) decidable" to say that deciding membership in $\text{SAT}(\mathcal{F})$ is solvable in P (is NP-hard). Abusing our vocabulary slightly, we say MAX ONES($\mathcal{F}$) (or MIN ONES($\mathcal{F}$)) is not decidable to indicate that determining if a given instance of this problem has a feasible solution is NP-hard.

We need to define some additional properties to describe the approximabilities of the optimization problems we consider:

- f is 2-*monotone* if $f(x_1, \ldots, x_k)$ is expressible as
$$(x_{i_1} \bigwedge \cdots \bigwedge x_{i_p}) \bigvee (\neg x_{j_1} \bigwedge \cdots \bigwedge \neg x_{j_q})$$
for some $p, q \geq 0$, $(p, q) \neq (0, 0)$ (i.e., f is expressible as a DNF-formula with at most two terms—one containing only positive literals and the other containing only negative literals).
- A constraint is *width*-2 *affine* if it is expressible as a conjunction of linear equations over $\mathcal{Z}_2$ such that each equation has at most two variables.
- A constraint is *strongly* 0-*valid* if it is satisfied by all assignments with at most one variable set to 1. (Note that a strongly 0-valid constraint is also 0-valid.)
- A constraint f is *IHS-B+* (for *implicative hitting set-bounded+*) if it is expressible as a CNF formula, where the clauses are of one of the following types: $x_1 \bigvee \cdots \bigvee x_k$ for some positive integer $k \leq B$, or $\neg x_1 \bigvee x_2$, or $\neg x_1$. IHS-$B-$ constraints and constraint families are defined analogously (with every literal being replaced by its complement). A family is an IHS-B family if the family is an IHS-$B+$ family or an IHS-$B-$ family.

We use the following shorthand for the above families: (1) $\mathcal{F}_0$ is the family of 0-valid constraints; (2) $\mathcal{F}_1$ is the family of 1-valid constraints; (3) $\mathcal{F}_{\text{S0}}$ is the family of strongly 0-valid constraints; (4) $\mathcal{F}_{\text{2M}}$ is the family of 2-monotone constraints; (5) $\mathcal{F}_{\text{IHS}}$ is the family of IHS-B constraints; (6) $\mathcal{F}_{\text{2A}}$ is the family of width-2 affine constraints; (7) $\mathcal{F}_{\text{2CNF}}$ is the family of 2CNF constraints; (8) $\mathcal{F}_{\text{A}}$ is the family of affine constraints; (9) $\mathcal{F}_{\text{WP}}$ is the family of weakly positive constraints; (10) $\mathcal{F}_{\text{WN}}$ is the family of weakly negative constraints.

2.5. Main results. We now present the main results of this paper. A pictorial representation is available in Appendices B.1, B.2, B.3, and B.4. All theorems are stated assuming that $\mathcal{F}$ has no trivial constraints, i.e., constraints that are always satisfied or never satisfied. The first theorem is a minor strengthening of Creignou's theorem [11] so as to cover problems such as MAX EkSAT. The remaining theorems cover MAX ONES, MIN CSP, and MIN ONES, respectively.

THEOREM 2.11 (MAX CSP classification). *For any constraint set $\mathcal{F}$, the problem (*WEIGHTED*)* MAX CSP($\mathcal{F}$) *is always either in* PO *or is* APX*-complete. Furthermore, it is in* PO *if and only if $\mathcal{F}$ is* 0*-valid or* 1*-valid or* 2*-monotone.*

THEOREM 2.12 (MAX ONES classification). *For any constraint set $\mathcal{F}$, the problem (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is either in* PO *or is* APX*-complete or* poly-APX*-complete or decidable but not approximable to within any factor or not decidable. Furthermore,*

1. *if $\mathcal{F}$ is* 1*-valid or weakly positive or affine with width-*2*, then (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is in* PO*;*
2. *otherwise if $\mathcal{F}$ is affine, then (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is* APX*-complete;*
3. *otherwise if $\mathcal{F}$ is strongly* 0*-valid or weakly negative or* 2*CNF, then (*WEIGHTED*)* MAX ONES($\mathcal{F}$) *is* poly-APX*-complete;*
4. *otherwise if $\mathcal{F}$ is* 0*-valid, then* SAT($\mathcal{F}$) *is in* P *but finding a solution of positive value is* NP*-hard;*

(5) *otherwise finding a feasible solution to (*Weighted*)* Max Ones($\mathcal{F}$) *is* NP*-hard.*

Theorem 2.13 (Min CSP classification). *For any constraint set $\mathcal{F}$, the problem (*Weighted*)* Min CSP($\mathcal{F}$) *is in* PO *or is* APX*-complete or* Min UnCut*-complete or* Min 2CNF Deletion*-complete or* Nearest Codeword*-complete or* Min Horn Deletion*-complete or even deciding if the optimum is zero is* NP*-hard. Furthermore,*

(1) *if $\mathcal{F}$ is* 0*-valid or* 1*-valid or* 2*-monotone, then (*Weighted*)* Min CSP($\mathcal{F}$) *is in* PO*;*
(2) *otherwise if $\mathcal{F}$ is IHS-B, then (*Weighted*)* Min CSP($\mathcal{F}$) *is* APX*-complete;*
(3) *otherwise if $\mathcal{F}$ is width-*2 *affine, then (*Weighted*)* Min CSP($\mathcal{F}$) *is* Min UnCut*-complete;*
(4) *otherwise if $\mathcal{F}$ is 2CNF, then (*Weighted*)* Min CSP($\mathcal{F}$) *is* Min 2CNF Deletion*-complete;*
(5) *otherwise if $\mathcal{F}$ is affine, then (*Weighted*)* Min CSP($\mathcal{F}$) *is* Nearest Codeword*-complete;*
(6) *otherwise if $\mathcal{F}$ is weakly positive or weakly negative, then (*Weighted*)* Min CSP($\mathcal{F}$) *is* Min Horn Deletion*-complete;*
(7) *otherwise deciding if the optimum value of an instance of (*Weighted*)* Min CSP($\mathcal{F}$) *is zero is* NP*-complete.*

Theorem 2.14 (Min Ones classification). *For any constraint set $\mathcal{F}$, the problem (*Weighted*)* Min Ones($\mathcal{F}$) *is either in* PO *or* APX*-complete or* Nearest Codeword*-complete or* Min Horn Deletion*-complete or* poly-APX*-complete or inapproximable to within any factor or not decidable. Furthermore,*

(1) *if $\mathcal{F}$ is* 0*-valid or weakly negative or width-*2 *affine, then (*Weighted*)* Min Ones($\mathcal{F}$) *is in* PO*;*
(2) *otherwise, if $\mathcal{F}$ is 2CNF or IHS-B, then (*Weighted*)* Min Ones($\mathcal{F}$) *is* APX*-complete;*
(3) *otherwise if $\mathcal{F}$ is affine, then* Min Ones($\mathcal{F}$) *is* Nearest Codeword*-complete;*
(4) *otherwise if $\mathcal{F}$ is weakly positive, then (*Weighted*)* Min Ones($\mathcal{F}$) *is* Min Horn Deletion*-complete;*
(5) *otherwise if $\mathcal{F}$ is* 1*-valid, then* Min Ones($\mathcal{F}$) *is* poly-APX*-complete and* Weighted Min Ones($\mathcal{F}$) *is decidable but hard to approximate to within any factor;*
(6) *otherwise finding any feasible solution to (*Weighted*)* Min Ones($\mathcal{F}$) *is* NP*-hard.*

2.6. Techniques. Two simple ideas play an important role in this paper. First is the notion of an *implementation* which shows how to use the constraints of a family $\mathcal{F}$ to enforce constraints of a different family $\mathcal{F}'$, thereby laying the groundwork of a reduction among problems. The notion of an implementation is inspired by the notion of gadgets formalized by Bellare, Goldreich, and Sudan [8], who in our language define implementations for specific pairs of function families $(\mathcal{F}, \mathcal{F}')$. In this work we unify their definition, so as to make it work for arbitrary pairs of function families. This definition of implementation also finds applications in the work of Trevisan et al. [43], who, in our language, show uniform methods for searching for efficient implementations for pairs of function families $(\mathcal{F}, \mathcal{F}')$.

A second simple idea we exploit here is that of working with weighted versions of optimization problems. Even though our primary concerns were only the approximability of the unweighted versions of problems, many of our results use as intermediate steps the weighted versions of these problems. The weights allow us to manipulate

problems more locally. However, simple and well-known ideas eventually allow us to get rid of the weights, thereby yielding hardness of the unweighted problem as well. As a side-effect we also show that the unweighted and weighted problems are equally hard to approximate in all the relevant optimization problems. This extends to minimization problems the results of Crescenzi, Silvestri, and Trevisan [15].

The definitions of implementations and weighted problems follow in section 3. Section 4 shows some technical results showing how we exploit the fact that we have functions which don't exhibit some property. The results of this section play a crucial role in all the hardness results. This sets us up for the proofs of our main theorems. In section 5 we show the containment results and hardness results for Max CSP. Similarly sections 6, 7, and 8 deal with the classes Max Ones, Min CSP, and Min Ones, respectively.

3. Implementations. We now describe the main technique used in this paper to obtain hardness of approximation results. Suppose we want to show that for some constraint set $\mathcal{F}$, the problem Max CSP($\mathcal{F}$) is APX-hard. We will start with a problem that is known to be APX-hard, such as Max Cut, which turns out to be Max CSP($\{\text{XOR}\}$). We will then wish to reduce this problem to Max CSP($\mathcal{F}$). The main technique we use to do this is to "implement" the constraint XOR using constraints from the constraint set $\mathcal{F}$. We show how to formalize this notion next and then show how this translates to approximation-preserving reductions.

Definition 3.1 (implementation). *A collection of constraint applications $C_1, \ldots, C_m$ over a set of variables $\mathbf{x} = \{x_1, \ldots, x_p\}$ called* primary variables *and $\mathbf{y} = \{y_1, \ldots, y_q\}$ called* auxiliary variables *is an* α-implementation *of a constraint $f(\mathbf{x})$ for a positive integer α if the following conditions are satisfied:*

1. *For any assignment to $\mathbf{x}$ and $\mathbf{y}$, at most α constraints from $C_1, \ldots, C_m$ are satisfied.*
2. *For any $\mathbf{x}$ such that $f(\mathbf{x}) = 1$, there exists $\mathbf{y}$ such that exactly α constraints are satisfied.*
3. *For any $\mathbf{x}, \mathbf{y}$ such that $f(\mathbf{x}) = 0$, at most $(\alpha - 1)$ constraints are satisfied.*

Definition 3.2 (strict/perfect implementations). *An α-implementation is a* strict α-implementation *if for every $\mathbf{x}$ such that $f(\mathbf{x}) = 0$, there exists $\mathbf{y}$ such that exactly $(\alpha - 1)$ constraints are satisfied. An α-implementation (not necessarily strict) is a* perfect *implementation if $\alpha = m$.*

We say that a constraint set $\mathcal{F}$ (strictly/perfectly) implements a constraint f if there exists a (strict/perfect) α-implementation of f using constraints of $\mathcal{F}$ for some $\alpha < \infty$. We use the notation $\mathcal{F} \Longrightarrow_\alpha f$ to denote that $\mathcal{F}$ α-implements f, and $\mathcal{F} \Longrightarrow f$ to denote that $\mathcal{F}$ implements f. Similarly we use the notation $\mathcal{F} \overset{s/p}{\Longrightarrow} f$ to denote that $\mathcal{F}$ implements f strictly/perfectly. The above notation is also extended to allow the target to be a family of functions. For instance, $\mathcal{F} \Longrightarrow \mathcal{F}'$ denotes that $\mathcal{F}$ implements every function in $\mathcal{F}'$.

Remark. The definition of [8] defined (nonstrict and nonperfect) implementations for specific choices of f and $\mathcal{F}$. For each choice they provided a separate definition. We unify their definitions into a single one. Furthermore, as we will show later, the use of strictness and/or perfectness greatly enhance the power of implementations. These aspects are formalized for the first time here.

A constraint f 1-implements itself strictly and perfectly ($\{f\} \overset{s/p}{\Longrightarrow}_1 f$). Some more examples of strict and/or perfect implementations are given below.

Proposition 3.3. $\{\text{XOR}\} \overset{s/p}{\Longrightarrow}_2 \text{XNOR}$.

Proof. The constraints $\mathrm{XOR}(x, z_{\mathrm{AUX}})$ and $\mathrm{XOR}(y, z_{\mathrm{AUX}})$ perfectly and strictly implement the constraint $\mathrm{XNOR}(x, y)$. □

PROPOSITION 3.4. *If* $f(\mathbf{x}) = f_1(\mathbf{x}) \bigwedge \cdots \bigwedge f_k(\mathbf{x})$, *then* $\{f_1, \ldots, f_k\} \stackrel{p}{\Longrightarrow}_k f$.

Proof. The collection $\{f_1(\mathbf{x}), \ldots, f_k(\mathbf{x})\}$ is a perfect (but not necessarily strict) k-implementation of $f(\mathbf{x})$. □

The following lemma shows that the implementations of constraints compose together if they are strict or perfect.

LEMMA 3.5. *If* $\mathcal{F}_a \stackrel{s}{\Longrightarrow} \mathcal{F}_b$ *and* $\mathcal{F}_b \stackrel{s}{\Longrightarrow} \mathcal{F}_c$, *then* $\mathcal{F}_a \stackrel{s}{\Longrightarrow} \mathcal{F}_c$. *An analogous result also holds for perfect implementations.*

Proof. It suffices to consider the case when $\mathcal{F}_c$ consists of a single function f. Furthermore, we observe that it suffices to prove the following simpler assertion (to prove the lemma): If $\mathcal{F} \stackrel{s}{\Longrightarrow} g$ and $\mathcal{F} \cup \{g\} \stackrel{s}{\Longrightarrow} f$, then $\mathcal{F} \stackrel{s}{\Longrightarrow} f$. To see that this suffices, let $\mathcal{F}_b = \{g_1, \ldots, g_l\}$. Define $\mathcal{F}_0 = \mathcal{F}_a$, $\mathcal{F}_i = \mathcal{F}_a \cup \{g_1, \ldots, g_i\}$. Note that by hypothesis we have $\mathcal{F}_l \stackrel{s}{\Longrightarrow} f$ and $\mathcal{F}_i \stackrel{s}{\Longrightarrow} g_{i+1}$ and $\mathcal{F}_l \stackrel{s}{\Longrightarrow} f$. The assertion above says that if $\mathcal{F}_{i+1} \stackrel{s}{\Longrightarrow} f$, then $\mathcal{F}_i \stackrel{s}{\Longrightarrow} f$. Thus by induction $\mathcal{F}_0 \stackrel{s}{\Longrightarrow} f$.

We now prove the assertion: If $\mathcal{F} \stackrel{s}{\Longrightarrow} g$ and $\mathcal{F} \cup \{g\} \stackrel{s}{\Longrightarrow} f$, then $\mathcal{F} \stackrel{s}{\Longrightarrow} f$. Let $C_1, \ldots, C_{m_1}$ be constraint applications from $\mathcal{F} \cup \{g\}$ on variables $\mathbf{x}, \mathbf{y}$ giving an α_1-implementation of $f(\mathbf{x})$ with $\mathbf{x}$ being the primary variables. Let $C'_1, \ldots, C'_{m_2}$ be constraint applications from $\mathcal{F}$ on variable set $\mathbf{x}', \mathbf{z}'$ yielding an α_2-implementation of $g(\mathbf{x}')$. Furthermore, let the first β constraints of $C_1, \ldots, C_{m_1}$ be applications of the constraints g.

We create a collection of $m_1 + \beta(m_2 - 1)$ constraints from $\mathcal{F}$ on a set of variables $\mathbf{x}, \mathbf{y}, \mathbf{z'}_1, \ldots, \mathbf{z'}_\beta$, where $\mathbf{x}$ and $\mathbf{y}$ are the original variables and $\mathbf{z}'_1, \ldots, \mathbf{z}'_\beta$ are new sets of disjoint auxiliary variables, i.e., the vectors $\mathbf{z'}_i$ and $\mathbf{z'}_j$ do not share any variables if $i \neq j$.

The $m_1 + \beta(m_2 - 1)$ constraints introduced are as follows. We include the constraint applications $C_{\beta+1}, \ldots, C_{m_1}$ on variables $\mathbf{x}, \mathbf{y}$ and for every constraint application C_j, for $j \in \{1, \ldots, \beta\}$, on variables $\mathbf{v}_j$ (which is a subset of variables from $\mathbf{x}, \mathbf{y}$), we place the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$ on variable set $\mathbf{v}_j, \mathbf{z'}_j$ with $\mathbf{z'}_j$ being the auxiliary variables.

We now show that this collection of constraints satisfies properties (1)–(3) from Definition 3.1 with $\alpha = \alpha_1 + \beta(\alpha_2 - 1)$. Additionally we show that perfectness and/or strictness is preserved. We start with properties (1) and (3).

Consider any assignment to $\mathbf{x}$ satisfying f. Then any assignment to $\mathbf{y}$ satisfies at most α_1 constraints from the set $C_1, \ldots, C_{m_1}$. Let γ of these be from the set $C_1, \ldots, C_\beta$. Now for every $j \in \{1, \ldots, \beta\}$ any assignment to $\mathbf{z'}_j$ satisfies at most α_2 of the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$. Furthermore, if the constraint C_j was not satisfied by the assignment to $\mathbf{x}, \mathbf{y}$, then at most $\alpha_2 - 1$ constraints are satisfied. Thus the total number of constraints satisfied by any assignment is at most $\gamma\alpha_2 + (\beta - \gamma)(\alpha_2 - 1) + (\alpha_1 - \gamma) = \alpha_1 + \beta(\alpha_2 - 1)$. This yields property (1). Property (3) is achieved similarly.

We now show that if the α_1- and α_2-implementations are perfect we get property (2) with perfectness. In this case, for any assignment to $\mathbf{x}$ satisfying f, there exists an assignment to $\mathbf{y}$ satisfying $C_1, \ldots, C_{m_1}$. Furthermore, for every $j \in \{1, \ldots, \beta\}$, there exists an assignment to $\mathbf{z'}_j$ satisfying all the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$. Thus there exists an assignment to $\mathbf{x}, \mathbf{y}, \mathbf{z'}_1, \ldots, \mathbf{z'}_\beta$ satisfying all $m_1 + \beta(m_2 - 1)$ constraints. This yields property (2) with perfectness.

Finally we consider the case when the α_1- and α_2-implementations are strict (but not necessarily perfect) and show that in this case the collection of constraints above

also satisfies property (2) with strictness. Given an assignment to $\mathbf{x}$ satisfying f there exists an assignment to $\mathbf{y}$ satisfying α_1 constraints from $C_1, \ldots, C_{m_1}$. If this assignment satisfied γ clauses from the set $C_1, \ldots, C_\beta$ and $\alpha_1 - \gamma$ constraints from the set $C_{\beta+1}, \ldots, C_{m_1}$, then for every $j \in \{1, \ldots, \beta\}$ such that the clauses C_j is satisfied by this assignment to $\mathbf{x}, \mathbf{y}$, there exists an assignment to $\mathbf{z'}_j$ satisfying α_2 clauses from the set $C'_{1,j}, \ldots, C'_{m_2,j}$. Furthermore, for the remaining values of $j \in \{1, \ldots, \beta\}$ there exists an assignment to the variables $\mathbf{z'}_j$ satisfying $\alpha_2 - 1$ of the constraints $C'_{1,j}, \ldots, C'_{m_2,j}$ (here we are using the strictness of the α_2-implementations). This setting to $\mathbf{y}, \mathbf{z'}_1, \ldots, \mathbf{z'}_\beta$ satisfies $\gamma\alpha_2 + (\beta - \gamma)(\alpha_2 - 1) + \alpha_1 - \gamma = \alpha_1 + \beta(\alpha_2 - 1)$ of the m constraints. This yields property (2). A similar analysis can be used to show the strictness. □

Next we show a simple monotonicity property of implementations.

LEMMA 3.6. *For integers α, α' with $\alpha \leq \alpha'$, if $\mathcal{F} \Longrightarrow_\alpha f$, then $\mathcal{F} \Longrightarrow_{\alpha'} f$. Furthermore, strictness and perfectness are preserved under this transformation.*

Proof. Let constraint applications $C_1, \ldots, C_m$ from $\mathcal{F}$ on $\mathbf{x}, \mathbf{y}$ form an α-implementation of $f(\mathbf{x})$. Let g be any constraint from $\mathcal{F}$ and let k be the arity of g. Let $C_{m+1}, \ldots, C_{m+\alpha'-\alpha}$ be $\alpha' - \alpha$ applications of the constraint g on new variables $\mathbf{z} = \{z_1, \ldots, z_k\}$. Then the collection of constraints $C_1, \ldots, C_{m+\alpha'-\alpha}$ on variable set $\mathbf{x}, \mathbf{y}, \mathbf{z}$ form an α'-implementation of f. Furthermore, the transformation preserves strictness and perfectness. □

3.1. Reduction from strict implementations. Here we show how strict implementations are useful in establishing AP-reducibility among MAX CSP problems. However, first we need a simple statement about the approximability of MAX CSP problems.

PROPOSITION 3.7 (see [39]). *For every constraint family $\mathcal{F}$ there exists a constant k such that given any instance $\mathcal{I}$ of* WEIGHTED MAX CSP($\mathcal{F}$) *with constraints of total weight W, a solution satisfying constraints of weight W/k can be found in polynomial time.*

Proof. The proposition follows from the proof of Theorem 1 in [39] which shows the above for every MAX SNP problem. (Note, in particular, that a random assignment satisfies a constant fraction of WEIGHTED MAX CSP($\mathcal{F}$) instance, and such an assignment can be found deterministically by using the method of conditional probabilities.) □

LEMMA 3.8. *If $\mathcal{F}' \stackrel{s}{\Longrightarrow} \mathcal{F}$, then* MAX CSP($\mathcal{F}$) $\leq_{\text{AP}}$ MAX CSP($\mathcal{F}'$).

Proof. The reduction uses Proposition 3.7 above. Let β be a constant such that given an instance $\mathcal{I}$ of MAX CSP($\mathcal{F}$) with m constraints, an assignment satisfying $\frac{m}{\beta}$ constraints can be found in polynomial time.

Recall that we need to show polynomial time computable functions F and G such that F maps an instance $\mathcal{I}$ of MAX CSP($\mathcal{F}$) to an instance of MAX CSP($\mathcal{F}'$), and G maps a solution to $F(\mathcal{I})$ back to a solution of $\mathcal{I}$.

Given an instance $\mathcal{I}$ on n variables and m constraints, the mapping F simply replaces every constraint in $\mathcal{I}$ (which belongs to $\mathcal{F}$) with a strict α-implementation using constraints of $\mathcal{F}'$ for some constant α. (Notice that by Lemma 3.6 some such α does exist.) The mapping retains the original n variables of $\mathcal{I}$ as primary variables and uses m independent copies of the auxiliary variables—one independent copy for every constraint in $\mathcal{I}$.

Let $\langle \mathbf{x}, \mathbf{y} \rangle$ be an r-approximate solution to the instance $F(\mathcal{I})$, where $\mathbf{x}$ denotes the original variables of $\mathcal{I}$ and $\mathbf{y}$ denotes the auxiliary variables introduced by F. The mapping G uses two possible solutions and takes the better of the two: the first

solution is x; and the second solution x' is the solution which satisfies at least m/β of the constraints in $\mathcal{I}$. G outputs the solution which satisfies more constraints.

We now show that an r-approximate solution leads to an r'-approximate solution, where $r' \leq 1+\gamma(r-1)$ for some constant γ. Let OPT denote the value of the optimum to $\mathcal{I}$. Then the optimum of $F(\mathcal{I})$ is exactly $\text{OPT}+m(\alpha-1)$. This computation uses the fact that for every satisfied constraint in the optimal assignment to $\mathcal{I}$, we can satisfy α constraints of its implementation by choosing the auxiliary variables appropriately (from properties (1) and (2) of Definition 3.1); and for every unsatisfied constraint exactly $\alpha-1$ constraints of its implementation can be satisfied (from property (3) and strictness of the implementation). Thus the solution $\langle \mathbf{x}, \mathbf{y} \rangle$ satisfies at least $\frac{1}{r}(\text{OPT} + m(\alpha-1))$ constraints of $F(\mathcal{I})$. Thus $\mathbf{x}$ satisfies at least $\frac{1}{r}(\text{OPT}+m(\alpha-1)) - m(\alpha-1)$ constraints in $\mathcal{I}$. (Here we use properties (1) and (3) of Definition 3.1 to see that there must be at least $\frac{1}{r}(\text{OPT} + m(\alpha-1)) - m(\alpha-1)$ constraints of $\mathcal{I}$ in whose implementations exactly α constraints must be satisfied.) Thus the solution output by G satisfies at least

$$\max\left\{\frac{1}{r}(\text{OPT} + m(\alpha-1)) - m(\alpha-1), \frac{m}{\beta}\right\}$$

constraints. Using the fact that $\max\{a,b\} \geq \lambda a + (1-\lambda)b$ for any $\lambda \in [0,1]$ and using $\lambda = \frac{r}{r+\beta(\alpha-1)(r-1)}$, we lower bound the above expression by

$$\frac{\text{OPT}}{r+\beta(\alpha-1)(r-1)}.$$

Thus

$$r' \leq \frac{\text{OPT}}{\text{OPT}/(r+\beta(\alpha-1)(r-1))} = r+\beta(\alpha-1)(r-1) = 1+(\beta(\alpha-1)+1)(r-1).$$

Thus we find that G maps r-approximate solutions of $F(\mathcal{I})$ to $(1+\gamma(r-1))$-approximate solutions to $\mathcal{I}$ for $\gamma = \beta(\alpha-1)+1 < \infty$ as required. ∎

3.2. Reductions from perfect implementations. We now show how to use perfect implementations to get reductions. Specifically we obtain reductions among Weighted Max Ones, Weighted Min Ones, and Min CSP problems.

Lemma 3.9. *If $\mathcal{F}' \stackrel{p}{\Longrightarrow} \mathcal{F}$, then* Weighted Max Ones($\mathcal{F}$) *(*Weighted Min Ones($\mathcal{F}$)*) is AP-reducible to* Weighted Max Ones($\mathcal{F}'$) *(resp.,* Weighted Min Ones($\mathcal{F}'$)*).*

Proof. Again we need to show polynomial time computable functions F and G such that F maps an instance $\mathcal{I}$ of Weighted Max Ones($\mathcal{F}$) (Weighted Min Ones($\mathcal{F}$)) to an instance of Weighted Max Ones($\mathcal{F}'$) (Weighted Min Ones($\mathcal{F}$)), and G maps a solution to $F(\mathcal{I})$ back to a solution of $\mathcal{I}$.

Given an instance $\mathcal{I}$ on n variables and m constraints, the mapping F simply replaces every constraint in $\mathcal{I}$ (which belongs to $\mathcal{F}$) with a perfect α-implementation using constraints of $\mathcal{F}'$ for some constant α. (Notice that by Lemma 3.6 some such α does exist.) The mapping retains the original n variables of $\mathcal{I}$ as primary variables and uses m independent copies of the auxiliary variables—one independent copy for every constraint in $\mathcal{I}$. Furthermore, $F(\mathcal{I})$ retains the weight of the primary variables from $\mathcal{I}$ and associates a weight of zero with all the newly created auxiliary variables. Given a solution to $F(\mathcal{I})$, the mapping G is simply the projection of the solution back to the primary variables. It is clear that every feasible solution to $\mathcal{I}$ can be extended

into a feasible solution to $F(\mathcal{I})$ such that $\text{OPT}(\mathcal{I}) = \text{OPT}(F(\mathcal{I}))$. Furthermore, the mapping G maps feasible solutions to $F(\mathcal{I})$ into feasible solutions to $\mathcal{I}$ with the same objective. (This is where the perfectness of the implementations is being used.) Thus the optimum of $F(\mathcal{I})$ equals the value of the optimum of $\mathcal{I}$ and given an r-approximate solution to $F(\mathcal{I})$, the mapping G yields an r-approximate solution to $\mathcal{I}$. □

LEMMA 3.10. *If* $\mathcal{F}' \stackrel{p}{\Longrightarrow} \mathcal{F}$, *then* MIN CSP($\mathcal{F}$) $\leq_{\text{A}}$ MIN CSP($\mathcal{F}'$).

Proof. Let α be large enough so that any constraint from $\mathcal{F}$ has a perfect α-implementation using constraints from $\mathcal{F}'$. Let $\mathcal{I}$ be an instance of MIN CSP($\mathcal{F}$) and let $\mathcal{I}'$ be the instance of MIN CSP($\mathcal{F}'$) obtained by replacing each constraint of $\mathcal{I}$ with the respective α-implementation. Once again each implementation uses the original set of variables for its primary variables and uses its own independent copy of the auxiliary variables. Note that the optimum of $\mathcal{I}'$ may be as high as αo if o is the optimum of $\mathcal{I}$ (since the implementations are not strict). It is easy to check that any assignment for $\mathcal{I}'$ of cost V yields an assignment for $\mathcal{I}$ whose cost is between V/α and V. In particular, if the solution is an r-approximate solution to $\mathcal{I}'$, then $V \geq \frac{o}{\alpha r}$ and thus it induces a solution that is at least an (αr)-approximate solution to $\mathcal{I}$. (Note that if the implementations were strict, we would have obtained an AP-reduction by the above.) □

3.3. Weighted vs. unweighted problems. Lemma 3.9 crucially depends on its ability to work with weighted problems to obtain reductions. The following lemma shows that in most cases showing hardness for weighted problems is sufficient. Specifically it shows that as long as a problem is weakly approximable, its weighted and unweighted versions are equivalent. The result uses a similar result from Crescenzi, Silvestri, and Trevisan [15], who prove that for a certain class of problems in poly-APX that they term "nice," weighted problems AP-reduce to problems with polynomially bounded integral weights. (We include a sketch of their proof, specialized to our case for the sake of completeness.) Using this result we scale all weights down to small integers and then simulate the small integral weights by replication of clauses and/or variables. (We note that the little-oh slackness in the definition of AP-reduction is exploited in this step.)

LEMMA 3.11. *For every family* $\mathcal{F}$, *if* WEIGHTED MAX ONES($\mathcal{F}$) *is in* poly-APX, *then* WEIGHTED MAX ONES($\mathcal{F}$) *AP-reduces to* MAX ONES($\mathcal{F}$). *Analogous results hold for* MIN CSP($\mathcal{F}$), MAX CSP($\mathcal{F}$), *and* MIN ONES($\mathcal{F}$).

Proof. Fix a family $\mathcal{F}$. We first reduce WEIGHTED MAX ONES($\mathcal{F}$) to WEIGHTED MAX ONES($\mathcal{F}$) restricted to instances with polynomially bounded positive integer weights, provided WEIGHTED MAX ONES($\mathcal{F}$) is in poly-APX. This step uses a scaling idea as in [15, Theorem 4]. Essentially the same proof also works for the cases of WEIGHTED MAX CSP($\mathcal{F}$), WEIGHTED MIN CSP($\mathcal{F}$), or WEIGHTED MIN ONES($\mathcal{F}$). Given an instance $\mathcal{I} = (\mathbf{x}, \mathbf{C}, \mathbf{w})$ of WEIGHTED MAX ONES($\mathcal{F}$), we will define a new vector of weights $\mathbf{w}'$ and use this to define a new instance $\mathcal{I}' = (\mathbf{x}, \mathbf{C}, \mathbf{w}')$ of WEIGHTED MAX ONES($\mathcal{F}$) with polynomially bounded weights. Let A be an $p(n)$-approximation algorithm for WEIGHTED MAX ONES($\mathcal{F}$), and let t be the value of the solution returned by A on $\mathcal{I}$. We let $N = n^2(p(n))^2 + np(n)$, and let $w_i'' = \left\lfloor \frac{w_i \cdot N}{t} \right\rfloor + 1$, and finally let $w_i' = \min\{w_i'', N \cdot p(n) + 1\}$. It is clear that the weights w_i' are polynomially bounded. Furthermore, note that if $w_i' < w_i''$, then no feasible solution to $\mathcal{I}$ (or $\mathcal{I}'$) can have x_i set to 1, since any such solution would have value at least $w_i > t \cdot p(n)$, contradicting the assumption that A is a $p(n)$-approximation algorithm. Thus, in particular, we have $\text{OPT}(\mathcal{I}') \geq (N/t) \cdot \text{OPT}(\mathcal{I})$. Given an r-approximate solution s' to $\mathcal{I}'$ we return the better of the solutions s' and the solution

returned A as the solution to $\mathcal{I}$. It is clear that if $r \geq p(n)$, then the returned solution is still an r-approximate solution. Below we see that an r-approximate solution to $\mathcal{I}'$, with $r \leq p(n)$, is also an $(r + 1/n)$-approximate solution to $\mathcal{I}$ of value at least

$$\begin{aligned}(t/N) \cdot (\text{OPT}(\mathcal{I}')/r) - n) &\geq \text{OPT}(\mathcal{I})/r - (nt/N) \\ &\geq \text{OPT}(\mathcal{I})/r - (n \cdot \text{OPT}(\mathcal{I})/N) \\ &\geq \text{OPT}(\mathcal{I}) \left(\frac{1}{r} - \frac{1}{nr^2 + r}\right) \\ &= \text{OPT}(\mathcal{I})/(r + 1/n).\end{aligned}$$

This concludes the first step of the reduction. In the next step we give an AP-reduction from the class of problems with polynomially bounded weights to the unweighted case.

We start with the case of Weighted Max CSP($\mathcal{F}$) first. Given an instance of Weighted Max CSP($\mathcal{F}$) on variables $x_1, \ldots, x_n$, constraints $C_1, \ldots, C_m$, and polynomially bounded integer weights $w_1, \ldots, w_m$, we reduce it to the unweighted case by replication of constraints. Thus the reduced instance has variables $x_1, \ldots, x_n$ and constraint $\{\{C_i^j\}_{j=1}^{w_i}\}_{i=1}^m$, where constraint $C_i^j = C_i$. It is clear that the reduced instance is essentially the same as the instance we started with. Similarly we reduce Weighted Min CSP($\mathcal{F}$) to Min CSP($\mathcal{F}$).

Given an instance $\mathcal{I}$ of Weighted Max Ones($\mathcal{F}$) on variables $x_1, \ldots, x_n$, constraints $C_1, \ldots, C_m$, and weights $w_1, \ldots, w_n$, we create an instance $\mathcal{I}'$ of Max Ones($\mathcal{F}$) on variables $\{\{y_i^j\}_{j=1}^{w_i}\}_{i=1}^n$. For every constraint C_j of $\mathcal{I}$ of the form $f(x_{i_1}, \ldots, x_{i_k})$, and for every $j \in \{1, \ldots, k\}$ and $n_j \in \{1, \ldots, w_{i_j}\}$, we impose the constraints $f(y_{i_1}^{n_1}, \ldots, y_{i_k}^{n_k})$. We now claim that the reduced instance is essentially equivalent to the instance we started with. To see this, notice that given any feasible solution $\mathbf{y}$ to the instance $\mathcal{I}'$, we may convert it to another feasible solution $\mathbf{y}'$ in which, for every i, all the variables $\{(\mathbf{y}')_i^j | j = 1, \ldots, w_i\}$ have the same assignment by setting $(\mathbf{y}')_i^j$ to 1 if any of the variables $y_i^{j'}$, $j' = 1, \ldots, w_i$, is set to 1. Notice that this preserves feasibility and increases only the contribution to the objective function. The assignment $\mathbf{y}'$ now induces an assignment to $\mathbf{x}$ with the same value of the objective function. Thus the reduced instance is essentially equivalent to the original one. This concludes the reduction from Weighted Max Ones($\mathcal{F}$) to Max Ones($\mathcal{F}$). The reduction from Weighted Min Ones($\mathcal{F}$) to Min Ones($\mathcal{F}$) is similar. □

4. Characterizations: New and old. In this section we characterize some of the properties of functions that we study. Most of the properties are defined so as to describe how a function behaves if it exhibits the property. For the hardness results, however, we need to see how to exploit the fact that a function does not satisfy some given property. For this we would like to see some simple witness to the fact that the function does not have a given property. As an example consider the affineness property. If a function is affine, it is easy to see how to use this property. What will be important to us is whether there exists a simple witness to the fact that a function f is not affine. Schaefer [42] provides such a characterization: If a function is not affine, then there exist assignments s_1, s_2, and s_3 that satisfy f such that $s_1 \oplus s_2 \oplus s_3$ does not satisfy f. This is exploited by Schaefer in his classification theorem (and by us, in our classifications). In this section, we describe other such characterizations and the implementations that are obtained from them. First we introduce some more definitions and notations that we will be used in the rest of the paper.

4.1. Definitions and notations. For $s \in \{0,1\}^k$, we let $\bar{s} \in \{0,1\}^k$ denote the bitwise complement of s. For a constraint f of arity k, let f^- be the constraint

$f^-(s) = f(\bar{s})$. For a constraint family $\mathcal{F}$, let $\mathcal{F}^- = \{f^- : f \in \mathcal{F}\}$. For $s_1, s_2 \in \{0,1\}^k$, $s_1 \oplus s_2$ denotes the bitwise exclusive-or of the assignments s_1 and s_2. For $s \in \{0,1\}^k$, $Z(s)$ denotes the subset of indices $i \in [k]$, where s is zero and $O(s)$ denotes the subset of indices where s in one.

For a constraint f of arity k, $S \subseteq [k]$ and $b \in \{0,1\}$, $f|_{(S,b)}$ is the constraint of arity $k' = k - |S|$ defined as follows: For variables $x_{i_1}, \ldots, x_{i_{k'}}$, where $\{i_1, \ldots, i_{k'}\} = [k] - S$, we define $f|_{(S,b)}(x_{i_1}, \ldots, x_{i_{k'}}) = f(x_1, \ldots, x_k)$, where $x_i = b$ for $i \in S$. We will sometimes use the notation $f|_{(i,b)}$ to denote the function $f|_{(\{i\},b)}$. For a constraint family $\mathcal{F}$, the family $\mathcal{F}|_0$ is the family $\{f|_{S,0} | f \in \mathcal{F}, S \subseteq [\text{arity}(f)]\}$. The family $\mathcal{F}|_1$ is defined analogously. The family $\mathcal{F}|_{0,1}$ is the family $(\mathcal{F}|_0)|_1$ (or equivalently the family $(\mathcal{F}|_1)|_0$).

DEFINITION 4.1 (C-closed). *A constraint f is C-closed (complementation-closed) if for every assignment s, $f(s) = f(\bar{s})$.*

DEFINITION 4.2 (existential zero/existential one). *A constraint f is an existential zero constraint if $f(\mathbf{0}) = 1$ and $f(\mathbf{1}) = 0$. A constraint f is an existential one constraint if $f(\mathbf{0}) = 0$ and $f(\mathbf{1}) = 1$.*

The terminology above is motivated by the fact that an existential zero constraint application $f(x_1, \ldots, x_k)$ forces at least one of the variables to be zero (while an all zero assignment definitely satisfies the application).

Every constraint f can be expressed as the conjunction of disjuncts. This representation of a function is referred to as the conjunctive normal form (CNF) representation of f. Alternately, a function can also be represented as a disjunction of conjuncts and this representation is called the disjunctive normal form (DNF) representation.

A partial setting to the variables of f that fixes the value of f to 1 is called a *term* of f. A partial setting that fixes f to 0 is called a *clause* of f. We refer to the terms and clauses in a functional form, i.e., we say $\text{OR}_{3,1}(x_1, x_2, x_3) = x_1 \bigvee x_2 \bigvee \neg x_3$ is a clause of $f(x_1, \ldots, x_p)$ if setting $x_1 = x_2 = 0$ and $x_3 = 1$ fixes f to being 0. Similarly we use the $\text{AND}_{i,j}$ to denote the terms. Notice that a DNF (CNF) representation of f can be obtained by expressing as the conjunction (disjunction) of its terms (clauses).

DEFINITION 4.3 (minterm/maxterm). *A partial setting to a subset of the variables of f is a minterm if it is a term of f and no restriction of the setting to any strict subset of the variables fixes the value of f. Analogously a clause of f is a maxterm if it is a minimal setting to the variables of f so as to fix its value to* 0.

As in the case of terms and clauses, we represent minterms and maxterms functionally, i.e., using $\text{OR}_{i,j}$ and $\text{AND}_{i,j}$.

DEFINITION 4.4 (basis). *A constraint family $\mathcal{F}'$ is a basis for a constraint family $\mathcal{F}$ if any constraint of $\mathcal{F}$ can be expressed as a conjunction of constraints drawn from $\mathcal{F}'$.*

Thus, for example, the basis for affine constraints is the set $\{\text{XOR}_p | p \geq 1\} \cup \{\text{XNOR}_p | p \geq 1\}$. The basis for width-2 affine constraints is the set $\mathcal{F} = \{\text{XOR}, \text{XNOR}, T, F\}$, and the basis for 2CNF constraints is the set $\mathcal{F} = \{\text{OR}_{2,0}, \text{OR}_{2,1}, \text{OR}_{2,2}, T, F\}$. The definition of a basis is motivated by the fact that if $\mathcal{F}'$ is a basis for $\mathcal{F}$, then $\mathcal{F}'$ can perfectly implement every function in $\mathcal{F}$ (see Proposition 3.4).

4.2. 0-validity and 1-validity. The characterization of 0-valid and 1-valid functions is obvious. We now show what can be implemented with functions that are not 0-valid and not 1-valid.

LEMMA 4.5. *Let f be a nontrivial constraint which is C-closed and is not* 0*-valid (or equivalently not* 1*-valid).*[5] *Then* $\{f\} \stackrel{s/p}{\Longrightarrow}$ XOR.

Proof. Let k denote the arity of f and let k_0 and k_1, respectively, denote the maximum number of 0's and 1's in any satisfying assignment for f; clearly $k_0 = k_1$. Now let $S_x = \{x_1, \ldots, x_{3k}\}$ and $S_y = \{y_1, \ldots, y_{3k}\}$ be two disjoint sets of $3k$ variables each. In the first phase of the proof, we place a large number of constraints on the variables of S_x and S_y that ends up implementing perfectly, but not necessarily strictly, the constraints $\mathrm{XOR}(x_i, y_j)$ for every i and j. In the second phase, we will introduce two new variables x and y and augment the constraints so as to implement the constraint $\mathrm{XOR}(x, y)$ perfectly and strictly.

We start by placing the constraint f on a large collection of inputs as follows: For every satisfying assignment s, we place $\binom{3k}{i}\binom{3k}{k-i}$ constraints on the variable set $S_x \cup S_y$ such that every i-variable subset of S_x appears in place of 0's in s and every $(k-i)$ variable subset of S_y appears in place of 1's in the assignment s, where i denotes the number of 0's in s. Let this collection of constraints be denoted by $\mathcal{I}$. We will first show that $\mathcal{I}$ gives a perfect (but possibly nonstrict) implementation of the constraint $\mathrm{XOR}(x_i, y_j)$.

Clearly, any solution which assigns identical values to all variables in S_x and the complementary value to all variables in S_y satisfies all the constraints in $\mathcal{I}$. We will show the converse, i.e., every assignment satisfying all the above constraints assigns identical values to all variables in S_x and the complementary value to every variable in S_y.

Fix any assignment satisfying all the constraints and let Z and O, respectively, denote the set of variables set to 0 and 1, respectively. We claim that any solution which satisfies all the constraints must satisfy either $Z = S_x$ and $O = S_y$ or $Z = S_y$ and $O = S_x$.

Note first that at least one of the conditions $|S_x \cap Z| \geq k$ or $|S_x \cap O| \geq k$ must hold. Consider the case where $|S_x \cap Z| \geq k$. In this case, we will show that $S_x = Z$ and $S_y = O$. (A similar argument for the other case will show $S_x = O$ and $S_y = Z$.)

- First we claim that $|S_y \cap Z| < k$ and thus $|S_y \cap O| > 2k$. Assume for contradiction that $|S_y \cap Z| \geq k$. Then there exists a constraint application in $\mathcal{I}$ with all its input variables coming from the sets $S_x \cap Z$ and $S_y \cap Z$. By definition of Z all these variables are set to zero, and hence this constraint application is unsatisfied (by the 0-validity of f).
- Next we claim that every variable of S_x is set to 0: Assume otherwise and, without loss of generality (w.l.o.g.), let x_1 be set to 1. Let s be an assignment with minimal number of 0's. Assume w.l.o.g. that $s = 0^{k_0}1^{k-k_0}$. W.l.o.g., let $y_1, \ldots, y_{2k}$ be set to one. (We know $2k$ such variables exist since $|S_y \cap O| > 2k$.) By our choice of constraint applications, $f(x_1, \ldots, x_{k_0}, y_1, \ldots, y_{k-k_0})$ is one of the constraint applications. However, at most $k_0 - 1$ variables of this constraint are set to 0 and thus this application cannot be satisfied.
- Finally, similar to the above step, we can show that every variable in S_y is set to 1.

Thus we have shown that if $|S_x \cap Z| \geq k$, then $S_x = Z$ and $S_y = O$. The other case is similar, and this concludes the first phase.

We next augment the collection of constraints above as follows. Consider a least Hamming weight satisfying assignment s for f. W.l.o.g., we assume that $s =$

[5] Notice that C-closedness implies that f is 0-valid if and only if it is 1-valid.

$10^{k-k_1-1}1^{k_1}$. We add the constraints $f(x, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$ and $f(y, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$. We now argue that the resulting collection of constraints yields a perfect and strict implementation of the constraint $\mathrm{XOR}(x, y)$.

Clearly $s' = 0^{k-k_1}1^{k_1}$ is not a satisfying assignment (since it has smaller Hamming weight than s). Since f is C-closed, we have the following situation:

		$\overbrace{\quad}^{k-k_1-1}$	$\overbrace{\quad}^{k_1}$	$f()$
s'	0	00...0	11...1	0
s	1	00...0	11...1	1
$\bar{s}$	0	11...1	00...0	1
$\bar{s'}$	1	11...1	00...0	0

If $x = 1$, then to satisfy the first of the two constraints (in addition to all the earlier constraints) above, we must have $Z = S_x$, $O = S_y$ and thus must have $y = 0$. Similarly if $x = 0$, then we must have $O = S_x$, $Z = S_y$ and $y = 1$. Thus the given constraints do form a perfect implementation of $\mathrm{XOR}(x, y)$. Finally if $x = y$, then the setting $O = S_x$ and $Z = S_y$ satisfies all constraints except one (which is one of the last two additional constraints). Thus the implementation satisfies the strictness property as well. □

LEMMA 4.6. *Let f_0, f_1 and g be nontrivial constraints, possibly identical, which are not 0-valid, not 1-valid, and not C-closed, respectively. Then $\{f_0, f_1, g\} \overset{s/p}{\Longrightarrow} \{T, F\}$.*

Proof. We will describe only the implementation of constraint $T(\cdot)$; the implementation for the constraint $F(\cdot)$ is identical.

Assume, for simplicity, that all the three functions f_0, f_1, and g are of arity k. We use an implementation similar to the one used in the proof of Lemma 4.5. To implement $T(x)$, we use a set of $6k$ auxiliary variables $S_x = \{x_1, \ldots, x_{3k}\}$ and $S_y = \{y_1, \ldots, y_{3k}\}$. For each $h \in \{f_0, f_1, g\}$, for each satisfying assignment s of h, if j is the number of 0's in s we place the $\binom{3k}{j}\binom{3k}{k-j}$ constraints h with all possible subsets of S_x appearing in the indices in $Z(s)$ and all possible subsets of S_y appearing in $O(s)$. Finally we introduce one constraint involving the primary variable x. Let s be the satisfying assignment of minimum Hamming weight which satisfies f_0. Notice that s must include at least one 1. Assume, w.l.o.g., that $s = 10^{k-k_1-1}1^{k_1}$. Then we introduce the constraint application $f_0(x, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$.

It is clear that by setting all variables in S_x to 0 and all variables in S_y to 1 we get an assignment that satisfies all constraints except possibly the last constraint (which involves x). Furthermore, the last constraint is satisfied if and only if $x = 1$. Thus, to prove the lemma, it suffices to show that any solution which satisfies all the constraints above must set x to 1, all variables in S_x to 0, and all variables in S_y to 1.

Fix an assignment satisfying all the constraints. Let O be the set of variables in $S_x \cup S_y$ set to 1 and Z be the set of variables set to 0. We need to show that $S_x \cap O = \emptyset$ and we do so in stages.

- First, we consider the possibility $|S_x \cap O| \geq k$. We consider two cases.
  - Case. $|S_y \cap Z| \geq k$. Consider a satisfying assignment s such that $g(\bar{s}) = 0$. Such an assignment must exist since g is not C-closed. Note that the constraint applications include at least one where g is applied to variables where the positions corresponding to $O(s)$ come from $S_y \cap Z$ and positions corresponding to $Z(s)$ come from $S_x \cap O$. However, this constraint is not satisfied by the assignment (since $g(\bar{s}) = 0$).

– Case. $|S_y \cap O| > 2k$. Let s_1 be a satisfying assignment for f_1. Note that the application of the constraint f_1 with the positions corresponding to $O(s)$ coming from $S_y \cap O$ and the positions corresponding to $Z(s)$ coming from $S_x \cap O$ is one of the constraints imposed above and is not satisfied (since f_1 is not 1-valid).

Thus in either case, we find a constraint that is not satisfied and thus this possibility ($|S_x \cap O| \geq k$) cannot occur. Thus we conclude $|S_x \cap O| < k$.

- From the above, we have $|S_x \cap Z| > 2k$. If $|S_y \cap Z| \geq k$, then we can find an application of the constraint f_0 to the variables in the set Z that will not be satisfied. Thus we have $|S_y \cap Z| < k$ and thus $|S_y \cap O| > 2k$. This can now be used to conclude that $S_y \cap Z = \phi$ as follows. Consider a satisfying assignment with smallest number of ones. The number of ones in such an assignment is positive since f_0 is not 0-valid. If we consider all the constraints corresponding to this assignment with inputs from S_y and $S_x \cap Z$ only, it is easy to see that there will be at least one unsatisfied constraint if $S_y \cap Z \neq \phi$. Hence each variable in S_y is set to 1 in this case. Finally, using the constraints on the constraint f_1 which is not 1-valid, it is easy to conclude that in fact $Z = S_x$.

Having concluded that $S_x = Z$ and $S_y = O$, it is easy to see that the constraint $f_0(x, x_1, \ldots, x_{k-k_1-1}, y_1, \ldots, y_{k_1})$ is satisfied only if $x = 1$. Thus the set of constraints imposed above yields a strict and perfect implementation of $T(\cdot)$. The constraint $F(\cdot)$ can be implemented analogously. □

For the CSP classes, it suffices to consider the case when $\mathcal{F}$ is neither 0-valid nor 1-valid. For the MAX ONES and MIN ONES classes we also need to consider the case when $\mathcal{F}$ fails only to have one of these two properties. Therefore keeping these classes in mind we prove the following lemma, which shows how to obtain a weak version of T and F in these cases.

LEMMA 4.7. *If $\mathcal{F}$ is not C-closed and not* 1*-valid, then $\mathcal{F} \overset{s/p}{\Longrightarrow} f$ for some existential zero constraint f_0. Analogously, if $\mathcal{F}$ is not C-closed and not* 0*-valid, then $\mathcal{F} \overset{s/p}{\Longrightarrow} f_1$ for some existential one constraint f_1.*

Proof. We prove only the first part of the lemma. The second part is similar.

The proof reduces to two simple subcases. Let $f \in \mathcal{F}$ be a constraint that is not 1-valid. If f is 0-valid, then we are done since f is an existential zero constraint. If f is not 0-valid, then $\mathcal{F}$ has a non-C-closed function, a non-0-valid function, and a non-1-valid function, and hence by Lemma 4.6, $\mathcal{F}$ perfectly and strictly implements F which is an existential zero function. □

4.3. 2-monotone functions.

DEFINITION 4.8 (0/1-term). *A set $V \subseteq \{1, \ldots, k\}$ is a* 0-term (1-term) *for a k-ary constraint f if every assignment s with $Z(s) \supseteq V$ (resp., $O(s) \supseteq V$) is a satisfying assignment for f.*

The choice of the name reflects the fact that a 0-term is a term consisting of all negated variables (or variables set to 0) and a 1-term consists of all positive variables.

LEMMA 4.9. *A constraint f is a* 2*-monotone constraint if and only if all the following conditions are satisfied:*

(a) *for every satisfying assignment s of f, either $Z(s)$ is a* 0*-term or $O(s)$ is a* 1*-term;*

(b) *if V_1 and V_2 are* 1*-terms for f, then $V_1 \cap V_2$ is a* 1*-term; and*

(c) *if V_1 and V_2 are* 0*-terms for f, then $V_1 \cap V_2$ is also a* 0*-term.*

Proof. Recall that a 2-monotone constraint is one that can be expressed as a

disjunction of two terms. Every satisfying assignment must satisfy one of the two terms and this gives property (a). Properties (b) and (c) are obtained from the fact that the constraint has at most one term with all positive literals and at most one term with all negated literals.

Conversely, consider a constraint f which satisfies properties (a)–(c). Let $s_1, \ldots, s_l$ be the satisfying assignments of f such that $Z(s_i)$ is a 0-term for $i \in \{1, \ldots, l\}$. Let $t_1 \ldots, t_k$ be the satisfying assignments of f such that $O(t_j)$ is a 1-term for $j \in \{1, \ldots, k\}$. Then $Z = \cap_i Z(s_i)$ is a 0-term and $O = \cap_j O(t_j)$ is a 1-term for f, respectively (using (b) and (c)), and together they cover all satisfying assignments of f. Thus $f(\mathbf{x}) = (\wedge_{i \in Z} \neg x_i) \vee (\wedge_{j \in O} x_j)$, which is 2-monotone. □

We now use the characterization above to prove, in Lemma 4.11, that if a function f is not 2-monotone, then the family $\{f, T, F\}$ implements the function XOR. We first prove a simple lemma which shows implementations of XOR by some specific constraint families. This will be used in Lemma 4.11.

LEMMA 4.10.

(1) $\{\mathrm{AND}_{2,1}\} \stackrel{s}{\Longrightarrow} \mathrm{XOR}$.

(2) *For every* $p \geq 2$, *we have* $\{f_p, T, F\} \stackrel{s/p}{\Longrightarrow} \mathrm{XOR}$, *where* $f_p(x_1, \ldots, x_p) = \mathrm{OR}_p(x_1, \ldots, x_p) \bigwedge \mathrm{NAND}_p(x_1, \ldots, x_p)$.

(3) *For every* $p \geq 2$, *we have* $\{\mathrm{NAND}_p, T, F\} \stackrel{s}{\Longrightarrow} \mathrm{XOR}$.

Proof. For part (1) we observe that the constraints

$$\{\mathrm{AND}_{2,1}(x_1, x_2), \mathrm{AND}_{2,1}(x_2, x_1)\}$$

provide a strict (but not perfect) 1-implementation of $\mathrm{XOR}(x_1, x_2)$.

For part (2) notice that the claim is trivial if $p = 2$, since the function $f_2 = \mathrm{XOR}$. For $p \geq 3$, the constraints $\{f_p(x_1, \ldots, x_p), T(x_3), \ldots, T(x_p)\}$ perfectly and strictly implement $\mathrm{NAND}(x_1, x_2)$. Similarly the constraints $\{f_p(x_1, \ldots, x_p), F(x_3), \ldots, F(x_p)\}$ perfectly and strictly implement the constraint $\mathrm{OR}(x_1, x_2)$. Finally the constraints $\mathrm{OR}(x_1, x_2)$ and $\mathrm{NAND}(x_1, x_2)$ perfectly and strictly implement the constraint XOR (x_1, x_2). Part (2) follows from the fact that perfect and strict implementations compose (Lemma 3.5).

Finally for part (3), we first use the constraints

$$\{\mathrm{NAND}_p(x_1, \ldots, x_p), F(x_3), \ldots, F(x_p)\}$$

to implement, strictly and perfectly, the constraint $\mathrm{NAND}(x_1, x_2)$. Now we may use $\{\mathrm{NAND}(x_1, x_2), \mathrm{NAND}(x_1.x_2), T(x_1), T(x_2)\}$ to obtain a 3-implementation of the constraint $\mathrm{XOR}(x_1, x_2)$. (Note that in the case the implementation is not perfect.) □

LEMMA 4.11. *Let* f *be a constraint which is not* 2*-monotone. Then* $\{f, T, F\} \stackrel{s}{\Longrightarrow}$ XOR.

Proof. The proof is divided into three cases which depend on which of the three conditions defining 2-monotonicity is violated by f. We first state and prove the claims.

CLAIM 4.12. *If* f *is a function violating property* (a) *of Lemma* 4.9, *then* $\{f, T, F\} \stackrel{s}{\Longrightarrow}$ XOR.

Proof. There exists some assignment s satisfying f, and two assignments s_0 and s_1 such that $Z(s) \subseteq Z(s_0)$ and $O(s) \subseteq O(s_1)$, such that $f(s_0) = f(s_1) = 0$. Rephrasing slightly, we know that there exists a triple (s_0, s, s_1) with the following properties:

$$f(s_0) = f(s_1) = 0; f(s) = 1; \; Z(s_0) \supseteq Z(s) \supseteq Z(s_1). \tag{1}$$

Note that the condition $Z(s_0) \supseteq Z(s) \supseteq Z(s_1)$ implies that $O(s_0) \subseteq O(s) \subseteq O(s_1)$. We call property (1) the "sandwich property." Of all triples satisfying the sandwich property, pick one that minimizes $|Z(s_0) \cap O(s_1)|$.

W.l.o.g., assume that $Z(s_0) \cap O(s_1) = \{1, \ldots, p\}$, $Z(s_0) \cap Z(s_1) = \{p+1, \ldots, q\}$, and $O(s_0) \cap O(s_1) = \{q+1, \ldots, k\}$. (Notice that the sandwich property implies that $O(s_0) \cap Z(s_1) = \emptyset$.) Let f_1 be the constraint given by $f_1(x_1, \ldots, x_p) = f(x_1, \ldots, x_p, 0, \ldots, 0, 1, \ldots, 1)$. Notice that the constraint applications $f(x_1 \ldots x_k)$ and $T(x_i)$ for every $i \in O(s_0) \cap O(s_1)$ and $F(x_i)$ for every $i \in Z(s_0) \cap Z(s_1)$ implement the function f_1. Thus it suffices to show that $\{f_1, T, F\}$ implements XOR.

Below we examine some properties of the constraint f_1. We will use the characters t, t', t_i, t_i' to denote assignments to f_1, while we use the characters s, s', s_i, s_i' to denote assignments to f. Note that

(1) $f_1(\mathbf{0}) = f_1(\mathbf{1}) = 0$.
(2) f_1 has a satisfying assignment. Thus p (the arity of f_1) is at least 2.
(3) If $f_1(t_1) = 0$ for some $t \neq \mathbf{1}$, then for every assignment t such that $Z(t) \supseteq Z(t_1)$, it is the case that $f_1(t_1) = 0$: This follows from the minimality of $|Z(s_0) \cap O(s_1)|$ above. If not, then consider the assignments $s_0' = s_0$, $s' = t0^{q-p}1^{k-q}$, and $s_1' = t_1 0^{q-p} 1^{k-q}$. The triple (s_0', s', s_1') also satisfies the sandwich property and has a smaller value of $|Z(s_0') \cap O(s_1')|$.
(4) If $f_1(t_0) = 0$ for some $t_0 \neq \mathbf{0}$, then for every assignment t such that $O(t) \supseteq O(t_0)$, it is the case that $f_1(t) = 0$ (again from the minimality of $|Z(s_0) \cap O(s_1)|$).

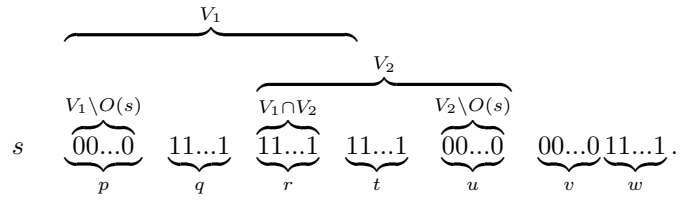
These properties of f_1 now allow us to identify f_1 almost completely. We show that either (a) $p = 2$ and $f_1(x_1 x_2)$ is either $\mathrm{AND}_{2,1}(x_1, x_2)$ or $\mathrm{AND}_{2,1}(x_2, x_1)$, or (b) f is satisfied by every assignment other than the all zeroes assignment and the all ones assignment. In either case $\{f_1, T, F\}$ strictly implements XOR by Lemma 4.10, parts (1) and (2). (Note that part (1) of Lemma 4.10 yields only a strict (but not perfect) implementation.) Thus proving that either (a) or (b) holds concludes the proof of the claim.

Suppose (b) is not the case. Thus, f_1 is left unsatisfied by some assignment t and $t \neq \mathbf{0}$ and $t \neq \mathbf{1}$. Then we will show that the only assignment that can satisfy f_1 is $\bar{t}$. However, this implies that t, $\bar{t}$, $\mathbf{0}$, and $\mathbf{1}$ are the only possible assignments to f_1, implying p must be 2, thereby yielding that (a) is true. Thus it suffices to show that if $f_1(t) = 0$, and $t' \neq \bar{t}$, then $f_1(t') = 0$. Since t' is not the bitwise complement of t, there must exist some input variable which shares the same assignment in t and t'. W.l.o.g. assume this is the variable x_1. Consider the case that this variable takes on the value 0 in the assignment t. Then we claim that the assignment $f_1(01 \ldots 1) = 0$. This is true since $O(01 \ldots 1) \supseteq O(t)$. Now notice that $f(t') = 0$ since $Z(t') \supseteq Z(01 \ldots 1)$. (In the case that the first variable takes on the value 1 in the assignment t, it is symmetric.) Thus we conclude that either (a) or (b) always holds and this concludes the proof of the claim. □

CLAIM 4.13. *Suppose f violates property* (b) *of Lemma* 4.9. *Then* $\{f, T, F\} \overset{s/p}{\Longrightarrow}$ XOR.

Proof. Let V_1 and V_2 be two 1-terms such that $V_1 \cap V_2$ is not a 1-term. Thus, there exists an assignment s such that (s.t.) $O(s) \supseteq V_1 \cap V_2$ and $f(s) = 0$. Among all such assignments let s be the one with the maximum number of 1's. The situation

looks as shown below:



In other words $s = 0^p 1^{q+r+t} 0^{u+v} 1^w$ and $f(s) = 0$. Furthermore, every assignment of the form $1^{p+q+r} *^{t+u+v+w}$ satisfies f and every assignment of the form $*^{p+q} 1^{r+t+u} *^{v+w}$ satisfies f (where the $*$'s above can be replaced by any of 0/1 independently). In particular this implies that $p, u \geq 1$. Consider the function f_1 on $p + u \geq 2$ variables obtained from f by restricting the variables in $O(s)$ to 1 and restricting the variables in $Z(s) - (V_1 \cup V_2)$ to 0. Notice that the constraint applications $f(x_1 \dots x_k)$, $T(x_i)$ for $i \in O(s)$ and $F(x_i)$ for $i \in Z(s) - (V_1 \cup V_2)$ strictly implement f_1. Thus it suffices to show that $\{f_1, T, F\}$ implements XOR. We do so by observing that $f_1(x_1 \dots x_{p+u})$ is the function NAND_{p+u}. Notice that $f_1(\mathbf{0}) = 0$. Furthermore, if $f_1(t) = 0$ for any other assignment t, then it contradicts the maximality of the number of 1's in s. The claim now follows from Lemma 4.10, part (3), which shows that the family $\{\text{NAND}_{p+u}, T, F\}$ implements XOR, provided $p + u \geq 2$. □

CLAIM 4.14. *Suppose f violates property* (c) *of Lemma* 4.9. *Then* $\{f, T, F\} \overset{s/p}{\Longrightarrow}$ XOR.

Proof. The proof is similar to the proof of the claim above. □

The lemma now follows from the fact that any constraint f_2 that is not 2-monotone must violate one of the properties (a), (b), or (c) from Lemma 4.9. □

4.4. Affine functions.

LEMMA 4.15 (see [42]). *f is an affine function if and only if for every three satisfying assignments s_1, s_2, and s_3 to f, $s_1 \oplus s_2 \oplus s_3$ is also a satisfying assignment.*

We first prove a simple consequence of the above which gives a slightly simpler sufficient condition for a function to be affine.

COROLLARY 4.16. *If f is not affine, then there exist two satisfying assignments s_1 and s_2 for f such that $s_1 \oplus s_2$ does not satisfy f.*

Proof. Assume otherwise. Then for any three satisfying assignments s_1, s_2, and s_3, we have that $f(s_1 \oplus s_2) = 1$, and hence $f((s_1 \oplus s_2) \oplus s_3) = 1$, thus yielding that f is affine. □

LEMMA 4.17. *If f is an affine constraint, then any function obtained by restricting some of the variables of f to constants and existentially quantifying over some other set of variables is also affine.*

Proof. We use Lemma 4.15 above. Let f_1 be a function derived from f as above. Consider any three assignments s_1', s_2', and s_3' which satisfy f_1. Let s_1, s_2, and s_3 be the respective extensions which satisfy f. Then the assignment $s_1 \oplus s_2 \oplus s_3$ extends $s_1' \oplus s_2' \oplus s_3'$ and satisfies f. Thus $s_1' \oplus s_2' \oplus s_3'$ satisfies f_1. Thus (using Lemma 4.15) again, we find that f_1 is affine. □

LEMMA 4.18. *If f is an affine function which is not of width-2, then* $\{f\} \overset{s/p}{\Longrightarrow}$ XOR_p *or* $\{f\} \overset{s/p}{\Longrightarrow} \text{XNOR}_p$ *for some $p \geq 3$.*

Proof. Let k be the arity of f. Define a dependent set of variables to be a set of variables $S \subseteq \{1, \dots, k\}$ such that not every assignment to the variables in S extends to a satisfying assignment of f. A dependent set S is a minimally dependent set if

no strict subset $S' \subset S$ is a dependent set. Notice that f can be expressed as the conjunction of constraints on its minimally dependent sets. Thus if f is not of width-2, then it must have a minimally dependent set S of cardinality at least 3. Assume $S = \{1, \ldots, p\}$, where $p \geq 3$. Consider the function

$$f_1(x_1 \ldots x_p) = \exists x_{p+1}, \ldots, x_k \text{ s.t. } f(x_1, \ldots x_k).$$

f_1 is affine (by Lemma 4.17), is not satisfied by every assignment, and has at least 2^{p-1} satisfying assignments. Thus f_1 has exactly 2^{p-1} assignments (since the number of satisfying assignments must be a power of 2). Thus f_1 is described by exactly one linear constraint and by the minimality of S this must be the constraint $\mathrm{XOR}(x_1 \ldots x_p)$ or the constraint $\mathrm{XNOR}(x_1 \ldots x_p)$. ∎

4.5. Horn Clauses, 2CNF, and IHS.

LEMMA 4.19. *If f is a weakly positive (weakly negative/IHS-B+/IHS-B−/*2CNF*) constraint, then any function obtained by restricting some of the variables of f to constants and existentially quantifying over some other set of variables is also weakly positive (resp., weakly negative/IHS-B+/IHS-B−/*2CNF*).*

Proof. It is easy to see that f remains weakly positive (weakly negative/IHS-$B+$/IHS-B−/2CNF) when some variable is restricted to a constant. Hence it suffices to consider the case where some variable y is quantified existentially. (Combinations of the possibilities can then be handled by a simple induction.) Thus consider the function $f_1(x_1, \ldots, x_k) \stackrel{\text{def}}{=} \exists y \text{ s.t. } f(x_1, \ldots, x_k, y)$. Let

$$f(x_1, \ldots, x_k, y) = \left(\bigwedge_{j=1}^{m} C_j(\bar{x}) \right) \bigwedge \left(\bigwedge_{j_0=1}^{m_0} (C^0_{j_0}(\bar{x}) \bigvee y) \right) \bigwedge \left(\bigwedge_{j_1=1}^{m_1} (C^1_{j_1}(\bar{x}) \bigvee \neg y) \right)$$

be a CNF expression for f which shows it is weakly positive (weakly negative/IHS-$B+$/IHS-B−/2CNF), where the clauses C_j, $C^0_{j_0}$, and $C^1_{j_1}$ involve literals on the variables $x_1, \ldots, x_k$.

We first show a simple transformation which creates a CNF expression for f_1. Later we show that f_1 inherits the appropriate properties of f.

Define $m_0 \times m_1$ clauses $C^{01}_{j_0 j_1}(\bar{x}) \stackrel{\text{def}}{=} C^0_{j_0}(\bar{x}) \bigvee C^1_{j_1}(\bar{x})$. Next, we note that $f_1(\bar{x})$ can be expressed as follows:

$$\begin{aligned} f_1(\bar{x}) &= f_1(\bar{x}, 0) \bigvee f_1(\bar{x}, 1) \\ &= \left(\left(\bigwedge_j C_j(\bar{x}) \right) \bigwedge \left(\bigwedge_{j_0} C^0_{j_0}(\bar{x}) \right) \right) \bigvee \left(\left(\bigwedge_j C_j(\bar{x}) \right) \bigwedge \left(\bigwedge_{j_1} C^1_{j_1}(\bar{x}) \right) \right) \\ &= \left(\bigwedge_j C_j(\bar{x}) \right) \bigwedge \left(\left(\bigwedge_{j_0} C^0_{j_0}(\bar{x}) \right) \bigvee \left(\bigwedge_{j_1} C^1_{j_1}(\bar{x}) \right) \right) \\ &= \left(\bigwedge_j C_j(\bar{x}) \right) \bigwedge \left(\bigwedge_{j_0} \bigwedge_{j_1} C^{01}_{j_0 j_1}(\bar{x}) \right). \end{aligned} \tag{2}$$

To conclude we need to verify that the right-hand side of (2) satisfies the same properties as f. Furthermore, we have only to consider clauses of the form $C^{01}_{j_0 j_1}(\bar{x})$ since all other clauses are directly from the expression for f. We verify this below:

- If f is weakly positive, then the clause $C^0_{j_0}$ involves at most one negated variable, and the clause $C^1_{j_1}$ involves no negated variable (since the clause participating in f is $(C^1_{j_1}(\bar{x}) \bigvee \neg y)$ which has a negated y involved in it). Thus the clause defining $C^{01}_{j_0 j_1}$ also has at most one negated variable.
- Similarly if f is weakly negative, then the clauses $C^{01}_{j_0 j_1}$ has at most one positive literal.
- If f is 2CNF, then the clauses $C^0_{j_0}$ and $C^1_{j_1}$ are of length 1, and hence the clause $C^{01}_{j_0 j_1}$ is of length at most 2.
- If f is IHS-$B+$, then the clause $C^0_{j_0}$ either has only one literal which is negated or has only positive literals. Furthermore, $C^1_{j_1}$ has at most one positive literal. Thus $C^{01}_{j_0 j_1}$ either has only positive literals or has at most two literals, one of which is negated. Hence $C^{01}_{j_0 j_1}$ is also IHS-$B+$.
- Similarly if f is IHS-$B-$, then the clause $C^{01}_{j_0 j_1}$ is also IHS-$B-$.

This concludes the proof of the lemma. □

LEMMA 4.20. *f is a weakly positive (weakly negative) constraint if and only if all its maxterms are weakly positive (weakly negative).*

Proof. We prove the lemma for the weakly positive case. The other case is similar. For the easy direction, recall that a function can be expressed as the conjunction of all its maxterms. If all maxterms are weakly positive, then this gives a weakly positive representation of f.

For the other direction, assume for contradiction that f is a weakly positive constraint that has $C = \neg x_1 \bigvee \cdots \bigvee \neg x_p \bigvee x_{p+1} \bigvee \cdots \bigvee x_q$ as a maxterm for some $p \geq 2$. Let the arity of f be k. Consider the function

$$f_1(x_1 x_2) \stackrel{\text{def}}{=} \exists x_{q+1}, \ldots, x_k \text{ s.t. } f(x_1 x_2 1^{p-2} 0^{q-p} x_{q+1} \ldots x_k).$$

Since C is an admissible clause in a CNF representation of f, we have that if we set $x_1, \ldots, x_p$ to 1 and setting $x_{p+1}, \ldots, x_q$ to 0, then no assignment to $x_{q+1}, \ldots, x_k$ satisfies f. Thus we find that $f_1(11) = 0$. By the fact that the clause is a maxterm we have that both the assignments $x_1 \ldots x_q = 01^{p-1}0^{q-p}$ and $x_1 \ldots x_q = 101^{p-2}0^{q-p}$ can be extended to satisfying assignments of f. Thus we find that $f_1(10) = f_1(01) = 1$. Thus f_1 is either the function NOR or XOR. It can be easily verified that neither of these is 2-monotone. (Every basic weakly positive function on two variables is unsatisfied on at least one of the two assignments 01 or 10.) However, this is in contradiction to Lemma 4.19 that showed that every function obtained by restricting some variables of f to constants and existentially quantifying over some others should yield a weakly positive function. □

LEMMA 4.21. *f is a* 2CNF *constraint if and only if all its maxterms are* 2CNF.

Proof. The "if" part is obvious. For the other direction we use Lemma 4.19. Assume for contradiction that f has a maxterm of the form

$$x_1 \bigvee x_2 \bigvee x_3 \bigvee \cdots \bigvee x_p \bigvee \neg x_{p+1} \bigvee \cdots \bigvee \neg x_q.$$

(For simplicity we assume $p \geq 3$. Other cases where one or more of the variables $x_1, \ldots, x_3$ are negated can be handled similarly.) Consider the function

$$f_1(x_1 x_2 x_3) \stackrel{\text{def}}{=} \exists x_{q+1}, \ldots, x_k \text{ s.t. } f(x_1, x_2, x_3, 0^{p-3}, 1^{q-p}, x_{q+1}, \ldots, x_k).$$

Then since $x_1 \bigvee x_2 \bigvee x_3 \ldots$ is a maxterm of f, we have that $f_1(000) = 0$ and $f_1(100) = f_1(010) = f_1(001) = 1$. We claim that f_1 cannot be a 2CNF function. If not, then

to make $f_1(000) = 0$, at least one of the clauses x_1, x_2, x_3, $x_1 \bigvee x_2$, $x_2 \bigvee x_3$, or $x_3 \bigvee x_1$ should be a clause of f_1 in any 2CNF representation. However, all these clauses are left unsatisfied by at least one of the assignments 100, 010, or 001. This validates our claim that f_1 is not a 2CNF constraint. However, f_1 was obtained from f by setting some variables to a constant and existentially quantifying over others, and by Lemma 4.19 f_1 must also be a 2CNF function. This yields the desired contradiction. □

LEMMA 4.22. *An affine function f is a width-2 affine function if and only if all its minimally dependent sets are of cardinality at most* 2.

Proof. We use the fact that $\mathcal{F}_{2A} \subseteq \mathcal{F}_{2CNF} \cap \mathcal{F}_A$. Suppose $f \in \mathcal{F}_{2A}$ has a minimally dependent set of size $p \geq 3$ and, say, the set is $x_1, \ldots, x_p$. Then by existential quantification over the variables $x_{p+1}, \ldots, x_k$ and by setting the variables $x_4, \ldots, x_p$ to 0, we obtain the function $f_1(x_1, x_2, x_3)$ which is an affine function (by Lemma 4.17) with x_1, x_2, x_3 as a minimally dependent set. Thus this function is either XOR_3 or XNOR_3. However, now notice that neither of these functions is a 2CNF function. However, since f is a 2CNF function Lemma 4.19 implies that f_1 must also be a 2CNF function. This yields the required contradiction. □

5. Classification of MAX CSP. The main results of this section are in sections 5.1 and 5.2. These results were originally obtained by Creignou [11]. Her focus, however, is on the the complexity of finding optimal solutions to the optimization problems. The proofs for hardness of approximation are left to the reader to verify. We give full proofs using the notions of implementations. Our proof is also stronger since it does not assume replication of variables as a basic primitive. This allows us to talk about problems such as MAX EkSAT. In section 5.3 we extend Schaefer's results to establish the hardness of satisfiable MAX CSP problems. Similar results, again with replication of variables being allowed, were first shown by Hunt, Marathe, and Stearns [26].

5.1. Containment results for MAX CSP. We start with the polynomial time solvable cases.

PROPOSITION 5.1. WEIGHTED MAX CSP($\mathcal{F}$) *(*WEIGHTED MIN CSP($\mathcal{F}$)*) is in* PO *if $\mathcal{F}$ is* 0*-valid (*1*-valid).*

Proof. Set each variable to zero (resp., one); this satisfies all of the constraints. □

Before proving the containment in PO of MAX CSP($\mathcal{F}$) for 2-monotone function families, we show that the corresponding WEIGHTED MIN CSP($\mathcal{F}$) is in PO. The containment for WEIGHTED MAX CSP($\mathcal{F}$) will follow easily.

LEMMA 5.2. WEIGHTED MIN CSP($\mathcal{F}$) *is in* PO *if $\mathcal{F}$ is* 2*-monotone.*

Proof. This problem reduces to the problem of finding s-t min-cut in directed weighted graphs. 2-monotone constraints have the following possible forms:

(a) $\mathrm{AND}_p(x_{i_1}, \ldots, x_{i_p})$,
(b) $\mathrm{NOR}_q(x_{j_1}, \ldots, x_{j_q})$, and
(c) $\mathrm{AND}_p(x_{i_1}, \ldots, x_{i_p}) \bigvee \mathrm{NOR}_q(x_{j_1}, \ldots, x_{j_q})$.

Construct a directed graph G with two special nodes, F and T, and a vertex v_i corresponding to each variable x_i in the input instance. Let ∞ denote an integer larger than the total weight of all constraints.

Now we proceed as follows for each of the above classes of constraints:

- For a constraint C of weight w of the form (a), create a new node e_C and add an edge from each v_{i_l}, $l \in [p]$, to e_C of capacity ∞ and an edge from e_C to T of capacity w.

- For a constraint C of weight w of the form (b), create a new node $\overline{e_C}$ and add an edge from $\overline{e_C}$ to each v_{j_l}, $l \in [q]$, of capacity ∞ and an edge from F to $\overline{e_C}$ of capacity w.
- Finally, for a constraint C of weight w of the form (c), we create two nodes e_C and $\overline{e_C}$. For every $l \in [p]$, we add an edge from v_{i_l} to e_C of capacity ∞, and for every $l \in [q]$, we add an edge from $\overline{e_C}$ to v_{j_l} of capacity ∞ and finally an edge from e_C to $\overline{e_C}$ of capacity w. (Note in this case there are no edges connecting F or T to any of the vertices.)

Notice that each vertex of type e_C or $\overline{e_C}$ can be associated with a term: e_C with a term on positive literals and $\overline{e_c}$ with a term on negated literals. We use this association to show that the value of the min F-T cut in this directed graph equals the weight of the minimum number of unsatisfied constraints in the given Weighted Min CSP($\mathcal{F}$) instance.

Given an assignment which fails to satisfy constraints of weight W, we associate a cut as follows: Vertex v_i is placed on the F side of the cut if and only if it is set to 0. A vertex e_C is placed on the T side if and only if the term associated with it is satisfied. A vertex $\overline{e_C}$ is placed on the F side if and only if the term associated with it is satisfied. It can be verified that such an assignment has no directed edges of capacity ∞ going from the F side of the cut to the T side of the cut. Furthermore, for every constraint C of weight w, the associated edge of capacity w crosses the cut if and only if the constraint is not satisfied. Thus the capacity of this cut is exactly W and thus we find that the min F-T cut value is at most W.

In the other direction, we show that given a F-T cut in this graph of cut capacity $W < \infty$, there exists an assignment which fails to satisfy constraints of weight at most W. Such an assignment is simply to assign $x_i = 0$ if and only if v_i is on the F side of the cut. Note that for any constraint C, the associated vertices e_C and $\overline{e_c}$ (whichever exist) may be placed on the T and F sides of the cut (respectively) only if the associated term is satisfied (otherwise there will be an edge of capacity ∞ crossing the cut). Thus, if a constraint C of capacity w is not satisfied by this assignment, then the edge of capacity w corresponding to C must cross the cut. Summing up we find that the assignment fails to satisfy constraints of total weight at most W.

Putting both directions together, we find that the min F-T cut in this graph has capacity exactly equal to the optimum of the Weighted Min CSP{XOR} instance, and thus the latter problem can be solved exactly in polynomial time. □

For the sake of completeness we also prove the converse direction to the above lemma. We show that the s-t min-cut problem can be phrased as a Min CSP($\mathcal{F}$) problem for a 2-monotone family $\mathcal{F}$.

Lemma 5.3. *The s-t min-cut problem is in* Weighted Min CSP($\{\mathrm{OR}_{2,1}, T, F\}$).

Proof. Given an instance $G = (V, E)$ of the s-t min-cut problem, we construct an instance of Weighted Min CSP($\mathcal{F}$) on variables $x_1, x_2, \ldots, x_n$, where x_i corresponds to the vertex $i \in V - \{s, t\}$:

- For each edge $e = (s, i)$ with weight w_e, we create the constraint $F(x_i)$ with weight w_e.
- For each edge $e = (i, t)$ with weight w_e, we create the constraint $T(x_i)$ with weight w_e.
- For each edge $e = (i, j)$ with weight w_e and such that $i, j \notin \{s, t\}$, we create the constraint $\mathrm{OR}_{2,1}(x_j, x_i)$ with weight w_e.

Given a solution to this instance of Weighted Min CSP($\mathcal{F}$), we construct an s-t cut by placing the vertices corresponding to the false variables on the s-side of

the cut and the remaining on the t-side of the cut. It is easy to verify that an edge e contributes to the cut if and only if its corresponding constraint is unsatisfied. Hence the optimal MIN CSP($\mathcal{F}$) solution and the optimal s-t min-cut solution coincide. □

Going back to our main objective, we obtain as a simple corollary to Lemma 5.2 the following corollary.

COROLLARY 5.4. *For every* $\mathcal{F} \subseteq \mathcal{F}_{2M}$, WEIGHTED MAX CSP($\mathcal{F}$)$\in$ PO.

Proof. The proof follows from the fact that given an instance $\mathcal{I}$ of WEIGHTED MAX CSP($\mathcal{F}$), the optimum solution to $\mathcal{I}$ viewed as an instance of WEIGHTED MIN CSP($\mathcal{F}$) is also an optimum solution to the WEIGHTED MAX CSP($\mathcal{F}$) version. □

Finally we prove a simple containment result for all of MAX CSP($\mathcal{F}$) which follows as an easy consequence of Proposition 3.7.

PROPOSITION 5.5. *For every* $\mathcal{F}$, WEIGHTED MAX CSP($\mathcal{F}$) *is in* APX.

Proof. The proof follows from Proposition 3.7 and the fact that the total weight of all constraints is an upper bound on the optimal solution. □

5.2. Negative results for MAX CSP. In this section we prove that if $\mathcal{F} \not\subseteq \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2M}$, then MAX CSP($\mathcal{F}$) is APX-hard. We start with a simple observation which establishes MAX CSP(XOR) as our starting point.

LEMMA 5.6. MAX CSP(XOR) *is* APX*-hard.*

Proof. We observe that MAX CSP(XOR) captures the MAX CUT problem shown to be APX-hard by [39, 3]. Given a graph $G = (V, E)$ with n vertices and m edges, create an instance $\mathcal{I}_G$ of MAX CSP($\{$XOR$\}$) with one variable x_u for every vertex $u \in V$ and with constraints XOR(x_u, x_v) corresponding to every edge $\{u, v\} \in E$. It is easily seen there is a one-to-one correspondence between (ordered) cuts in G and the assignments to the variables of $\mathcal{I}_G$ which maintains the values of the objective functions (i.e., the cut value and the number of satisfied constraints). □

We start with the following lemma which shows how to use the functions which are not 0-valid or 1-valid.

LEMMA 5.7. *If* $\mathcal{F} \not\subseteq \mathcal{F}_0, \mathcal{F}_1$, *then* MAX CSP($\mathcal{F} \cup \{T, F\}$) *is AP-reducible to* MAX CSP($\mathcal{F}$) *and* MIN CSP($\mathcal{F} \cup \{T, F\}$) *is A-reducible to* MIN CSP($\mathcal{F}$).

Proof. Let f_0 be the function from $\mathcal{F}$ that is not 0-valid and let f_1 be the function that is not 1-valid. If some function g in $\mathcal{F}$ is not C-closed, then by Lemma 4.6 $\mathcal{F}$ perfectly and strictly implements T and F. Hence, by Lemmas 3.8 and 3.10, MAX CSP($\mathcal{F} \cup \{T, F\}$) is AP-reducible to MAX CSP($\mathcal{F}$) and MIN CSP($\mathcal{F} \cup \{T, F\}$) is A-reducible to MIN CSP($\mathcal{F}$).

Otherwise, every function of $\mathcal{F}$ is C-closed, and hence by Lemma 4.5, $\mathcal{F}$ perfectly and strictly implements the XOR function and hence, by Proposition 3.3, the XNOR function. Thus it suffices to show that MAX CSP($\mathcal{F} \cup \{T, F\}$) is AP-reducible to MAX CSP($\mathcal{F} \cup \{$XOR, XNOR$\}$) (and MIN CSP($\mathcal{F} \cup \{T, F\}$) is A-reducible to MIN CSP($\mathcal{F} \cup \{$XOR, XNOR$\}$)) for C-closed families $\mathcal{F}$. Here we use an idea from [8] described next.

Given an instance $\mathcal{I}$ of MAX CSP($\mathcal{F} \cup \{T, F\}$) on variables $x_1, \ldots, x_n$ and constraints $C_1, \ldots, C_m$, we define an instance $\mathcal{I}'$ of MAX CSP($\mathcal{F} \cup \{$XOR, XNOR$\}$) (MIN CSP($\mathcal{F} \cup \{$XOR, XNOR$\}$)) whose variables are $x_1, \ldots, x_n$ and additionally one new auxiliary variable x_F. Each constraint of the form $F(x_i)$ (resp., $T(x_i)$) in $\mathcal{I}$ is replaced by a constraint XNOR(x_i, x_F) (resp., XOR(x_i, x_F)). All the other constraints are not changed. Thus $\mathcal{I}'$ also has m constraints. Given a solution $a_1, \ldots, a_n, a_F$ for $\mathcal{I}'$ that satisfies m' of these constraints, notice that the assignment $\neg a_1, \ldots, \neg a_n, \neg a_F$ also satisfies the same collection of constraints (since every function in $\mathcal{F}$ is C-closed). In

one of these cases the assignment to x_F is false and then we notice that a constraint of $\mathcal{I}$ is satisfied if and only if the corresponding constraint in $\mathcal{I}'$ is satisfied. Thus every solution to $\mathcal{I}'$ can be mapped to a solution to $\mathcal{I}$ with the same contribution to the objective function. □

The required lemma now follows as a simple combination of Lemmas 4.9 and 5.7.

LEMMA 5.8. *If* $\mathcal{F} \not\subseteq \mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2M}$, *then* MAX CSP($\mathcal{F}$) *is* APX*-hard.*

Proof. By Lemma 4.11 $\mathcal{F} \cup \{T, F\}$ strictly implements the XOR function. Thus MAX CSP(XOR) AP-reduces to MAX CSP($\mathcal{F} \cup \{T, F\}$) which in turn (by Lemma 5.7) AP-reduces to MAX CSP($\mathcal{F}$). Thus MAX CSP($\mathcal{F}$) is APX-hard. □

5.3. Hardness at gap location 1. Schaefer's dichotomy theorem can be extended to show that in the cases where SAT($\mathcal{F}$) is NP-hard to decide, it is actually hard to distinguish satisfiable instances from instances which are not satisfiable in a constant fraction of the constraints. This is termed hardness at gap location 1 by Petrank [40], who highlights the utility of such hardness results in other reductions. The essential observation needed is that perfect implementations preserve hardness gaps located at 1 and that Schaefer's proof is based on perfect implementations.

However, Schaefer's proof of NP-hardness in his dichotomy theorem relies on the ability to replicate variables within a constraint application. Specifically, the following lemma can be abstracted from his paper.

LEMMA 5.9 (see [42]). *If* $\mathcal{F}$ *is not* 0*-valid or* 1*-valid or affine or bijunctive or weakly positive or weakly negative, then* $\mathcal{F} \cup \{\text{XNOR}\} \stackrel{p}{\Longrightarrow} \mathcal{F}_{3\text{SAT}}$.

In this section, we show that a family $\mathcal{F}$ that is not decidable also perfectly implements the XNOR constraint and thus the lemma above can be strengthened. We start with the following lemma that shows how to use functions that are not weakly negative.

LEMMA 5.10. *If* f *is not weakly negative, then* $\{f, T, F\} \stackrel{p}{\Longrightarrow}$ XOR *or* $\{f, T, F\} \stackrel{p}{\Longrightarrow}$ OR. *Similarly, if* f *is not weakly positive, then* $\{f, T, F\} \stackrel{p}{\Longrightarrow}$ XOR *or* $\{f, T, F\} \stackrel{p}{\Longrightarrow}$ NAND.

Proof. We prove only the first part—the second part follows by symmetry. By Lemma 4.20 we find that f has a maxterm with at least two positive literals. W.l.o.g. the maxterm is of the form $x_1 \bigvee x_2 \bigvee \cdots x_p \bigvee \neg x_{p+1} \bigvee \cdots \bigvee \neg x_q$ with $p \geq 2$. We consider the function f' which is f existentially quantified over all variables but $x_1, \ldots, x_q$. Furthermore, we set $x_3, \ldots, x_p$ to 0 and $x_{p+1}, \ldots, x_q$ to 1. Then the assignment $x_1 = x_2 = 0$ is a nonsatisfying assignment. The assignments $x_1 = 0 \neq x_2$ and $x_1 \neq 0 = x_2$ must be satisfying assignments by the definition of maxterm (and in particular by the minimality of the clause). The assignment $x_1 = x_2 = 1$ may go either way. Depending on this we get either the function XOR or OR. □

COROLLARY 5.11. *If* f_2 *is not weakly positive and* f_3 *is not weakly negative, then* $\{f_2, f_3, T, F\} \stackrel{p}{\Longrightarrow}$ XOR.

LEMMA 5.12. *If* $\mathcal{F}$ *is not* 0*-valid or* 1*-valid or weakly positive or weakly negative, then* $\mathcal{F} \stackrel{s/p}{\Longrightarrow} \{\text{XOR}, \text{XNOR}\}$.

Proof. If $\mathcal{F}$ is C-closed, then, by Lemma 4.5, we immediately get a strict and perfect implementation of XOR. If it is not C-closed, then, by Lemma 4.6, we get perfect and strict implementations of the constraints T and F. Applying Corollary 5.11 now, we get a perfect and strict implementation of XOR in this case also. Finally we use Proposition 3.3 to get a perfect and strict implementation of XNOR from the constraint XOR. □

Combining Lemma 5.9 and the above, we get the following corollary.

COROLLARY 5.13. *If $\mathcal{F}$ is not 0-valid or 1-valid or affine or bijunctive or weakly positive or weakly negative, then $\mathcal{F} \stackrel{p}{\Longrightarrow} \mathcal{F}_{3\mathrm{SAT}}$.*

Thus we get the following theorem.

THEOREM 5.14. *For every constraint set $\mathcal{F}$, either SAT($\mathcal{F}$) is easy to decide or there exists $\epsilon = \epsilon_{\mathcal{F}} > 0$ such that it is NP-hard to distinguish satisfiable instances of SAT($\mathcal{F}$) from instances where $1 - \epsilon$ fraction of the constraints are not satisfiable.*

6. Classification of MAX ONES. Again we will first prove the positive results and then show the negative results. However, before we do either, we will show a useful reduction between unweighted and weighted MAX ONES($\mathcal{F}$) problems which holds for most interesting function families $\mathcal{F}$.

6.1. Preliminaries. We begin with a slightly stronger notion of the definition of polynomial time solvability of SAT($\mathcal{F}$) (than that of [42]). We then show that given this stronger form of polynomial time decidability the weighted and unweighted cases of MAX ONES($\mathcal{F}$) are equivalent by showing that this stronger form of polynomial time decidability leads to a polynomial approximation algorithm. We conclude by showing that for the MAX ONES problems, which we hope to show to be APX-complete or poly-APX-complete, the strong form of decidability does hold.

DEFINITION 6.1. *We say that a constraint family $\mathcal{F}$ is* strongly decidable *if, given m constraints from $\mathcal{F}$ on n variables $x_1, \ldots, x_n$* and an index $i \in \{1, \ldots, n\}$, *there exists a polynomial time algorithm to find an assignment to $x_1, \ldots, x_n$ satisfying all m constraints and additionally satisfying the property $x_i = 1$ if one such exists.*

LEMMA 6.2. *For every strongly decidable constraint family $\mathcal{F}$,* WEIGHTED MAX ONES($\mathcal{F}$) *is in* poly-APX.

Proof. Consider an instance of WEIGHTED MAX ONES($\mathcal{F}$) with variables $x_1, \ldots, x_n$, constraint applications $C_1, \ldots, C_m$, and weights $w_1, \ldots, w_n$. Assume $w_1 \leq w_2 \leq \cdots \leq w_n$. Let i be the largest index such that there exists a feasible solution with $x_i = 1$. Notice that i can be determined in polynomial time due to the strong decidability of $\mathcal{F}$. We also use the strong decidability to find an assignment with $x_i = 1$. It is easily verified that this yields an n-approximate solution. (Weight of this solution is at least w_i, while weight of optimal is at most $\sum_{j=1}^{i} w_j \leq i w_i \leq n w_i$.) ∎

Before concluding we show that most problems of interest to us will be able to use the equivalence established above between weighted and unweighted problems.

LEMMA 6.3. *If $\mathcal{F} \subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{S0}}, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then $\mathcal{F}$ is strongly decidable.*

Proof. Recall that for $i \in [k]$, $f|_{(\{i\},1)}$ is the constraint obtained from f by restricting the ith input to 1. Define $\mathcal{F}^*$ to be the constraint set

$$\mathcal{F}^* \stackrel{\mathrm{def}}{=} \mathcal{F} \cup \{f|_{i,1} | f \in \mathcal{F}, i \in [k]\}.$$

First, observe that the problem of strong decidability of $\mathcal{F}$ reduces to the decision problem SAT($\mathcal{F}^*$). Furthermore, observe that if $\mathcal{F} \subseteq \mathcal{F}'$ for $\mathcal{F}' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then $\mathcal{F}^* \subseteq \mathcal{F}'$ as well. Finally, if $\mathcal{F}^* \subseteq \mathcal{F}_{\mathrm{S0}}$, then $\mathcal{F}^* \subseteq \mathcal{F}_0$. Thus in each case we end up with a problem from SAT($\mathcal{F}$) for a family $\mathcal{F}$ which is polynomial time decidable in Schaefer's dichotomy. ∎

LEMMA 6.4. *If $\mathcal{F} \stackrel{p}{\Longrightarrow} f_0$ for some existential zero constraint f_0, then $\mathcal{F} \stackrel{p}{\Longrightarrow} \mathcal{F}|_0$. Similarly, if $\mathcal{F} \stackrel{p}{\Longrightarrow} f_1$ for some existential one constraint f_1, then $\mathcal{F} \stackrel{p}{\Longrightarrow} \mathcal{F}|_1$.*

Proof. Let $f \in \mathcal{F}$. We show how to implement the constraint $f(0, x_1, \ldots, x_{k-1})$. The proof can be extended to other constraints in $\mathcal{F}|_0$ by induction. Let f_0 be an

existential zero constraint implementable by $\mathcal{F}$ and let K be the arity of f_0. Then the constraints $f(y_i, x_1, \ldots, x_{k-1})$, for $i \in [K]$, along with the constraint $f_0(y_1, \ldots, y_K)$ perfectly implement the constraint $f(0, x_1, \ldots, x_{k-1})$. (Observe that since at least one of the y_i's in the set $y_1, \ldots, y_K$ is zero, the constraint $f(0, x_1, \ldots, x_{k-1})$ is being enforced. Furthermore, we can always set all of $y_1, \ldots, y_K$ to zero, ensuring that any assignment to $x_1, \ldots, x_{k-1}$ satisfying $f(0, x_1, \ldots, x_{k-1})$ does satisfy all the constraints listed above.) □

6.2. Containment results.

LEMMA 6.5. *If $\mathcal{F}$ is* 1*-valid or weakly positive or width-*2 *affine, then* WEIGHTED MAX ONES($\mathcal{F}$) *is in* PO.

Proof. If $\mathcal{F}$ is 1-valid, then setting each variable to 1 satisfies all constraint applications with the maximum possible variable weight.

If $\mathcal{F}$ is weakly positive, consider the CNF formulae for the $f_i \in \mathcal{F}$ such that each clause has at most one negated variable. Clearly, clauses consisting of a single literal force the assignment of these variables. Setting these variables may create new clauses of a single literal; set these variables and continue the process until all clauses have at least two literals or until a contradiction is reached. In the latter case, no feasible assignment is possible. In the former case, setting the remaining variables to 1 satisfies all constraints, and there exists no feasible assignment with a greater weight of ones.

In the case that $\mathcal{F}$ is affine with width-2, we reduce the problem of finding a feasible solution to that of checking whether a graph is bipartite and then use the bipartition to find the optimal solution. Notice that each constraint corresponds to a conjunction of constraints of the form $X_i = X_j$ or $X_i \neq X_j$. Create a vertex X_j for each variable X_j and for each constraint $X_i \neq X_j$, add an edge (X_i, X_j). For each constraint $X_i = X_j$, identify the vertices X_i and X_j and associate the sum of their weights to the identified vertex; if this creates a self-loop, then clearly no feasible assignment is possible. Check whether the graph is bipartite; if not, then there is no feasible assignment. If it is bipartite, then for each connected component of the graph choose the larger weight side of the bipartition and set the corresponding variables to 1. □

LEMMA 6.6. *If $\mathcal{F}$ is affine, then* WEIGHTED MAX ONES($\mathcal{F}$) *is in* APX.

Remark. Our proof actually shows that MAX ONES($\mathcal{F}$) has a 2-approximation algorithm. Combined with the fact that the AP-reduction of Lemma 3.11 does not lose much in the approximation factor we essentially get the same factor for WEIGHTED MAX ONES($\mathcal{F}$) as well.

Proof. By Lemmas 3.11, 6.2, and 6.3 it suffices to consider the unweighted case. (Lemma 6.3 shows that $\mathcal{F}$ is strongly decidable; Lemma 6.2 uses this to show that WEIGHTED MAX ONES($\mathcal{F}$) is in poly-APX; and Lemma 3.11 uses this to provide an AP-reduction from WEIGHTED MAX ONES($\mathcal{F}$) to MAX ONES($\mathcal{F}$).)

Given an instance $\mathcal{I}$ of MAX ONES($\mathcal{F}$), notice that finding a solution which satisfies all constraints is the problem of solving a linear system of equations over GF [2]. Say the linear system is given by $Ax = b$, where A is an $m \times n$ matrix, and b is a $m \times 1$ column vector, and the x is an $n \times 1$ vector. Assume w.l.o.g. that the rows of A are independent. By simple row operations and reordering of the variables, we can set up the linear system as $[I|A']x = b'$. Thus if x' represents the vector $\langle x_1, \ldots, x_m \rangle$ and x'' represents the vector $\langle x_{m+1}, \ldots, x_n \rangle$, then the set of feasible solutions to the given linear system are given by

$$\{\langle x', x'' \rangle | x'' \in \{0,1\}^{n-m}, x' = -A'x'' + b'\}.$$

Pick a random element of this set by picking x'' at random and setting x' accordingly. Notice that for any $i \in \{m+1, \ldots, n\}$ $x_i = 1$ with probability (w.p.) $\frac{1}{2}$. Furthermore, for any $i \in [m]$, x_i is either forced to 0 in all feasible solutions, or x_i is forced to 1 in all feasible solutions, or $x_i = 1$ w.p. 1/2. Thus, if $S \subseteq [n]$ is the set of variables which are ever set to 1 in a feasible solution, then the expected number of 1's in a random solution is at least $|S|/2$. However, S is an upper bound on OPT. Thus the expected value of the solution is at least OPT/2, and hence the solution obtained is a 2-approximate solution. □

PROPOSITION 6.7. *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{S0}}, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then* WEIGHTED MAX ONES$(\mathcal{F}) \in$ poly-APX.

Proof. The proof follows immediately from Lemmas 6.2 and 6.3. □

PROPOSITION 6.8 (see [42]). *If $\mathcal{F} \subseteq \mathcal{F}_0$, then* SAT$(\mathcal{F})$ *is in* P.

6.3. Hardness results.

6.3.1. APX-hard case. We wish to show in this section that if $\mathcal{F}$ is an affine family but not width-2 affine, then MAX ONES$(\mathcal{F})$ is APX-hard. By Lemmas 6.2 and 3.11 it suffices to show this for WEIGHTED MAX ONES$(\mathcal{F})$. The basic APX-hard problems we work with in this section are described in the following lemma.

LEMMA 6.9. WEIGHTED MAX ONES(XNOR_3) *and* WEIGHTED MAX ONES $(\{\mathrm{XOR}, \mathrm{XNOR}_4\})$ *are* APX*-hard.*

Proof. We reduce the MAX CUT problem to the WEIGHTED MAX ONES(XNOR_3) problem as follows. Given a graph $G = (V, E)$ we create a variable x_v for every vertex $v \in V$ and a variable y_e for every edge $e \in E$. The weight w_v associated with the vertex variable x_v is 0. The weight w_e of an edge variable y_e is 1. For every edge e between u and v we create the constraint $y_e \oplus x_u \oplus x_v = 0$. It is clear that any 0/1 assignment to the x_v's define a cut and for an edge $e = \{u, v\}$, y_e is 1 if and only if u and v are on opposite sides of the cut. Thus solutions to the WEIGHTED MAX ONES problem correspond to cuts in G with the objective function being the number of edges crossing the cut. This shows the APX-hardness of WEIGHTED MAX ONES(XNOR_3).

The reduction for WEIGHTED MAX ONES$(\{\mathrm{XOR}, \mathrm{XNOR}_4\})$ is similar. Given a graph $G = (V, E)$, we create the variables x_v for every $v \in V$, y_e for every $e \in E$, and one global variable z (which is supposed to be zero) and $m \stackrel{\mathrm{def}}{=} |E|$ auxiliary variables y'_e for every $e \in E$. For every edge $e = \{u, v\}$ in G we impose the constraints $y_e \oplus x_u \oplus x_v \oplus z = 0$. In addition we throw in the constraints $z \oplus y'_e = 1$ for every $i \in \{1, \ldots, m\}$. Finally we make the weight of the vertex variables and z 0, and the weight of the edge variables y_e and the auxiliary variables y'_e is made 1. The optimum to this WEIGHTED MAX ONES problem is MAX CUT$(G) + m$. Given an r-approximate solution for the WEIGHTED MAX ONES$(\{\mathrm{XOR}_4, \mathrm{XOR}\})$ instance created above, we consider the two possible solutions (as usual): (1) the solution induced by the assignment with zero vertices on one side and one vertex on the other and (2) a cut with m/K edges crossing the cut (notice such a cut can be found based on Proposition 3.7). The better of these solutions has $\max\{(\frac{1}{r})(m + \text{MAX CUT}(G)) - m, \frac{m}{K}\} \geq \frac{1}{r(K(1-1/r)+1)}\text{MAX CUT}(G) \geq \frac{1}{1+K(r-1)}\text{MAX CUT}(G)$ edges crossing the cut. Thus an r-approximate solution to WEIGHTED MAX ONES$(\{\mathrm{XOR}, \mathrm{XNOR}_4\})$ yields a $(1+K(r-1))$-approximate solution to MAX CUT(G). Thus MAX CUT(G) AP-reduces to WEIGHTED MAX ONES$(\{\mathrm{XOR}, \mathrm{XNOR}_4\})$, and hence the latter is APX-hard. □

LEMMA 6.10. *If $\mathcal{F}$ is affine but neither width-2 affine nor 1-valid, then $\mathcal{F} \xrightarrow{p} \mathrm{XNOR}_3$ or $\mathcal{F} \xrightarrow{p} \{\mathrm{XOR}, \mathrm{XNOR}_4\}$.*

Proof. Since $\mathcal{F}$ is affine but not of width-2, it can perfectly (and strictly) implement the function XOR_p or XNOR_p for some $p \geq 3$ (Lemma 4.18). Let $f \in \mathcal{F}$ be an affine constraint that is not 1-valid. We consider two possible cases depending on whether $\mathcal{F}$ is C-closed or not. If $g \in \mathcal{F}$ is not C-closed, then we find (by Lemma 4.7) that $\{f, g\}$ (and hence $\mathcal{F}$) perfectly implements some existential zero constraint. This case is covered in Claim 6.11 and we show that in this case $\mathcal{F}$ perfectly implements XNOR_3. In the other case, $\mathcal{F}$ is C-closed, and hence (by Lemma 4.5) $\mathcal{F}$ perfectly implements the constraint XOR. This case is covered in Claim 6.12 and we show that in this case $\mathcal{F}$ perfectly implements either XNOR_3 or XNOR_4. This concludes the proof of Lemma 6.10 (modulo Claims 6.11 and 6.12). □

CLAIM 6.11. *If $\{f\}$ is an existential zero constraint and h is either the constraint XOR_p or XNOR_p for some $p \geq 3$, then $\{f, h\} \xrightarrow{p} \mathrm{XNOR}_3$.*

Proof. Since f is an existential zero constraint, the family $\{f, h\}$ can perfectly implement $\{f, h\}|_0$ (using Lemma 6.4). In particular, $\{f, h\}$ can implement the constraints $x_1 \oplus x_2 = b$ and $x_1 \oplus x_2 \oplus x_3 = b$ for some $b \in \{0, 1\}$. Notice finally that the constraints $x_1 \oplus x_2 \oplus y = b$ and $y \oplus x_3 = b$ form a perfect implementation of the constraint $x_1 \oplus x_2 \oplus x_3 = 0$. Thus $\{f, h\}$ perfectly implements the constraint XNOR_3. □

CLAIM 6.12. *If $f \in \{\mathrm{XOR}_p, \mathrm{XNOR}_p \mid p \geq 3\}$, then $\{f, \mathrm{XOR}\} \xrightarrow{p} \mathrm{XNOR}_3$ or $\{f, \mathrm{XOR}\} \xrightarrow{p} \mathrm{XNOR}_4$.*

Proof. Since XOR perfectly implements XNOR it suffices to prove this using the constraints $\{f, \mathrm{XOR}, \mathrm{XNOR}\}$.

W.l.o.g assume that f is the constraint XNOR, since otherwise $\mathrm{XOR}_p(x_1, \ldots, x_{p-1}, y)$ and $\mathrm{XOR}(y, x_p)$ perfectly implement the constraint $\mathrm{XNOR}_p(x_1, \ldots, x_p)$.

Now if p is odd, then the constraints $\mathrm{XNOR}_p(x_1, \ldots, x_p)$ and $\mathrm{XNOR}(x_4, x_5)$, $\mathrm{XNOR}(x_6, x_7)$, and so on up to $\mathrm{XNOR}(x_{p-1}, x_p)$ perfectly implement the constraint $\mathrm{XNOR}_3(x_1, x_2, x_3)$.

Now if p is even, then the constraints $\mathrm{XNOR}_p(x_1, \ldots, x_p)$ and $\mathrm{XNOR}(x_5, x_6)$, $\mathrm{XNOR}(x_7, x_8)$, and so on up to $\mathrm{XNOR}(x_{p-1}, x_p)$ perfectly implement the constraint $\mathrm{XNOR}_4(x_1, x_2, x_3, x_4)$. □

LEMMA 6.13. *If $\mathcal{F}$ is affine but neither width-2 affine nor 1-valid, then* MAX ONES($\mathcal{F}$) *is* APX*-hard.*

Proof. By Lemma 6.6 we have WEIGHTED MAX ONES($\mathcal{F}$) is in APX and thus (by Lemma 3.11) it suffices to show APX-hardness of WEIGHTED MAX ONES($\mathcal{F}$). This now follows from Lemmas 3.9, 6.9, and 6.10. □

6.3.2. The poly-APX-hard case. This part turns out to be long and the bulk of the work will be done in Lemmas 6.16–6.21. We first describe the proof of the hardness result modulo the above lemmas. (Hopefully, the proof will also provide some motivation for the rest of the lemmas.)

LEMMA 6.14. *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{2\mathrm{CNF}}, \mathcal{F}_{\mathrm{WN}}\}$ but $\mathcal{F} \not\subseteq \mathcal{F}''$ for any $\mathcal{F}'' \in \{\mathcal{F}_1, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}\}$, then* MAX ONES($\mathcal{F}$) *is* poly-APX*-hard.*

Proof. As usual, by Lemmas 6.2 and 3.11, it suffices to show hardness of the weighted version. First we show in Lemma 6.15 that MAX ONES($\{\mathrm{NAND}_k\}$) is poly-APX-hard for every $k \geq 2$. Thus our goal is to establish that any non-1-valid, nonaffine, and nonweakly positive constraint family can implement some NAND_k constraint. We do so in three phases.

The main complication here is that we don't immediately have a non-0-valid constraint to work with and thus we can't immediately reduce MAX ONES($\mathcal{F} \cup \{T, F\}$) to MAX ONES($\mathcal{F}$). Therefore we go after something weaker and try to show that $\mathcal{F}$ can perfectly implement $\mathcal{F}|_{0,1}$. In Phase 3 (Lemmas 6.20 and 6.21) we show that this suffices. Lemma 6.20 uses the fact that $\mathcal{F}|_{0,1}$ is not weakly positive to implement either NAND_2 or XOR. In the former case we are done and in the latter case, Lemma 6.21 uses the fact that $\mathcal{F}|_{0,1}$ is not affine to implement NAND.

Thus our task reduces to that of showing that $\mathcal{F}$ can implement $\mathcal{F}|_{0,1}$. Part of this is easy. In Phase 1, we show that $\mathcal{F}$ implements every constraint in $\mathcal{F}|_0$. This is shown via Lemma 6.16 which shows that any family which is either 0-valid or 2CNF or weakly negative but not 1-valid or affine or weakly positive must have a non-C-closed constraint. This along with the non-1-valid constraint allows it to implement every constraint in $\mathcal{F}|_0$ (by Lemmas 4.7 and 6.4). The remaining task for Phase 2 is to show that $\mathcal{F}|_0$ can implement $\mathcal{F}|_1$. If $\mathcal{F}$ also has a non-0-valid constraint, then we are done since now we can implement all of $\mathcal{F}|_{0,1}$ (another application of Lemmas 4.7 and 6.4). Thus all lemmas in Phase 2 focus on $\mathcal{F}|_0$ for 0-valid constraint families $\mathcal{F}$. If $\mathcal{F}|_0$ is all 0-valid, then all we can show is that $\mathcal{F}|_0$ either implements NAND_k for some k or $\text{OR}_{2,1}$ (Lemmas 6.17 and 6.18). The former is good, but the latter seems insufficient. In fact we are unable to implement $\mathcal{F}|_{0,1}$ in this case. We salvage the situation by reverting back to reductions. We AP-reduce the problem WEIGHTED MAX ONES($\mathcal{F}|_0 \cup \{\text{OR}_{2,1}\}$) to WEIGHTED MAX ONES($\mathcal{F}|_{0,1}$) (Lemma 6.19). This suffices to establish the poly-APX-hardness of WEIGHTED MAX ONES($\mathcal{F}$) since

$$\begin{aligned}\text{WEIGHTED MAX ONES}(\mathcal{F}|_{0,1}) &\leq_{\text{AP}} \text{WEIGHTED MAX ONES}(\mathcal{F}|_0 \cup \{\text{OR}_{2,1}\}) \\ &\leq_{\text{AP}} \text{WEIGHTED MAX ONES}(\mathcal{F})\end{aligned}$$

and the problem WEIGHTED MAX ONES($\mathcal{F}|_{0,1}$) is poly-APX-hard. □

LEMMA 6.15. MAX ONES($\{\text{NAND}_k\}$) *is* poly-APX-*hard for every* $k \geq 2$.

Proof. We reduce from MAX CLIQUE, which is known to be poly-APX-hard. Given a graph G, construct a MAX ONES($\{f\}$) instance consisting of a variable for every vertex in G and the constraint f is applied to every subset of k vertices in G which does not induce a clique. It may be verified that the optimum number of ones in any satisfying assignment to the instance created in this manner is $\max\{k-1, \omega(G)\}$, where $\omega(G)$ is the size of the largest clique in G. Given a solution to the MAX ONES($\{f\}$) instance with $l \geq k$ ones, the set of vertices corresponding to the variables set to 1 form a clique of size l. If $l < k$, output any singleton vertex. Thus in all cases we obtain a clique of size at least $l/(k-1)$ vertices. Thus given an r-approximate solution to the MAX ONES($\{\text{NAND}_k\}$) problem, we can find a $(k-1)r$-approximate solution to MAX CLIQUE. Thus MAX CLIQUE is A-reducible to MAX ONES($\{\text{NAND}_k\}$). □

Phase 1. $\mathcal{F}$ implements $\mathcal{F}|_0$.

LEMMA 6.16. *If* $\mathcal{F} \subseteq \mathcal{F}'$ *for some* $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{2\text{CNF}}, \mathcal{F}_{\text{WN}}\}$ *but* $\mathcal{F} \not\subseteq \{\mathcal{F}_1, \mathcal{F}_{2\text{A}}, \mathcal{F}_{\text{WP}}\}$, *then there exists a constraint in* $\mathcal{F}$ *that is not C-closed constraint.*

Proof. Notice that a C-closed 0-valid constraint is also 1-valid. Thus if $\mathcal{F}$ is 0-valid, then the non-1-valid constraint is not C-closed.

Next we claim that a C-closed weakly positive constraint f is also weakly negative. To do so, consider the constraint $\bar{f}$ given by $\bar{f}(x) = f(\bar{x})$. Notice that for a C-closed constraint $f = \bar{f}$. Suppose $f(x) = \bigwedge_j C_j(x)$, where the C_j's are weakly positive clauses. Then $\bar{f}(x)$ can be described as $\bigwedge_j \bar{C}_j(x)$ (where $\bar{C}_j(x) = C_j(\bar{x})$). However, in this representation $\bar{f}$ (and thus f) is seen to be a weakly negative constraint,

thereby verifying our claim. Thus if $\mathcal{F}$ is weakly negative but not weakly positive, the nonweakly positive constraint is the non-C-closed constraint.

Finally we consider the case when f is a 2CNF formula. Again define $\bar{f}(x) = f(\bar{x})$ and $f'(x) = f(x)\bar{f}(x)$. Notice that $f' = f$ if f is C-closed. Again consider the CNF representation of $f = \bigwedge_j C_j(x)$, where the $C_j(x)$'s are clauses of f of length 2. Then $f'(x)$ can be expressed as $\bigwedge_j (C_j(x) \bigwedge \bar{C}_j(x))$. However, $C_j \bigwedge \bar{C}_j$ are affine constraints of width-2! Thus f', and hence f, is an affine width-2 constraint. Thus if $\mathcal{F}$ is 2CNF but not width-2 affine, the non-width-2 affine constraint is the non-C-closed constraint. □

Lemma 4.7 along with Lemma 6.4 suffice to prove that $\mathcal{F}$ implements $\mathcal{F}|_0$. We now move on to Phase 2.

Phase 2. From $\mathcal{F}|_0$ to $\mathcal{F}|_{0,1}$.

Recall that if $\mathcal{F}$ has a non-0-valid constraint, then by Lemmas 6.16, 4.7, and 6.4 it implements an existential one constraint and thus $\mathcal{F}|_{0,1}$. Thus all lemmas in this phase assume $\mathcal{F}$ is 0-valid.

LEMMA 6.17. *If f is 0-valid and not weakly positive, then $\{f\}|_0$ either perfectly implements* NAND_k *for some $k \geq 2$ or* $\mathrm{OR}_{2,1}$ *or* XNOR.

Proof. Let $C = \neg x_1 \bigvee \cdots \bigvee \neg x_p \bigvee y_1 \bigvee \cdots \bigvee y_q$ be a maxterm in f with more than one negation, i.e., $p \geq 2$. Since f is not weakly positive, Lemma 4.20 shows that such a maxterm exists. Substituting a 0 in place of variables $y_1, y_2, \ldots, y_q$ and existentially quantifying over all variables not in C, we get a constraint g such that $\neg x_1 \bigvee \neg x_2 \bigvee \cdots \bigvee \neg x_p$ is a maxterm in g. Consider an unsatisfying assignment s for g with the smallest number of 1's and let k denote the number of 1's in s; we know $k > 0$ since the original constraint is 0-valid. W.l.o.g. assume that s assigns value 1 to the variables $x_1, x_2, \ldots, x_k$ and 0 to the remaining variables. It is easy to see that by fixing the variables $x_{k+1}, x_{k+2}, \ldots, x_p$ to 0, we get a constraint $g' = (\neg x_1 \bigvee \neg x_2 \bigvee \cdots \bigvee \neg x_k)$. If $k > 1$, then this perfectly implements the constraint $\mathrm{NAND}_k(x_1, \ldots, x_k)$ and we are done.

Otherwise $k = 1$, i.e., there exists an unsatisfying assignment s which assigns value 1 to exactly one of the x_i's, say, x_1. Now consider a satisfying assignment s' which assigns 1 to x_1 and has a minimum number of 1's among all assignments which assign 1 to x_1. The existence of such an assignment follows from C being a maxterm in g. For instance, the assignment $1^{p-1}0$ is a satisfying assignment which satisfies such a property. W.l.o.g. assume that $s' = 1^i 0^{p-i}$. Thus the constraint g looks as follows:

	x_1	x_2	$x_3 ... x_i$	$x_{i+1} ... x_p$	$g()$
s_1	0	0	00...0	00...0	1
s_2	1	0	00...0	00...0	0
$s' = s_3$	1	1	11...1	00...0	1
s_4	0	1	_..._	00...0	?

Existential quantification over the variables $x_3, x_4, \ldots, x_i$ and fixing the variables x_{i+1} through x_p to 0 yields a constraint g' which is either $\mathrm{OR}_{2,1}(x_2, x_1)$ or $\mathrm{XNOR}(x_1, x_2)$. The lemma follows. □

Now we consider the case where we can implement the function XNOR and show that in this case we can perfectly implement either NAND or $\mathrm{OR}_{2,1}$. In the former case we are done, and for the latter case we show in Lemma 6.19 that WEIGHTED MAX ONES($\mathcal{F}|_1$) is AP-reducible to WEIGHTED MAX ONES($\mathcal{F} \cup \{\mathrm{OR}_{2,1}\}$).

LEMMA 6.18. *If f is 0-valid but not affine, then $\{f\}|_0 \cup \{\mathrm{XNOR}\}$ perfectly implements either* NAND *or the constraint* $\mathrm{OR}_{2,1}$.

Proof. Corollary 4.16 shows that if f is not affine, then there exist two satisfying assignments s_1 and s_2 such that $s_1 \oplus s_2$ is not a satisfying assignment for f. Reorder the variables such that $Z(s_1) \cap Z(s_2) = \{x_1, \ldots, x_p\}$, $Z(s_1) \cap O(s_2) = \{x_{p+1}, \ldots, x_q\}$, $O(s_1) \cap Z(s_2) = \{x_{q+1}, \ldots, x_r\}$, and $O(s_1) \cap O(s_2) = \{x_{r+1}, \ldots, x_k\}$. Using the fact that f is 0-valid, we find that f looks as follows:

	$x_1 \ldots x_p$	$x_{p+1} \ldots x_q$	$x_{q+1} \ldots x_r$	$x_{r+1} \ldots x_k$	$g(\mathbf{x})$
	00...0	00...0	00...0	00...0	1
s_1	00...0	00...0	11...1	11...1	1
s_2	00...0	11...1	00...0	11...1	1
$s_1 \oplus s_2$	00...0	11...1	11...1	00...0	0

Consider the following collection of constraints:

(1) $f(0, \ldots, 0, x_{p+1}, \ldots, x_k)$.
(2) $\mathrm{XNOR}(x, x_i)$ for $i \in Z(s_1) \cap O(s_2)$.
(3) $\mathrm{XNOR}(y, x_i)$ for $i \in O(s_1) \cap Z(s_2)$.
(4) $\mathrm{XNOR}(z, x_i)$ for $i \in O(s_1) \cap O(s_2)$.

Existentially quantifying over the variables $x_{p+1}, \ldots, x_k$ we obtain an implementation of a constraint $h(x, y, z)$ such that $h(000) = h(011) = h(101) = 1$ and $h(110) = 0$. Furthermore, by restricting more of the variables in (1) above to 0, we get a perfect implementation of any constraint in $\{h\}|_0$. Using Claim 6.22 again we get that $\{h\}|_0$ can implement either NAND or $\mathrm{OR}_{2,1}$, and thus we are done. ∎

Finally we show how to use $\mathrm{OR}_{2,1}$ constraints.

LEMMA 6.19. *If $\mathcal{F}$ is 0-valid, then* WEIGHTED MAX ONES$(\mathcal{F}|_1)$ *AP-reduces to* WEIGHTED MAX ONES$(\mathcal{F} \cup \{\mathrm{OR}_{2,1}\})$.

Proof. We show something stronger, namely, WEIGHTED MAX ONES$(\mathcal{F} \cup \{T\})$ AP-reduces to WEIGHTED MAX ONES$(\mathcal{F} \cup \{\mathrm{OR}_{2,1}\})$. This suffices since T is an existential one constraint and thus $\mathcal{F} \cup \{T\}$ can perfectly implement $\mathcal{F}|_1$.

Given an instance $\mathcal{I}$ of WEIGHTED MAX ONES$(\mathcal{F} \cup \{T\})$ construct an instance $\mathcal{I}'$ of WEIGHTED MAX ONES$(\mathcal{F} \cup \{\mathrm{OR}_{2,1}\})$ as follows. The variable set of $\mathcal{I}'$ is the same as that of $\mathcal{I}$. Every constraint from $\mathcal{F}$ in $\mathcal{I}$ is also included in $\mathcal{I}'$. The only remaining constraints are of the form $T(x_i)$ for some variables x_i. We simulate this constraint in $\mathcal{I}'$ with $n-1$ constraints of the form $\mathrm{OR}_{2,1}(x_j, x_i)$ (i.e., $\neg x_j \bigvee x_i$) for every $j \in [n]$, $j \neq i$. Every nonzero solution to the resulting instance $\mathcal{I}'$ is also a solution to $\mathcal{I}$, since the solution must have $x_i = 1$ or else have $x_j = 0$ for every $j \neq i$. Thus the resulting instance of MAX ONES$(\mathcal{F} \cup \{\mathrm{OR}_{2,1}\})$ has the same objective function and the same feasible space and hence is at least as hard as the original problem. ∎

This concludes Phase 2.

Phase 3. $\mathcal{F}|_{0,1}$ implements NAND.

LEMMA 6.20. *If f is not weakly positive, then $\{f\}|_{0,1}$ perfectly implements either* XOR *or* NAND.

Proof. Let $C = (\neg x_1 \bigvee \cdots \bigvee \neg x_p \bigvee y_1 \bigvee \cdots \bigvee y_q)$ be a maxterm in f with more than one negation, i.e., $p \geq 2$. Substituting a 1 for variables $x_3, \ldots, x_p$, a 0 for variables $y_1, \ldots, y_q$, and existentially quantifying over all variables not in C, we get a constraint f' such that $f'(11) = 0$, $f'(01) = f'(10) = 1$. (These three properties follow from the definition of a maxterm.) Depending on whether $f'(00)$ is 0 or 1 we get the function XOR or NAND, respectively. ∎

LEMMA 6.21. *If g is a nonaffine constraint, then* $\{g, \mathrm{XOR}\}|_{0,1} \stackrel{p}{\Longrightarrow}$ NAND.

Proof. Again it suffices to consider $\{g, \mathrm{XOR}, \mathrm{XNOR}\}|_{0,1}$. Let g be of arity k. By Lemma 4.15 we find that there must exist assignments s_1, s_2, and s_3 satisfying g such

	S_{000}	S_{001}	S_{010}	S_{011}	S_{100}	S_{101}	S_{110}	S_{111}	$g(\mathbf{x})$
s_1	0...0	0...0	0...0	0...0	1...1	1...1	1...1	1...1	1
s_2	0...0	0...0	1...1	1...1	0...0	0...0	1...1	1...1	1
s_3	0...0	1...1	0...0	1...1	0...0	1...1	0...0	1...1	1
$s_1 \oplus s_2 \oplus s_3$	0...0	1...1	1...1	0...0	1...1	0...0	0...0	1...1	0

FIG. 1. *Partition of inputs to g.*

that $s_1 \oplus s_2 \oplus s_3$ does not satisfy g. Partition the set $[k]$ into up to eight equivalence classes $S_{b_1b_2b_3}$ for $b_1, b_2, b_3 \in \{0,1\}$ such that for any index $i \in S_{b_1b_2b_3}$, $(s_j)_i = b_j$ for every $j \in \{1,2,3\}$ (refer to Figure 1).

W.l.o.g. assume that $S_{000} = \{1, \ldots, p\}$ and $S_{111} = \{q+1, \ldots, k\}$. Notice that the assignment of a variable in $S_{b_1b_2b_3}$ under assignment $s_1 \oplus s_2 \oplus s_3$ is also fixed (to $b_1 \oplus b_2 \oplus b_3$). Now consider the following collection of constraints:

(1) $g(0, \ldots, 0, x_{p+1} \ldots, x_q, 1, \ldots, 1)$.
(2) $\mathrm{XNOR}(x, x_i)$ for $i \in S_{001}$.
(3) $\mathrm{XNOR}(y, x_i)$ for $i \in S_{010}$.
(4) $\mathrm{XNOR}(z, x_i)$ for $i \in S_{011}$.
(5) $\mathrm{XOR}(z, x_i)$ for $i \in S_{100}$.
(6) $\mathrm{XOR}(y, x_i)$ for $i \in S_{101}$.
(7) $\mathrm{XOR}(x, x_i)$ for $i \in S_{110}$.

By existentially quantifying over the variables $x_{p+1}, \ldots, x_q$ we perfectly implement a constraint $h(x,y,z)$ with the following properties: $h(000) = h(011) = h(101) = 1$ and $h(110) = 0$. Furthermore, by restricting more variables in condition (1) above, we can actually implement any function in the set $\{h\}|_{0,1}$. Claim 6.22 now shows that for any such function h, the set $\{h\}|_0$ perfectly implements either $\mathrm{OR}_{2,1}$ or NAND. In the latter case, we are done. In the former case, notice that the constraints $\mathrm{OR}_{2,1}(x,z)$ and $\mathrm{XOR}(z,y)$ perfectly implement the constraint $\mathrm{NAND}(x,y)$; so in this case too we are done (modulo Claim 6.22). □

CLAIM 6.22. *If h is ternary function such that* $h(000) = h(011) = h(101) = 1$ *and* $h(110) = 0$*, then* $\{h\}|_0 \xRightarrow{p} \mathrm{NAND}$ *or* $\{h\}|_0 \xRightarrow{p} \mathrm{OR}_{2,1}$.

Proof. Figure 2 describes the truth table for the function h. The undetermined values of interest to us are indicated in the table by A and B. The following analysis shows that for every possible value of A and B, we can perfectly implement either NAND or $\mathrm{OR}_{2,1}$:

$$\begin{aligned} A = 0 &\implies \exists\, x\ h(x,y,z) = \neg y \bigvee z, \\ B = 0 &\implies \exists\, y\ h(x,y,z) = \neg x \bigvee z, \\ A = 1, B = 1 &\implies h(x,y,0) = \neg x \bigvee \neg y. \end{aligned}$$

Thus in each case we perfectly implement either the constraint NAND or $\mathrm{OR}_{2,1}$. □

6.3.3. Remaining cases. We now prove that if $\mathcal{F}$ is not strongly decidable, then deciding if there exists a nonzero solution is NP-hard. This is shown in Lemma 6.23. The last of the hardness results, claiming that finding a feasible solution is NP-hard if $\mathcal{F}$ is not 0-valid or 1-valid or 2cnf or weakly positive or weakly negative or linear, follows directly from Schaefer's theorem (Theorem 2.10).

LEMMA 6.23. *If* $\mathcal{F} \not\subseteq \mathcal{F}'$*, for any* $\mathcal{F}' \in \{\mathcal{F}_{\mathrm{S0}}, \mathcal{F}_1, \mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$*, then the problem of finding solutions of nonzero value to a given instance of (unweighted)*

x \ yz	00	01	11	10
0	1	-	1	A
1	B	1	-	0

FIG. 2. *Truth table of the constraint* $h(x, y, z)$.

MAX ONES($\mathcal{F}$) *is NP-hard.*

Proof. Assume, for simplicity, that all constraints of $\mathcal{F}$ have arity k. Given a constraint $f : \{0,1\}^k \to \{0,1\}$ and an index $i \in [k]$, let $f{\downarrow_i}$ be the constraint mapping $\{0,1\}^{k-1}$ to $\{0,1\}$ given by

$$f{\downarrow_i}(x_1, \ldots, x_k) \stackrel{\text{def}}{=} f(x_1, \ldots, x_{i-1}, 1, x_{i+1}, \ldots, x_k) \wedge f(x_1, \ldots, x_{i-1}, 0, x_{i+1}, \ldots, x_k).$$

Let $\mathcal{F}'$ be the set of constraints defined as follows:

$$\mathcal{F}' \stackrel{\text{def}}{=} \mathcal{F} \cup \{f{\downarrow_i} \quad | \quad f \in \mathcal{F}, i \in [k]\}.$$

We will show that deciding SAT($\mathcal{F}'$) is NP-hard and that the problem of deciding SAT($\mathcal{F}'$) reduces to finding nonzero solutions to MAX ONES($\mathcal{F}$).

First observe that $\mathcal{F}' \not\subseteq \mathcal{F}''$ for any $\mathcal{F}'' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\text{CNF}}, \mathcal{F}_{\text{A}}, \mathcal{F}_{\text{WP}}, \mathcal{F}_{\text{WN}}\}$. In particular it is not 0-valid, since $\mathcal{F}$ is not strongly 0-valid. Hence, once again applying Schaefer's result, we find that deciding SAT($\mathcal{F}'$) is NP-hard.

Given an instance of SAT($\mathcal{F}'$) on n variables $\mathbf{x}$ with m constraints $\mathbf{C}$, with $C_1, \ldots, C_{m'} \in \mathcal{F}$ and $C_{m'+1}, \ldots, C_m \in \mathcal{F}' \setminus \mathcal{F}$, consider the instance of MAX ONES($\mathcal{F}$) defined on variable set

$$w_1, \ldots, w_{k+1}, y_1, \ldots, y_n, z_1, \ldots, z_n$$

with the following constraints:

(1) Let f be a non-1-valid constraint in $\mathcal{F}$. We introduce the constraint $f(w_1, \ldots, w_k)$.
(2) For every constraint $C_i(v_{i_1}, \ldots, v_{i_k})$, $1 \le i \le m'$, we introduce two constraints $C_i(y_{i_1}, \ldots, y_{i_k})$ and $C_i(z_{i_1}, \ldots, z_{i_k})$.
(3) For every constraint $C_i(v_{i_1}, \ldots, v_{i_{k-1}})$, $m' + 1 \le i \le m$, we introduce $2(n + k + 1)$ constraints. For simplicity of notation, let $C_i(v_{i_1}, \ldots, v_{i_{k-1}}) = g(1, v_{i_1}, \ldots, v_{i_{k-1}}) \wedge g(0, v_{i_1}, \ldots, v_{i_{k-1}})$, where $g \in \mathcal{F}$. The $2(n+k+1)$ constraints are
   - $g(w_j, y_{i_1}, \ldots, y_{i_{k-1}})$ for $1 \le j \le k+1$,
   - $g(z_j, y_{i_1}, \ldots, y_{i_{k-1}})$ for $1 \le j \le n$,
   - $g(w_j, z_{i_1}, \ldots, z_{i_{k-1}})$ for $1 \le j \le k+1$,
   - $g(y_j, z_{i_1}, \ldots, z_{i_{k-1}})$ for $1 \le j \le n$.

We now show that the instance of MAX ONES($\mathcal{F}$) created above has a nonzero satisfying assignment if and only if the instance of SAT($\mathcal{F}'$) has a satisfying assignment. Let $s = s_1 s_2 \ldots s_k$ be a satisfying assignment for the non-1-valid constraint f chosen above. First if $v_1, \ldots, v_n$ form a satisfying assignment to the instance of SAT($\mathcal{F}'$), then we claim that the assignment $w_j = s_j$ for $1 \le j \le k$, $w_{k+1} = 1$ and $y_j = z_j = v_j$ for $1 \le j \le n$ is a satisfying assignment to the instance of MAX ONES($\mathcal{F}$) which has at least one 1 (namely, w_{k+1}). Conversely, let some nonzero setting $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n, z_1, \ldots, z_n$ satisfy the instance of MAX ONES($\mathcal{F}$). W.l.o.g.

assume that one of the variable $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n$ is a 1. Then we claim that the setting $v_j = z_j$, $1 \leq j \leq n$, satisfies the instance of SAT($\mathcal{F}'$). It is easy to see that the constraints $C_i(v_{i_1}, \ldots, v_{i_k})$, $1 \leq i \leq m'$, are satisfied. Now consider a constraint $C_i(v_{i_1}, \ldots, v_{i_{k-1}}) = g(0, v_{i_1}, \ldots, v_{i_{k-1}}) \wedge g(1, v_{i_1}, \ldots, v_{i_{k-1}})$. Since at least one of the variables in the set $w_1, \ldots, w_k$ is a 0 and at least one of the variables in the set $w_1, \ldots, w_{k+1}, y_1, \ldots, y_n$ is 1, we know that both $g(0, z_{i_1}, \ldots, z_{i_{k-1}})$ and $g(1, z_{i_1}, \ldots, z_{i_{k-1}})$ are satisfied, and hence $C_i(v_{i_1}, \ldots, v_{i_{k-1}}) = 1$. Thus the reduced instance of MAX ONES($\mathcal{F}$) has a nonzero satisfying assignment if and only if the instance of SAT($\mathcal{F}'$) is satisfiable. □

7. Classification of MIN CSP.

7.1. Preliminary results. We start with a simple equivalence between the complexity of the (WEIGHTED) MIN CSP problem for a function family and the family of functions obtained by complementing the 0's and 1's in its domain. Recall that for a function f, we defined f^- to be the function $f^-(\mathbf{x}) = f(\mathbf{1} - \mathbf{x})$, and for a function family $\mathcal{F}$, we defined $\mathcal{F}^- = \{f^- \mid f \in \mathcal{F}\}$.

PROPOSITION 7.1. *For every constraint family $\mathcal{F}$,* (WEIGHTED) MIN CSP($\mathcal{F}$) *is AP-reducible to* (WEIGHTED) MIN CSP($\mathcal{F}^-$).

Proof. The reduction substitutes every constraint $f(\mathbf{x})$ from $\mathcal{F}$ with the constraint $f^-(\mathbf{x})$ from $\mathcal{F}^-$. A solution for the latter problem is converted into a solution for the former one by complementing the value of each variable. The transformation preserves the cost of the solution. □

PROPOSITION 7.2. *If $\mathcal{F}$ is decidable, then* WEIGHTED MIN CSP($\mathcal{F}$) *is in* poly-APX *and is AP-reducible to* MIN CSP($\mathcal{F}$).

Proof. Given an instance $\mathcal{I}$ of WEIGHTED MIN ONES($\mathcal{F}$) with constraints $C_1, \ldots, C_m$ sorted in order of decreasing weight $w_1 \geq \cdots \geq w_m$. Let j be the largest index such that the constraints $C_1, \ldots, C_j$ are simultaneously satisfiable. Notice that j is computable in polynomial time and an assignment $\mathbf{a}$ satisfying $C_1, \ldots, C_j$ is computable in polynomial time. Then the solution $\mathbf{a}$ is an m-approximate solution to $\mathcal{I}$, since every solution must fail to satisfy at least one of the constraints $C_1, \ldots, C_{j+1}$ and thus have an objective of at least w_{j+1}, while $\mathbf{a}$ achieves an objective of at most $\sum_{i=j+1}^{m} w_i \leq m w_{j+1}$. Thus we conclude that WEIGHTED MIN CSP($\mathcal{F}$) is in poly-APX. The second part of the proposition follows by Lemma 3.11. □

7.2. Containment results (algorithms) for MIN CSP. We now show the containment results described in Theorem 2.13. Most results described here are simple containment results which follow easily from the notion of a "basis." The more interesting result here is a constant factor approximation algorithm for IHS-B which is presented in Lemma 7.3.

Recall that the classes contained in PO have already been dealt with in section 5.1. We now move on to APX-containment results.

LEMMA 7.3. *If $\mathcal{F} \subseteq \mathcal{F}_{\text{IHS}}$, then* WEIGHTED MIN CSP($\mathcal{F}$) $\in$ APX.

Proof. By Propositions 3.4 and 7.1 it suffices to prove the lemma for the problem WEIGHTED MIN CSP(IHS-B), where IHS-$B = \{\text{OR}_k | k \in [B]\} \cup \{\text{OR}_{2,1}, F\}$. We will show that for every B, WEIGHTED MIN CSP(IHS-B) is $B+1$-approximable.

Given an instance $\mathcal{I}$ of WEIGHTED MIN CSP(IHS-B) on variables $x_1, \ldots, x_n$ with constraints $C_1, \ldots, C_m$ with weights $w_1, \ldots, w_m$, we create a linear program on variables $y_1, \ldots, y_n$ (corresponding to the Boolean variables $x_1, \ldots, x_n$) and variables $z_1, \ldots, z_m$ (corresponding to the constraints $C_1, \ldots, C_m$). For every constraint C_j in

the instance $\mathcal{I}$ we create an LP constraint using the following transformation rules:

$$\begin{array}{lcll} C_j & : & x_{i_1} \bigvee \cdots \bigvee x_{i_k} \text{ for } k \le B & \rightarrow \quad z_j + y_{i_1} + \cdots + y_{i_k} \ge 1, \\ C_j & : & \neg x_{i_1} \bigvee x_{i_2} & \rightarrow \quad z_j + (1 - y_{i_1}) + y_{i_2} \ge 1, \\ C_j & : & \neg x_{i_1} & \rightarrow \quad z_j + (1 - y_{i_1}) \ge 1. \end{array}$$

In addition we add the constraints $0 \le z_j, y_i \le 1$ for every i, j. It may be verified that any integer solution to the above LP corresponds to an assignment to the MIN CSP problem with the variable z_j set to 1 if the constraint C_j is not satisfied. Thus the objective function for the LP is to minimize $\sum_j w_j z_j$.

Given any feasible solution vector $y_1, \ldots, y_n, z_1, \ldots, z_m$ to the LP above, we show how to obtain a 0/1 vector $y_1'', \ldots, y_n'', z_1'', \ldots, z_m''$ that is also feasible such that $\sum_j w_j z_j'' \le (B+1) \sum_j w_j z_j$.

First we set $y_i' = \min\{1, (B+1)y_i\}$ and $z_j' = \min\{1, (B+1)z_j\}$. Observe that the vector $y_1', \ldots, y_n', z_1', \ldots, z_m'$ is also feasible and gives a solution of value at most $(B+1)\sum_j w_j z_j$. We now show how to get an *integral* solution whose value is at most $\sum_j w_j z_j' \le (B+1)\sum_j w_j z_j$. For this part we first set $y_i'' = 1$ if $y_i' = 1$ and $z_j'' = 1$ if $z_i' = 1$. Now we remove every constraint in the LP that is made redundant. Notice in particular that every constraint of type $z_j + y_{i_1} + \cdots + y_{i_k} \ge 1$ is now redundant (either z_j'' or one of the y_i'''s has already been set to 1, and hence the constraint will be satisfied by any assignment to the remaining variables). We now observe that, on the remaining variables, the LP constructed above reduces to the following:

$$\begin{array}{lcll} \text{Minimize} & \sum_j w_j z_j & & \\ \text{Subject to} & y_{i_2} - y_{i_1} + z_j & \ge & 0, \\ & y_{i_2} + z_j & \ge & 1, \\ & -y_{i_1} + z_j & \ge & 0 \end{array}$$

with the y_i''s and z_j''s forming a feasible solution to the above LP. Notice further that every z_j occurs in at most one constraint above. Thus the above LP represents s-t min-cut problem and therefore has an optimal integral solution. We set z_j'''s and y_i'' to such an integral optimal solution. Notice that the solution thus obtained is integral and satisfies $\sum_j w_j z_j'' \le \sum_j w_j z_j' \le (B+1)\sum_j w_j z_j$. ∎

LEMMA 7.4. *For any family* $\mathcal{F} \subseteq \mathcal{F}_{2A}$, WEIGHTED MIN CSP($\mathcal{F}$) *A-reduces to* MIN CSP(XOR).

Proof. First we will argue that the family $\mathcal{F}' = \{\text{XOR}, T, F\}$ perfectly implements $\mathcal{F}$. By Proposition 3.4 it suffices to implement the basic width-2 affine functions, namely, the functions XOR, XNOR, T, and F. Every function except XNOR is already present in $\mathcal{F}'$ and by Proposition 3.3 XOR perfectly implements XNOR.

We conclude by observing that the family {XOR} is neither 0-valid nor 1-valid and hence, by Lemma 5.7, WEIGHTED MIN CSP($\mathcal{F}'$) A-reduces to WEIGHTED MIN CSP(XOR). Finally the weights can be removed using Proposition 7.2. ∎

Lemmas 7.5–7.7 show reducibility to MIN 2CNF DELETION, NEAREST CODEWORD, and MIN HORN DELETION.

LEMMA 7.5. *For any family* $\mathcal{F} \subseteq \mathcal{F}_{2CNF}$, *the family* $\{\text{OR}, \text{NAND}\} \stackrel{p}{\Longrightarrow} \mathcal{F}$ *and hence* WEIGHTED MIN CSP($\mathcal{F}$)$\le_A$ MIN 2CNF DELETION.

Proof. Again it suffices to consider the basic constraints of $\mathcal{F}$ and this is some subset of

$$\{\text{OR}_{2,0}, \text{OR}_{2,1}, \text{OR}_{2,2}, T, F\}.$$

The family $\{\mathrm{OR}, \mathrm{NAND}\}$ contains the first and the third functions. Since it contains a non-0-valid function, a non-1-valid function and a non-C-closed function, it can also implement T and F (by Lemma 4.6). This leaves the function $\mathrm{OR}_{2,1}$ which is implemented by the constraints $\mathrm{NAND}(x, z_{\mathrm{AUX}})$ and $\mathrm{OR}(y, z_{\mathrm{AUX}})$ (on the variables x and y). The A-reduction now follows from Lemma 3.10. □

LEMMA 7.6. *For any family $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{A}}$, the family $\{\mathrm{XOR}_3, \mathrm{XNOR}_3\}$ perfectly implements every function in $\mathcal{F}$. Thus* WEIGHTED MIN CSP$(\mathcal{F}) \leq_{\mathrm{A}}$ NEAREST CODEWORD.

Proof. It suffices to show implementation of the basic affine constraints, namely, constraints of the form XNOR_p and XOR_q for every $p, q \geq 1$. We focus on the former type, as the implementation of the latter is analogous. First, we observe that the constraint $\mathrm{XNOR}(x_1, x_2)$ is perfectly implemented by the constraints $\{\mathrm{XNOR}_3(x_1, x_2, z_1), \mathrm{XNOR}_3(x_1, x_2, z_2), \mathrm{XNOR}_3(x_1, x_2, z_3), \mathrm{XNOR}_3(z_1, z_2, z_3)\}$. Next, the constraint $F(x_1)$ can be perfectly implemented by $\{\mathrm{XNOR}(x_1, z_1), \mathrm{XNOR}(x_1, z_2), \mathrm{XNOR}(x_1, z_3), \mathrm{XNOR}_3(z_1, z_2, z_3)\}$. Finally, the constraint $\mathrm{XNOR}_p(x_1, \ldots, x_p)$ for any $p > 3$ can be implemented as follows. We introduce the following set of constraints using the auxiliary variables $z_1, z_2 \ldots, z_{p-2}$ and the set of constraints $\{\mathrm{XNOR}_3(x_1, x_2, z_1), \mathrm{XNOR}_3(z_1, x_3, z_2), \mathrm{XNOR}_3(z_2, x_4, z_3), \ldots, \mathrm{XNOR}_3(z_{p-2}, x_{p-1}, x_p)\}$ □

LEMMA 7.7. *For any family $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$, we have $\{\mathrm{OR}_{3,1}, T, F\} \overset{p}{\Longrightarrow} \mathcal{F}$ and thus* WEIGHTED MIN CSP$(\mathcal{F}) \leq_{\mathrm{A}}$ MIN HORN DELETION.

Proof. As usual, it suffices to perfectly implement every function in the basis $\{\mathrm{OR}_k \mid k \geq 1\} \cup \{\mathrm{OR}_{k,1} \mid k \geq 1\}$. The constraint $\mathrm{OR}(x, y)$ is implemented by the constraints $\mathrm{OR}_{3,1}(a, x, y)$ and $T(a)$. $\mathrm{OR}_{2,1}(x, y)$ is implemented by $\mathrm{OR}_{3,1}(x, y, a)$ and $F(a)$. The implementation of $\mathrm{OR}_3(x, y, z)$ is $\mathrm{OR}(x, a)$ and $\mathrm{OR}_{3,1}(a, y, z)$ (the constraint $\mathrm{OR}(x, a)$, in turn, may be implemented with the already shown method). Thus every k-ary constraint for $k \leq 3$ can be perfectly implemented by the family $\{\mathrm{OR}_{3,1}, T, F\}$). For $k \geq 4$, we use the textbook reduction from SAT to 3-SAT (see, e.g., [19, p. 49]) and we observe that when applied to k-ary weakly positive constraints it yields a perfect implementation using only 3-ary weakly positive constraints. □

To conclude this section we describe the trivial approximation algorithms for NEAREST CODEWORD and MIN HORN DELETION. They follow easily from Proposition 7.2 and the fact that both families are decidable.

COROLLARY 7.8 (to Proposition 7.2). MIN HORN DELETION *and* NEAREST CODEWORD *are in* poly-APX.

7.3. Hardness results (reductions) for MIN CSP.

LEMMA 7.9 (APX-hardness). *If $\mathcal{F} \not\subseteq \mathcal{F}'$, for $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}\}$, then* MIN CSP$(\mathcal{F})$ *is* APX*-hard.*

Proof. The proof essentially follows from Lemma 5.8 in combination with Proposition 3.7. We show that for every $\mathcal{F}$, MAX CSP$(\mathcal{F})$ AP-reduces to MIN CSP$(\mathcal{F})$. Let $\mathcal{I}$ be an instance of MAX CSP$(\mathcal{F})$ on n variables and m constraints. Let $\mathbf{x}'$ be a solution satisfying m/k constraints that can be found in polynomial time (by Proposition 3.7). Let $\mathbf{x}''$ be an r-approximate solution to the same instance $\mathcal{I}$ viewed as an instance of MIN CSP$(\mathcal{F})$. If OPT is the optimum solution to the maximization problem $\mathcal{I}$, then $\mathbf{x}''$ satisfies at least $m - r(m - \mathrm{OPT}) = r\mathrm{OPT} - (r-1)m$ constraints. Thus the better of the two solutions is an r'-approximate solution to the instance $\mathcal{I}$ of MAX CSP$(\mathcal{F})$, where

$$\begin{aligned} r' &\le \frac{\text{OPT}}{\max\{m/k, r\text{OPT} - (r-1)m\}} \\ &\le \frac{((r-1)k+1)\text{OPT}}{(r-1)k(m/k) + r\text{OPT} - (r-1)m} \\ &= \frac{1+(r-1)k}{r} \\ &\le 1 + (r-1)k. \end{aligned}$$

Thus MAX CSP($\mathcal{F}$) AP-reduces to MIN CSP($\mathcal{F}$). The lemma follows from the APX-hardness of MAX CSP($\mathcal{F}$) (Lemma 5.8). □

LEMMA 7.10 (MIN UNCUT-hardness). *If $\mathcal{F} \not\subseteq \mathcal{F}'$, for $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2M}, \mathcal{F}_{IHS}\}$, and $\mathcal{F} \subseteq \mathcal{F}_{2A}$, then* MIN CSP($\mathcal{F}$) *is* MIN UNCUT*-hard.*

Proof. Recall that MIN UNCUT-hardness requires that MIN CSP(XOR) be A-reducible to MIN CSP($\mathcal{F}$).

Let $f \in \mathcal{F}$. Consider (all) the minimally dependent sets of f. By Lemma 4.22 all such sets are of cardinality at most 2. For a minimally dependent set $\{i, j\}$ let

$$f_{i,j}(x_i, x_j) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k \text{ s.t. } f(x_1, \ldots, x_k).$$

By Lemma 4.17 all the $f_{i,j}$'s are affine and thus must be one of the functions $T(x_i)$, $F(x_i)$, XOR(x_i, x_j), or XNOR(x_i, x_j). Furthermore, f can be expressed as the conjunction of $f_{i,j}$'s over all the minimally dependent sets. It follows that there exist i, j such that $f_{i,j}(x_i, x_j) = \text{XOR}(x_i, x_j)$. (Otherwise f would be a conjunction of T, F and XNOR functions, all of which are in $\mathcal{F}_{IHS}$, and thus f would also be in $\mathcal{F}_{IHS}$.) Thus we conclude that f implements XOR and by Lemma 3.10 we conclude that MIN CSP(XOR) is A-reducible to MIN CSP($\mathcal{F}$) as desired. □

For the MIN 2CNF DELETION-hardness proof, we need the following three simple lemmas.

LEMMA 7.11. *If f is a* 2CNF *function which is not width-2 affine, then $f \stackrel{p}{\Longrightarrow} \text{OR}_{2,l}$ for some $l \in \{0, 1, 2\}$.*

Proof. For $i, j \in [k]$, let

$$f_{i,j}(x_i, x_j) \stackrel{\text{def}}{=} \exists x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_{j-1}, x_{j+1}, \ldots, x_k \text{ s.t. } f(x_1, \ldots, x_k).$$

Recall that if f can be expressed as the conjunction of $f_{i.j}$'s over all its maxterms and by Lemma 4.21, all the maxterms of f's have at most two literals in them. Thus $f(x_1, \ldots, x_k)$ can be expressed as $\bigwedge_{i,j \in [k]} f_{i,j}(x_i, x_j)$. It follows that some $f_{i,j}$ must be one of the functions $\text{OR}_{2,0}$, $\text{OR}_{2,1}$, or $\text{OR}_{2,2}$ (all other functions on two variables are affine). Thus existentially quantifying over all variables other than x_i and x_j, f perfectly implements $\text{OR}_{2,l}$ for some $l \in \{0, 1, 2\}$. □

LEMMA 7.12. *If $f \in \mathcal{F}_{2CNF}$ is not in IHS-B, then $f \stackrel{p}{\Longrightarrow}$ XOR.*

Proof. Once again we use the fact that f can be expressed as $\bigwedge_{i,j \in [k]} f_{i,j}(x_i, x_j)$, where $f_{i,j}$ is the function obtained from f by existentially quantifying over all variables other than x_i and x_j. It follows that one of the $f_{i,j}$'s must be NAND or XOR, since all the other functions on two variables are in IHS-$B+$. In the latter case we are done; otherwise we use the fact that f is not in IHS-$B-$ to conclude that f perfectly implements OR or XOR. In the latter case again we are done; otherwise we use the fact that f perfectly implements both the functions NAND and OR, and that NAND(x, y) and OR(x, y) perfectly implement XOR(x, y) to conclude that in this case too, the function f perfectly implements XOR. □

LEMMA 7.13. *If f is the function $\mathrm{OR}_{2,l}$ for some $l \in \{0,1,2\}$, then $\{f, \mathrm{XOR}\} \stackrel{p}{\Longrightarrow} \{\mathrm{OR}, \mathrm{NAND}\}$.*

Proof. The lemma follows from the fact that the function XOR essentially allows us to negate literals. For example, given the function $\mathrm{OR}_{2,1}(x,y)$ and XOR, the applications $\mathrm{OR}_{2,1}(x, z_{Aux})$ and $\mathrm{XOR}(z_{Aux}, y)$ perfectly and strictly implement the function $\mathrm{NAND}(x,y)$. Other implementations are obtained similarly. □

LEMMA 7.14 (MIN 2CNF DELETION-hardness). *If $\mathcal{F} \not\subseteq \mathcal{F}'$ for $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}, \mathcal{F}_{\mathrm{IHS}}, \mathcal{F}_{2\mathrm{A}}\}$, and $\mathcal{F} \subseteq \mathcal{F}_{2\mathrm{CNF}}$, then* MIN CSP($\mathcal{F}$) *is* MIN 2CNF DELETION*-hard.*

Proof. By Lemmas 7.11 and 7.12, $\mathcal{F}$ implements one of the functions $\mathrm{OR}_{2,l}$ for $l \in \{0,1,2\}$ and the function XOR. By Lemma 7.13 this suffices to implement the family $\{\mathrm{NAND}, \mathrm{OR}\}$. Thus by Lemma 3.10 we conclude that MIN CSP($\{\mathrm{OR}, \mathrm{NAND}\}$) A-reduces to MIN CSP($\mathcal{F}$). □

LEMMA 7.15. *If $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{A}}$, but $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}, \mathcal{F}_{\mathrm{IHS}}, \mathcal{F}_{2\mathrm{A}}\}$, then* MIN CSP($\mathcal{F}$) *is* NEAREST CODEWORD*-hard.*

Proof. By Lemma 4.18 we know that in this case $\mathcal{F}$ perfectly implements the constraint $x_1 \oplus \cdots \oplus x_p = b$ for some $p \geq 3$ and some $b \in \{0,1\}$. Thus the family $\mathcal{F} \cup \{T, F\}$ implements the functions $x \oplus y \oplus z = 0, x \oplus y \oplus z = 1$. Thus NEAREST CODEWORD =MIN CSP($\{x \oplus y \oplus z = 0, x \oplus y \oplus z = 1\}$ is A-reducible to MIN CSP($\mathcal{F} \cup \{F, T\}$). Since $\mathcal{F}$ is neither 0-valid nor 1-valid, we can use Lemma 5.7 to conclude that MIN CSP($\mathcal{F}$) is NEAREST CODEWORD-hard. □

The next lemma describes the best known hardness of approximation for the NEAREST CODEWORD problem. The result relies on an assumption stronger than NP ≠ P.

LEMMA 7.16 (see [2]). *For every $\epsilon > 0$,* NEAREST CODEWORD *is hard to approximate to within a factor of $\Omega(2^{\log^{1-\epsilon} n})$ unless* NP *has deterministic algorithms running in time $n^{\log^{O(1)} n}$.*

Proof. The required hardness of the NEAREST CODEWORD problem is shown by Arora et al. [2]. The NEAREST CODEWORD problem, as defined in Arora et al., works with the following problem: Given an $m \times n$ matrix A and an m-dimensional vector b, find an n-dimensional vector x which minimizes the Hamming distance between Ax and b. Thus this problem can be expressed as a MIN CSP problem with m affine constraints over n-variables. The only technical point to be noted is that these constraints have unbounded arity. In order to get rid of such long constraints, we replace a constraint of the form $x_1 \oplus \cdots \oplus x_l = 0$ into $l-2$ constraints $x_1 \oplus x_2 \oplus z_1 = 0$, $z_1 \oplus x_3 \oplus z_2 = 0$, etc. on auxiliary variables $z_1, \ldots, z_{l-3}$. (The same implementation was used in Lemma 7.6.) This increases the number of constraints by a factor of at most n but does not change the objective function. Thus if M represents the number of constraints in the new instance of the problem, then the approximation hardness which is $2^{\log^{1-\epsilon} m}$ can be expressed as $2^{\frac{1}{2}\log^{1-\epsilon} M}$ which is still growing faster than, say, $2^{\log^{1-2\epsilon} M}$. Since the result of [2] holds for every positive ϵ, we still get the desired result claimed above. □

It remains to see the MIN HORN DELETION-hard case. We will have to draw some nontrivial consequences from the fact that a family is not IHS-B.

LEMMA 7.17. *Assume $\mathcal{F} \not\subseteq \mathcal{F}_{\mathrm{IHS}}$ and either $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$ or $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WN}}$. Then $\mathcal{F}$ contains a function that is not C-closed.*

Proof. Let f be a C-closed function in $\mathcal{F}_{\mathrm{WP}}$ ($\mathcal{F}_{\mathrm{WN}}$). We claim that all of f's maxterms must be of the form $T(x_i)$, $F(x_i)$, or $\mathrm{OR}_{2,1}(x_i, x_j)$. If not, then since f is C-closed, the maxterm involving the complementary literals is also a maxterm of f,

x1 \ x2 x3	00	01	11	10
0	1	A	B	D
1	0	1	C	1

FIG. 3. *Truth table of the constraint f_2.*

but the complementary maxterm is not weakly positive (and by Lemma 4.20 every maxterm of f must be weakly positive). However, if all of f's maxterms are of the form $T(x_i)$, $F(x_i)$, or $\mathrm{OR}_{2,1}(x_i, x_j)$, then f is in IHS-B. The lemma follows from the fact that $\mathcal{F} \not\subseteq \mathcal{F}_{\mathrm{IHS}}$. □

LEMMA 7.18. *If f is a weakly positive function not expressible as IHS-$B+$, then $\{f, T, F\} \stackrel{p}{\Longrightarrow} \mathrm{OR}_{3,1}$. If f is a weakly negative function not expressible as IHS-$B-$, then $\{f, T, F\} \stackrel{p}{\Longrightarrow} \mathrm{OR}_{3,2}$.*

Proof. Let f be a weakly positive function. By Lemma 4.20 all maxterms of f are weakly positive. Since f is not IHS-$B+$, f must have a maxterm of the form $(\neg x_1 \bigvee x_2 \bigvee \cdots \bigvee x_p)$ for some $p \geq 3$. We first show that $\{f, F\}$ can perfectly implement the function XNOR. To get the former, consider the function

$$f_1(x_1, x_2) \stackrel{\mathrm{def}}{=} \exists x_{p+1}, \ldots, x_k \text{ s.t. } f(x_1, x_2, 0^{p-2}, x_{p+1}, \ldots, x_k).$$

The function f_1 satisfies the properties $f_1(10) = 0$, $f_1(00) = f_1(11) = 1$. Thus f_1 is either the function XNOR or $\mathrm{OR}_{2,1}$. Notice that the constraints $f(x_1, \ldots, x_k)$ and $F(x_i)$, $i \in \{3, \ldots, p\}$, perfectly implement f_1. Thus $\{f, F\}$ perfectly implement either the function XNOR or $\mathrm{OR}_{2,1}$. In the former case, we have the claim and in the latter case we use the fact that the constraints $\mathrm{OR}_{2,1}(x, y)$ and $\mathrm{OR}_{2,1}(y, x)$ perfectly implement $\mathrm{XNOR}(x, y)$.

Next, we show how the family $\{f, T, F, \mathrm{XNOR}\}$ (and hence $\{f, T, F\}$) can perfectly implement $\mathrm{OR}_{2,1}$. To do so, we consider the function

$$f_2(x_1, x_2, x_3) \stackrel{\mathrm{def}}{=} \exists x_{p+1}, \ldots, x_k \text{ s.t. } f(x_1, x_2, x_3, 0^{p-3}, x_{p+1}, \ldots, x_k).$$

Again $\{f, F\}$ implement f_2 perfectly. By the definition of a maxterm, we find that f_2 satisfies the following properties: $f_2(100) = 0$ and $f_2(000) = f_2(110) = f_2(101) = 1$. Figure 3 gives the truth table for f_2, where the unknown values are denoted by A, B, C, and D. If $C = 0$, then restricting $x_1 = 1$ gives the constraint $\mathrm{XOR}(x_2, x_3)$. However, notice that XOR is not a weakly positive function and by Lemma 4.19 every function obtained by setting some of the variables in a weakly positive function to constants and existentially quantifying over some other subset of variables is a weakly positive function. Thus $C = 1$. If $D = 1$, we implement the function $\mathrm{OR}_{2,1}(x_1, x_2)$ by the constraints $f_2(x_1, x_2, x_3)$ and $F(x_3)$. Otherwise we have $D = 0$, and the constraints $f_2(x_1, x_2, x_3)$ and $\mathrm{XNOR}(x_1, x_3)$ implement the constraint $\mathrm{OR}_{2,1}(x_2, x_1)$.

Finally we conclude by observing that the constraints $f_2(x, z^1, z^2)$, $\mathrm{OR}_{2,1}(z^1, y)$ and $\mathrm{OR}_{2,1}(z^2, z)$, perfectly implement the constraint $\mathrm{OR}_{3,1}(x, y, z)$.

This completes the proof for the first part. The proof if f is weakly negative is similar. □

LEMMA 7.19 (the MIN HORN DELETION-hard case). *If $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_1, \mathcal{F}_{2\mathrm{M}}, \mathcal{F}_{\mathrm{IHS}}, \mathcal{F}_{2\mathrm{A}}, \mathcal{F}_{2\mathrm{CNF}}\}$, and either $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$ or $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WN}}$, then* WEIGHTED MIN CSP($\mathcal{F}$) *is* MIN HORN DELETION*-hard.*

Proof. From Lemma 7.18 we have that either MIN CSP($\{\mathrm{OR}_{3,1}, T, F\}$ or MIN CSP($\{\mathrm{OR}_{3,2}, T, F\}$ is A-reducible to MIN CSP($\mathcal{F}$). Furthermore, since $\mathcal{F}$ is not 0-valid or 1-valid we have that MIN CSP($\mathcal{F} \cup \{T, F\}$) is A-reducible to MIN CSP($\mathcal{F}$). The lemma follows by an application of Proposition 7.1 which shows that the problems MIN CSP($\{\mathrm{OR}_{3,1}, T, F\}$) A-reduces to MIN CSP($\{\mathrm{OR}_{3,2}, T, F\}$). ∎

To show the hardness of MIN HORN DELETION we define a variant of the "label cover" problem. The original definition from [2] used a different objective function. Our variant is similar to the one used by Amaldi and Kann [1] under the name TOTAL LABEL COVER.

DEFINITION 7.20 (TOTAL LABEL COVER$_p$).

INSTANCE. *An instance is described by sets $\mathcal{R}$, $\mathcal{Q}$, and $\mathcal{A}$ and by p functions (given by their tables) $Q_1, \ldots, Q_p : \mathcal{R} \to \mathcal{Q}$ and a function* ACC $: \mathcal{R} \times (\mathcal{A})^p \to \{0, 1\}$.

FEASIBLE SOLUTIONS. *A solution is a collection of p functions $A_1, \ldots, A_p : \mathcal{Q} \to 2^{\mathcal{A}}$. The solution is feasible if for every $R \in \mathcal{R}$, there exists $a_1 \in A_1(Q_1(R)), \ldots, a_p \in A_p(Q_p(R))$ such that* ACC$(R, a_1, \ldots, a_p) = 1$.

OBJECTIVE. *The objective is to minimize* $\sum_{i=1}^{p} \sum_{q \in Q} |A_i(q)|$.

In Appendix A, we show how results from interactive proofs imply the hardness of approximating MIN LABEL-COVER to within a factor of $2^{\log^{1-\epsilon} n}$. We now use this result to show that hardness of MIN HORN DELETION.

LEMMA 7.21. *For every $\epsilon > 0$,* MIN HORN DELETION *is* NP*-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$.*

Proof. Let p be such that MIN LABEL-COVER$_p$ is NP-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$. (By Lemma A.3 such a p exists.) We now reduce MIN LABEL-COVER$_p$ to MIN HORN DELETION.

Let $(Q_1, \ldots, Q_p, \mathrm{ACC})$ be an instance of MIN LABEL-COVER$_p$, where $Q_i : \mathcal{R} \to \mathcal{Q}$ and ACC $: \mathcal{R} \times (\mathcal{A})^p \to \{0, 1\}$. For any $R \in \mathcal{R}$, we define $Acc(R) = \{(a_1, \ldots, a_p) : V(R, a_1, \ldots, a_p) = 1\}$.

We now describe the reduction. For any $R \in \mathcal{R}$, $a_1, \ldots, a_p \in \mathcal{A}$, we have a variable $v_{R,a_1,\ldots,a_p}$ whose intended meaning is the value of ACC$(R, a_1, \ldots, a_p)$. Moreover, for every $i \in [p]$, $Q \in \mathcal{Q}$, and $a \in A_i$ we have a variable $x_{i,Q,a}$ with the intended meaning being that its value is 1 if and only if $a \in A_i(Q)$. For any $x_{i,Q,a}$ we have the weight-one constraint $\neg x_{i,q,a}$. The following constraints (each with weight $(p \times |\mathcal{Q}| \times |\mathcal{A}|)$) enforce the variables to have their intended meaning. Due to their weight, it is never convenient to contradict them.

$$\begin{array}{ll} \forall R \in \mathcal{R}: & \bigvee_{(a_1,\ldots,a_p) \in Acc(R)} v_{R,a_1,\ldots,a_p} \\ \forall R \in \mathcal{R}, a_1, \ldots, a_p \in \mathcal{A}, i \in [p]: & v_{R,a_1,\ldots,a_p} \Rightarrow x_{i,Q_i(R),a_i}. \end{array}$$

The constraints of the first kind can be perfectly implemented with OR_3 and $\mathrm{OR}_{3,1}$ (see Lemma 7.7). It can be checked that this is an AP-reduction from MIN LABEL-COVER$_p$ to MIN HORN DELETION and thus the lemma follows. ∎

8. MIN ONES classification.

8.1. Preliminaries: MIN ONES vs. MIN CSP.

We start with the following easy relation between MIN CSP and MIN ONES problems. Recall that a family $\mathcal{F}$ is decidable if membership in SAT($\mathcal{F}$) is decidable in polynomial time.

PROPOSITION 8.1. *For any decidable constraint family $\mathcal{F}$,* WEIGHTED MIN ONES($\mathcal{F}$) *AP-reduces to* WEIGHTED MIN CSP($\mathcal{F} \cup \{F\}$).

Proof. Let $\mathcal{I}$ be an instance of WEIGHTED MIN ONES($\mathcal{F}$) over variables $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$. Let $w_{\max}$ be the largest weight. We construct an instance $\mathcal{I}'$ of WEIGHTED MIN CSP($\mathcal{F} \cup \{F\}$) by leaving the constraints of $\mathcal{I}$ (each with weight $nw_{\max}$) and adding a constraint $F(x_i)$ of weight w_i for any $i = 1, \ldots, n$. Notice that whenever $\mathcal{I}$ is feasible, the optimum value for $\mathcal{I}$ equals the optimum value for $\mathcal{I}'$. Given an r-approximate solution to $\mathbf{x}$ to $\mathcal{I}'$, we check to see if $\mathcal{I}$ is feasible and if so find any feasible solution $\mathbf{x}'$ and output solution (from among $\mathbf{x}$ and $\mathbf{x}'$) that achieves a lower objective. It is clear that the solution is at least an r-approximate solution if $\mathcal{I}$ is feasible. □

Reducing a MIN CSP problem to a MIN ONES problem is slightly less general.

PROPOSITION 8.2. *For any function f, let f' and f'' denote the functions $f'(\mathbf{x}, y) = \mathrm{OR}(f(\mathbf{x}), y)$ and $f''(\mathbf{x}, y) = \mathrm{XOR}(f(\mathbf{x}), y)$, respectively. If constraint families $\mathcal{F}$ and $\mathcal{F}'$ are such that for every $f \in \mathcal{F}$, f' or f'' is in $\mathcal{F}'$, then* WEIGHTED MIN CSP($\mathcal{F}$) *AP-reduces to* WEIGHTED MIN ONES($\mathcal{F}'$).

Proof. Given an instance $\mathcal{I}$ of WEIGHTED MIN CSP($\mathcal{F}$) we create an instance $\mathcal{I}'$ of WEIGHTED MIN ONES($\mathcal{F}'$) as follows: For every constraint C_j we introduce an auxiliary variable y_j. The variable takes the same weight as the constraint C_j in $\mathcal{I}$. The original variables are retained with weight zero. If the constraint $C_j(\mathbf{x}) \bigvee y_j$ is a constraint of $\mathcal{F}'$ we apply that constraint; otherwise we apply the constraint $C_j(\mathbf{x}) \oplus y = 1$. Given an assignment to the variables of $\mathcal{I}$, notice that by setting $y_j = \neg C_j$, we get a feasible solution to $\mathcal{I}'$ with the same objective value; conversely, a feasible solution to $\mathcal{I}'$ when projected onto the variables $\mathbf{x}$ gives a solution with the same value to the objective function of $\mathcal{I}$. This shows that the optimum value to $\mathcal{I}'$ equals that of $\mathcal{I}$ and that an r-approximate solution to $\mathcal{I}'$ projects to give an r-approximate solution to $\mathcal{I}$. □

Finally the following easy proposition is invoked at a few places.

PROPOSITION 8.3. *If $\mathcal{F} \Longrightarrow f$, then $\mathcal{F}^- \Longrightarrow f^-$.*

8.2. Containment results for MIN ONES.

LEMMA 8.4 (PO containment). *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{\mathrm{WN}}, \mathcal{F}_{\mathrm{2A}}\}$, then* WEIGHTED MIN ONES($\mathcal{F}$) *is solvable exactly in polynomial time.*

Proof. The proof follows from Lemma 6.5 and from the observation that for any family $\mathcal{F}$, solving WEIGHTED MIN ONES($\mathcal{F}$) to optimality reduces to solving WEIGHTED MAX ONES($\mathcal{F}^-$) to optimality. □

LEMMA 8.5. *If $\mathcal{F} \subseteq \mathcal{F}'$ for $\mathcal{F}' \in \{\mathcal{F}_{\mathrm{2CNF}}, \mathcal{F}_{\mathrm{IHS}}\}$, then* WEIGHTED MIN ONES($\mathcal{F}$) *is in* APX.

Proof. For the case $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{2CNF}}$, a 2-approximate algorithm is given by Hochbaum et al. [25].

Now consider the case $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{IHS}}$. From Proposition 3.4 it is sufficient to consider only basic IHS-B constraints. Since IHS-$B-$ constraints are weakly negative, we will restrict ourselves to basic IHS-$B+$ constraints. We use linear programming relaxations and deterministic rounding. Let k be the maximum arity of a function in $\mathcal{F}$; we will give a k-approximate algorithm. Let $\phi = \{C_1, \ldots, C_m\}$ be an instance of WEIGHTED MIN ONES($\mathcal{F}$) over variable set $X = \{x_1, \ldots, x_n\}$ with weights $w_1, \ldots, w_n$. The following is an integer linear programming formulation of finding the minimum weight satisfying assignment for ϕ.

$$\begin{array}{lll}
\text{Minimize} & \sum_i w_i y_i, & \\
\text{Subject to} & & \\
& y_{i_1} + \cdots + y_{i_h} \geq 1 & \forall (x_{i_1} \bigvee \cdots \bigvee x_{i_h}) \in \phi, \\
& y_{i_1} - y_{i_2} \geq 0 & \forall (x_{i_1} \bigvee \neg x_{i_2}) \in \phi, \qquad \text{(SCB)} \\
& y_i = 0 & \forall \neg x_i \in \phi, \\
& y_i = 1 & \forall x_i \in \phi, \\
& y_i \in \{0,1\} & \forall i \in \{1,\ldots,n\}.
\end{array}$$

Now consider the linear programming relaxation obtained by relaxing the $y_i \in \{0,1\}$ constraints into $0 \leq y_i \leq 1$. We first find an optimum solution $\mathbf{y}^*$ for the relaxation, and then we define a 0/1 solution by setting $y_i = 0$ if $y_i^* < 1/k$ and $y_i = 1$ if $y_i^* \geq 1/k$. It is easy to see that this rounding increases the cost of the solution at most k times and that the obtained solution is feasible for (SCB). □

LEMMA 8.6. *For any $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{A}}$,* WEIGHTED MIN ONES($\mathcal{F}$) *is A-reducible to* NEAREST CODEWORD.

Proof. From Lemmas 7.6 and 3.9 we have that WEIGHTED MIN ONES($\mathcal{F}$) is A-reducible to WEIGHTED MIN ONES($\{\mathrm{XNOR}_3, \mathrm{XOR}_3\}$). From Proposition 8.1, we have that WEIGHTED MIN ONES($\mathcal{F}$) A-reduces to WEIGHTED MIN CSP($\{\mathrm{XOR}_3$, $\mathrm{XNOR}_3, F\}$). Notice further that the family $\{\mathrm{XNOR}_3, \mathrm{XOR}_3\}$ can implement F (by Lemma 4.6). Thus we have that WEIGHTED MIN ONES($\mathcal{F}$) A-reduces to WEIGHTED MIN CSP($\{\mathrm{XOR}_3, \mathrm{XNOR}_3, \}$) = NEAREST CODEWORD. □

LEMMA 8.7. *For any $\mathcal{F} \subseteq \mathcal{F}_{\mathrm{WP}}$,* WEIGHTED MIN ONES($\mathcal{F}$) *AP-reduces to* MIN HORN DELETION.

Proof. The proof follows from the following sequence of assertions:

(1) $\{\mathrm{OR}_{3,1}, T, F\}$ perfectly implements $\mathcal{F}$ (Lemma 7.7).

(2) WEIGHTED MIN ONES($\mathcal{F}$) AP-reduces to WEIGHTED MIN ONES($\{\mathrm{OR}_{3,1}, T, F\}$) (Lemma 3.9).

(3) WEIGHTED MIN ONES($\{\mathrm{OR}_{3,1}, T, F\}$) AP-reduces to WEIGHTED MIN CSP ($\{\mathrm{OR}_{3,1}, T, F\}$) = MIN HORN DELETION (Proposition 8.1). □

PROPOSITION 8.8. *If $\mathcal{F}$ is decidable, then* MIN ONES($\mathcal{F}$) *is in* poly-APX.

Proof. The proposition follows immediately from the fact that in this case it is easy to determine if the input instance is feasible and if so, if the optimum value is zero. If so we output the $\mathbf{0}$ as the solution; otherwise we output any feasible solution. Since the objective is at least 1 and the solution has value at most n, this is an n-approximate solution. □

8.3. Hardness results for MIN ONES. We start by considering the hardest problems first. The case when $\mathcal{F}$ is not decidable is immediate. We move to the case where $\mathcal{F}$ may be 1-valid but not in any other of Schaefer's easy classes.

LEMMA 8.9. *If $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{2\mathrm{CNF}}, \mathcal{F}_{\mathrm{A}}, \mathcal{F}_{\mathrm{WP}}, \mathcal{F}_{\mathrm{WN}}\}$, then* WEIGHTED MIN ONES($\mathcal{F}$) *is hard to approximate to within any factor, and* MIN ONES($\mathcal{F}$) *is* poly-APX-*hard.*

Proof. We first show how to handle the weighted case. The hardness for the unweighted case will follow easily. Consider a function $f \in \mathcal{F}$ which is not weakly positive. For such an f, there exists assignments $\mathbf{a}$ and $\mathbf{b}$ such that $f(\mathbf{a}) = 1$ and $f(\mathbf{b}) = 0$ and $\mathbf{a}$ is zero in every coordinate where $\mathbf{b}$ is zero. (Such an input pair exists for every nonmonotone function f and every monotone function is also weakly positive.) Now let f' be the constraint obtained from f by restricting it to inputs where $\mathbf{b}$ is one and setting all other inputs to zero. Then f' is a satisfiable function which is not 1-valid. We can now apply Schaefer's theorem [42] to conclude that

SAT($\mathcal{F} \cup \{f'\}$) is hard to decide. We now reduce an instance of deciding SAT($\mathcal{F} \cup \{f'\}$) to approximating WEIGHTED MIN CSP($\mathcal{F}$). Given an instance $\mathcal{I}$ of SAT($\mathcal{F} \cup \{f'\}$) we create an instance which has some auxiliary variables $W_1, \ldots, W_k$ which are all supposed to be zero. This in enforced by giving them very large weights. We now replace every occurrence of the constraint f' in $\mathcal{I}$ by the constraint f on the corresponding variables with the W_i's in place which were set to zero in f to obtain f'. It is clear that if a "small" weight solution exists to the resulting WEIGHTED MIN CSP problem, then $\mathcal{I}$ is satisfiable; otherwise it is not. Thus we conclude it is NP-hard to approximate WEIGHTED MIN CSP to within any bounded factors.

For the unweighted case, it suffices to observe that by using polynomially bounded weights above, we get a poly-APX hardness. Furthermore, one can get rid of weights entirely by replicating variables. □

We may now restrict our attention to function families $\mathcal{F}$ that are 2CNF or affine or weakly positive or weakly negative or 0-valid. In particular, by the containment results shown in the previous section, in all such cases the problem WEIGHTED MIN ONES($\mathcal{F}$) is in poly-APX. We now give a weight-removing lemma which allows us to focus on showing the hardness of the weighted problems.

LEMMA 8.10. *If $\mathcal{F} \subseteq \mathcal{F}'$ for some $\mathcal{F}' \in \{\mathcal{F}_{2\text{CNF}}, \mathcal{F}_{\text{A}}, \mathcal{F}_{\text{WP}}, \mathcal{F}_{\text{WN}}, \mathcal{F}_0\}$, then* WEIGHTED MIN ONES($\mathcal{F}$) *AP-reduces to* MIN ONES($\mathcal{F}$).

Proof. By Lemma 3.11 it suffices to verify that WEIGHTED MIN ONES($\mathcal{F}$) is in poly-APX in all cases. If $\mathcal{F}$ is weakly negative or 0-valid, then this follows from Lemma 8.4. If $\mathcal{F}$ is 2CNF, then this follows from Lemma 8.5. If $\mathcal{F}$ is affine or weakly positive, then it A-reduces to NEAREST CODEWORD or MIN HORN DELETION, respectively, which are in poly-APX by Corollary 7.8. □

Before dealing with the remaining cases, we prove one more lemma that is useful in dealing with MIN ONES problems.

LEMMA 8.11. *For every constraint family $\mathcal{F}$ such that $\mathcal{F} \cup \{F\}$ is decidable,* WEIGHTED MIN ONES($\mathcal{F} \cup \{F\}$) *AP-reduces to* WEIGHTED MIN ONES($\mathcal{F}$).

Proof. Given an instance $\mathcal{I}$ of WEIGHTED MIN ONES($\mathcal{F} \cup \{F\}$) on n variables $x_1, \ldots, x_n$ with weights $w_1, \ldots, w_n$ we create an instance $\mathcal{I}'$ of WEIGHTED MIN ONES($\mathcal{F}$) on the variables $x_1, \ldots, x_n$ using all the constraints of $\mathcal{I}$ that are from $\mathcal{F}$; and for every variable x_i such that $F(x_i)$ is a constraint of $\mathcal{I}$, we increase the weight of the variable x_i to $nw_{\max}$, where $w_{\max}$ is the maximum of the weights $w_1, \ldots, w_n$. As in the proof of Proposition 8.1 we observe that if $\mathcal{I}$ is feasible, then the optima for $\mathcal{I}$ and $\mathcal{I}'$ are equal and given an r-approximate solution to $\mathcal{I}'$ we can find an r-approximate solution to $\mathcal{I}$. Furthermore, since $\mathcal{F} \cup \{F\}$ is decidable, we can decide whether or not $\mathcal{I}$ is feasible. □

We now deal with the affine problems.

LEMMA 8.12. *If $\mathcal{F}$ is affine but not width-2 affine or 0-valid, then* MIN ONES(XOR_3) *is AP-reducible to* WEIGHTED MIN ONES($\mathcal{F}$).

Proof. Notice that since $\mathcal{F}$ is affine, so is $\mathcal{F}^-$. Furthermore, $\mathcal{F}^-$ is neither width-2 affine nor 1-valid. Thus by Lemma 6.10 $\mathcal{F}^-$ perfectly implements either the family $\{\text{XNOR}_3\}$ or the family $\{\text{XOR}, \text{XNOR}_4\}$. Thus, by applying Proposition 8.3, we get that $\mathcal{F}$ implements either XOR_3 or the family $\{\text{XOR}, \text{XNOR}_4\}$. In the former case, we are done (by Lemma 3.9). In the latter case, notice that the constraints $\text{XNOR}_4(x_1, x_2, x_3, x_5)$ and $\text{XOR}(x_4, x_5)$ perfectly implement the constraint $\text{XOR}_4(x_1, x_2, x_3, x_5)$. Thus we conclude that WEIGHTED MIN ONES(XOR_4) is AP-reducible to WEIGHTED MIN ONES($\mathcal{F}$). Finally we use Lemma 8.11 to conclude that the family WEIGHTED MIN ONES($\mathcal{F}$)($\{\text{XOR}\}|_0$) is AP-reducible to WEIGHTED MIN

ONES($\mathcal{F}$). The lemma follows from the fact that $\mathrm{XOR}_3 \in \{\mathrm{XOR}_4\}|_0$. □

LEMMA 8.13. *If $\mathcal{F}$ is affine but not width-2 affine or 0-valid, then, for every $\epsilon > 0$,* MIN ONES($\mathcal{F}$) *is* NEAREST CODEWORD*-hard and hard to approximate to within a factor of $\Omega(2^{\log^{\epsilon} n})$.*

Proof. The proof follows from the following sequence of reductions:

NEAREST CODEWORD
$=$ WEIGHTED MIN CSP($\{\mathrm{XOR}_3, \mathrm{XNOR}_3\}$)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\{\mathrm{XOR}_4, \mathrm{XNOR}_4\}$) (using Proposition 8.2)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\{\mathrm{XOR}_3, \mathrm{XOR}\}$) (see below)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES(XOR_3) (using Lemma 8.11)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\mathcal{F}$) (using Lemmas 8.12 and 3.9)
$\leq_{\mathrm{AP}}$MIN ONES($\mathcal{F}$) (using Lemma 8.10).

The second reduction above follows by combining Lemma 3.9 with the observation that the family $\{\mathrm{XOR}_3, \mathrm{XOR}\}$ perfectly implement the functions XOR_4 and XNOR_4 as shown next. The constraints $\mathrm{XOR}_3(u, v, w)$ and $\mathrm{XOR}_3(w, x, y)$ perfectly implement the constraint $\mathrm{XNOR}_4(u, v, x, y)$; the constraints $\mathrm{XOR}_4(u, v, w, x)$ and $\mathrm{XOR}(w, y)$ perfectly implement $\mathrm{XOR}_4(u, v, x, y)$. The hardness of approximation of NEAREST CODEWORD is given by Lemma 7.16. □

LEMMA 8.14. *If $\mathcal{F}$ is weakly positive and not IHS-B (nor 0-valid), then* MIN ONES($\mathcal{F}$) *is* MIN HORN DELETION*-hard, and hence hard to approximate within $2^{\log^{1-\epsilon} n}$ for any $\epsilon > 0$.*

Proof. The proof follows from the following sequence of reductions:

MIN HORN DELETION
$=$ WEIGHTED MIN CSP($\{\mathrm{OR}_{3,1}, T, F\}$
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\{\mathrm{OR}_{4,1}, \mathrm{OR}_2, \mathrm{OR}_{2,1}\}$) (using Proposition 8.2)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\{\mathrm{OR}_{3,1}, T, F\}$) (using Lemmas 7.7 and 3.9)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\mathcal{F} \cup \{T, F\}$) (using Lemmas 7.18 and 3.9)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\mathcal{F} \cup \{F\}$) (using Lemma 4.6 to perfectly implement T)
$\leq_{\mathrm{AP}}$WEIGHTED MIN ONES($\mathcal{F}$) (using Lemma 8.11)
$\leq_{\mathrm{AP}}$MIN ONES($\mathcal{F}$) (using Lemma 8.10).

The hardness of approximation follows from Lemma 7.21. □

LEMMA 8.15. MIN ONES(OR) *is* APX*-hard.*

Proof. We reduce VERTEX COVER to MIN ONES(OR). Given a graph G on n vertices, we construct an instance of MIN ONES(OR) on n variables $x_1, \ldots, x_n$. For every edge between vertex i and j of G, we create a constraint $\mathrm{OR}(x_i, x_j)$. We notice that there is a one-to-one correspondence between an assignment to the variables and vertex covers in G (with variables assigned 1 corresponding to vertices in the cover) and the minimum vertex cover minimizes the sum of the variables. The lemma follows from the fact that VERTEX COVER is APX-hard [39, 3]. □

LEMMA 8.16 (APX-hardness). *If $\mathcal{F} \not\subseteq \mathcal{F}'$ for any $\mathcal{F}' \in \{\mathcal{F}_0, \mathcal{F}_{\mathrm{WN}}, \mathcal{F}_{2\mathrm{A}}\}$, then* MIN ONES($\mathcal{F}$) *is* APX*-hard.*

Proof. We mimic the proof of Lemma 6.14. We assume that $\mathcal{F}$ is not affine—the case where $\mathcal{F}$ is affine is shown to be NEAREST CODEWORD-hard in Lemma 8.13. By

Lemma 8.10 it suffices to show that WEIGHTED MIN ONES($\mathcal{F}$) is APX-hard; and by Lemma 8.11 it suffices to show that WEIGHTED MIN ONES($\mathcal{F} \cup \{F\}$) is APX-hard. Since $\mathcal{F} \cup \{F\}$ is not 0-valid or 1-valid or C-closed, it implements every function in $\mathcal{F} \cup \{T, F\}$ and thus every function in $\mathcal{F}|_{0,1}$. We now shift focus on to the family $(\mathcal{F}|_{0,1})^-$. Furthermore, $(\mathcal{F}|_{0,1})^-$ is neither weakly positive nor affine and thus by Lemmas 6.20 and 6.21 it implements NAND. Using Proposition 8.3 we get that $\mathcal{F}_{0,1}$ implements OR. Using Lemma 8.15 we get that WEIGHTED MIN ONES(OR) is APX-hard. Thus we conclude that WEIGHTED MIN ONES($\mathcal{F}$) is APX-hard. ∎

Appendix A. Hardness of TOTAL LABEL COVER.

DEFINITION A.1. *$L \in \mathrm{MIP}_{c,s}[p, r, q, a]$ if there exists a polynomial time bounded probabilistic oracle machine V (verifier) such that on input $x \in \{0,1\}^n$, the verifier picks a random string $R \in \{0,1\}^{r(n)}$ and generates p queries $Q_1 = Q_1(x, R), \ldots, Q_p = Q_p(x, R) \in \{0,1\}^{q(n)}$ and sends query Q_i to prover Π_i and receives from prover Π_i an answer $A_i = A_i(Q_i) \in \{0,1\}^{a(n)}$ and then computes a verdict $\mathrm{ACC}(x, R, A_1, \ldots, A_p) \in \{0,1\}$ with the following properties:*

Completeness: *$x \in L \Rightarrow \exists A_1(\cdot), \ldots, A_p(\cdot)$ such that $\mathbf{E}_R[\mathrm{ACC}(x, R, A_1, \ldots, A_p)] \geq c(n)$.*

Soundness: *$x \notin L \Rightarrow \forall A_1(\cdot), \ldots, A_p(\cdot), \quad \mathbf{E}_R[\mathrm{ACC}(x, R, A_1, \ldots, A_p)] < s(n)$.*

We say V is uniform if for every x and i, there exists $d_{x,i}$ s.t. for every query $Q_i \in \{0,1\}^{q(n)}$, $|\{R \in \{0,1\}^{r(n)} | Q_i(R) = Q_i\}| = d_{x,i}$. We say L is in UNIFORM-$\mathrm{MIP}_{c,s}[p, r, q, a]$ *if there exists a uniform verifier V which places L in $\mathrm{MIP}_{c,s}[p, r, q, a]$.*

We use a recent result of Raz and Safra [41] (see also [5] for an alternate proof) which provides a strong UNIFORM-MIP containment result for NP.

LEMMA A.2 (see [41, 5]). *For every $\epsilon > 0$, there exist constants p, c_1, c_2, and c_3 such that*

$$\mathrm{NP} \subseteq \text{UNIFORM-MIP}_{1,2^{-\log^{1-\epsilon} n}}[p, c_1 \log n, c_2 \log n, c_3 \log n].$$

Remark.

(1) The result shown by [41, 5] actually has smaller answer sizes, but this turns out to be irrelevant to our application below; therefore we don't mention their stronger result.

(2) The uniformity property is not mentioned explicitly in the above papers. However, it can be verified from their proofs that this property does hold for the verifier constructed there.

The following reduction is essentially from [36, 7, 2].

LEMMA A.3. *For every $\epsilon > 0$, there exists a $p = p_\epsilon$ such that* TOTAL LABEL COVER$_p$ *is NP-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$.*

Proof. We use Lemma A.2. Let L be an NP-complete language and for $\epsilon > 0$, let p, c_1, c_2, c_3 be such that $L \in \text{UNIFORM-MIP}_{1,2^{-\log^{1-\epsilon/2} n}}[p, c_1 \log n, c_2 \log n, c_3 \log n]$, and let V be the verifier that shows this containment. Given an instance $x \in \{0,1\}^n$ of L, we create an instance of TOTAL LABEL COVER$_p$ as follows: Set $Q_i(R)$ to be the query generated by V to prover Π_i on input x and random string R. For every $R, a_1, \ldots, a_p$ $\mathrm{ACC}(R, a_1, \ldots, a_p)$ is 1 if V accepts the answers $a_1, \ldots, a_p$ on random string R.

Let $\mathcal{Q} = \{0,1\}^{c_2 \log n}$ denote the set of all possible queries and let $\mathcal{R}$ denote the space of all possible random strings (i.e., $\mathcal{R} = \{0,1\}^{c_1 \log n}$). If $x \in L$, it is clear that there exists a feasible solution $A_1, \ldots, A_p$ such that for every query $q \in \mathcal{Q}$, and for

every $i \in \{1, \ldots, p\}$, it is the case that $|A_i(q)| = 1$. Thus the value of the optimum solution is at most $p \cdot |\mathcal{Q}|$.

Now we claim for a given x, if the mapped instance of TOTAL LABEL COVER has a solution of size $Kp|\mathcal{Q}|$, then there exist provers $\Pi_1, \ldots, \Pi_p$ such that V accepts with probability at least $K^{-1/p}/(p+1)^{p+1}$.

To see this let $\Pi_i(q)$ be a random element of $A_i(q)$. If $n_{i,q}$ denotes the cardinality of $A_i(q)$, then the probability that V accepts the provers response is given by

$$\frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}} \prod_i 1/n_{i,Q_i(R)}.$$

Define $\mathcal{R}_i$ to be $\{R \in \mathcal{R} | n_{i,Q_i(R)} \geq (p+1)K\}$. By Markov's inequality and the uniformity of the protocol $|\mathcal{R}_i|/|\mathcal{R}| \leq 1/(p+1)$.

Let $\mathcal{R}_0 = \mathcal{R} - \mathcal{R}_1 - \mathcal{R}_2 - \cdots - \mathcal{R}_p$. Then $|\mathcal{R}_0|/|\mathcal{R}| \geq 1/(p+1)$.

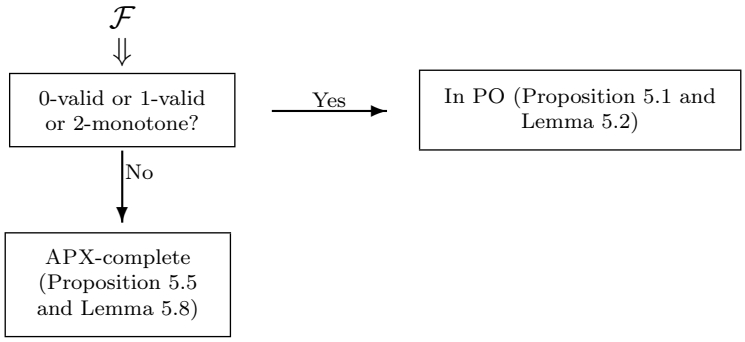
We go back to bounding the probability above:

$$\begin{aligned}
\frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}} \prod_i 1/n_{i,Q_i(R)} &\geq \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}_0} \prod_i 1/n_{i,Q_i(R)} \\
&\geq \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}_0} \prod_i 1/n_{i,Q_i(R)} \\
&\geq \frac{1}{|\mathcal{R}|} \sum_{R \in \mathcal{R}_0} (1/((p+1)K)^p) \\
&\geq K^{-1/p}/(p+1)^{p+1}.
\end{aligned}$$

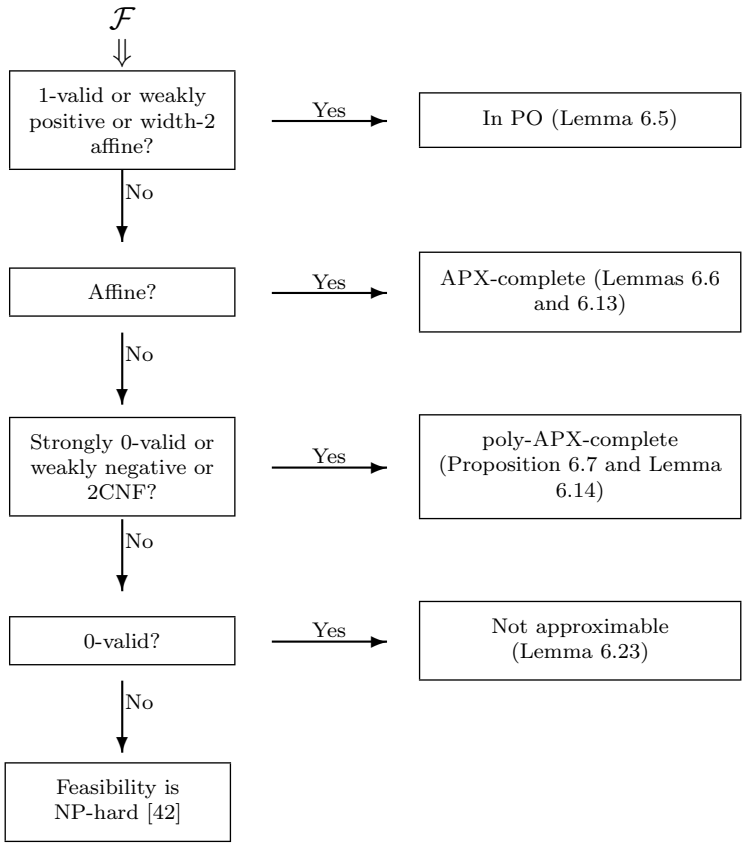
It follows that if $K = K(n)$ is less than $2^{\log^{1-\epsilon} n}$, then for sufficiently large n, $K^{-1/p}/(p+1)^{p+1}$ is greater than $2^{\log^{1-\epsilon/2} n}$. Thus a K-approximation algorithm for TOTAL LABEL COVER$_p$ can be used to decide L. Thus TOTAL LABEL COVER$_p$ is NP-hard to approximate to within a factor of $2^{\log^{1-\epsilon} n}$. □

Appendix B. Schematic representations of the classification theorems.

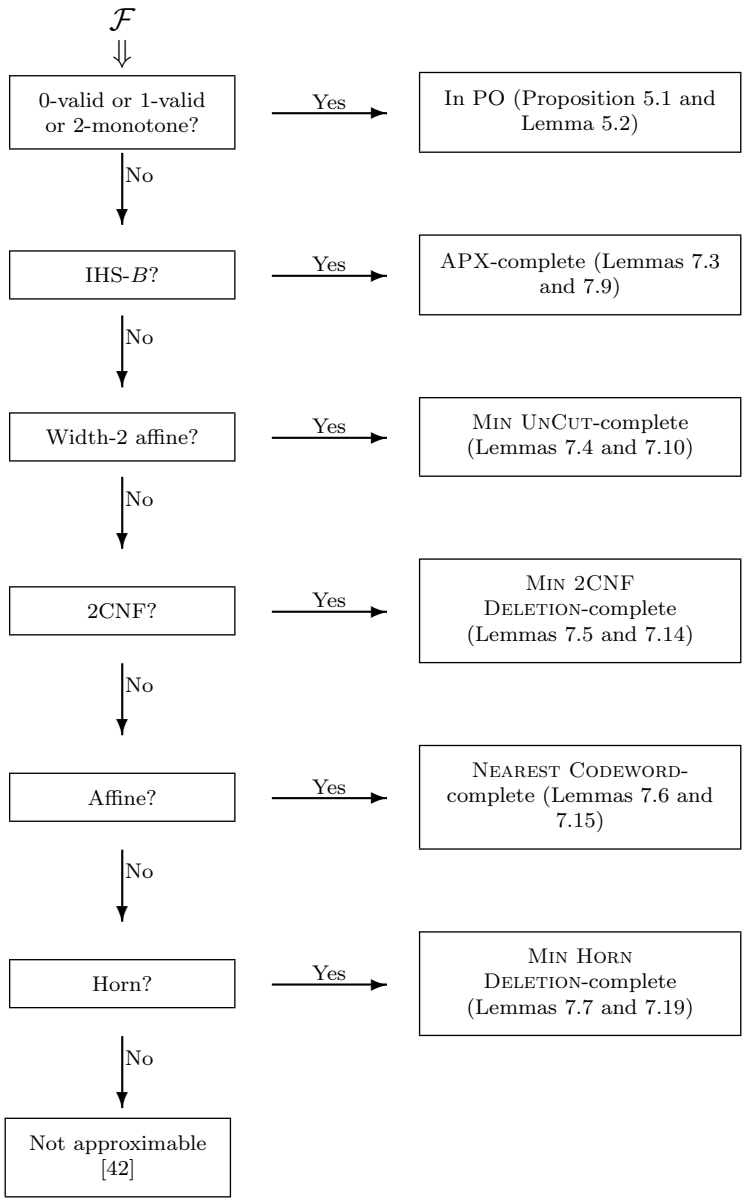
B.1. The Max CSP classification.



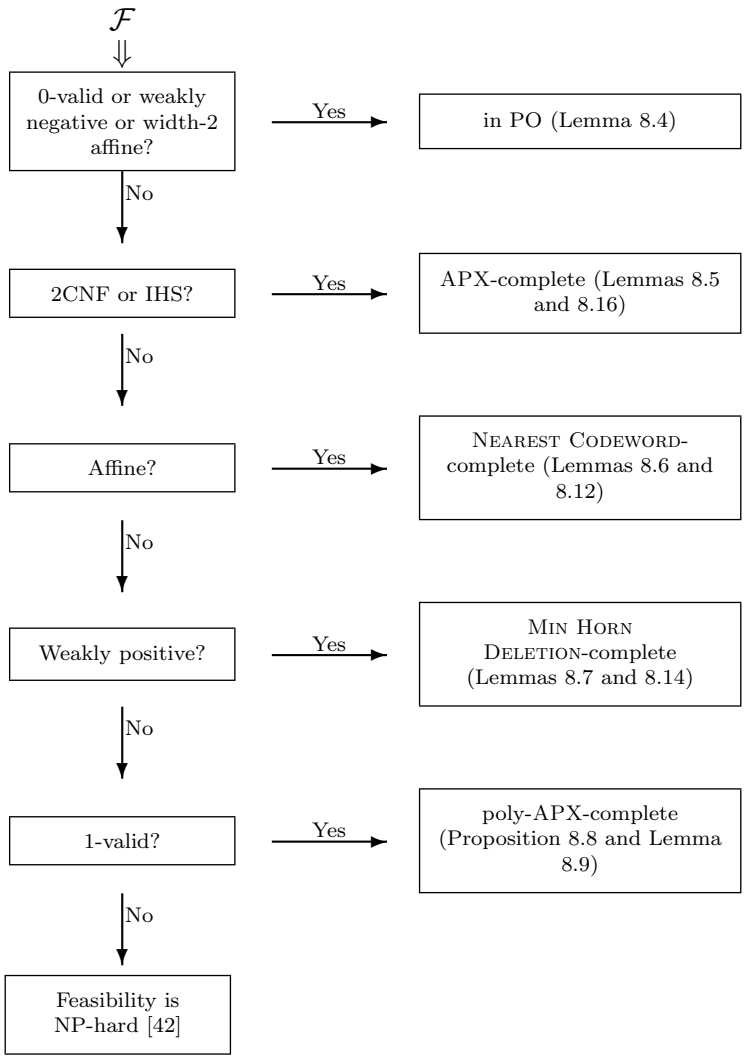
B.2. The Max Ones classification.



B.3. The MIN CSP classification.



B.4. The Min Ones classification.



Acknowledgments. We thank Mihir Bellare, Nadia Creignou, Oded Goldreich, and Jean-Pierre Seifert for useful discussions. We thank an anonymous referee for pointing out numerous errors in a previous version of this paper.

REFERENCES

[1] E. Amaldi and V. Kann, *The complexity and approximability of finding maximum feasible subsystems of linear relations*, Theoret. Comput. Sci., 147 (1995), pp. 181–210.

[2] S. Arora, L. Babai, J. Stern, and Z. Sweedyk, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, J. ACM, 45 (1998), pp. 501–555.

[4] S. Arora and S. Safra, *Probabilistic checking of proofs: A new characterization of NP*, J. ACM, 45 (1998), pp. 70–122.

[5] S. Arora and M. Sudan, *Improved low degree testing and its applications*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 485–495.

[6] T. Asano, T. Ono, and T. Hirata, *Approximation algorithms for the maximum satisfiability problem*, in SWAT '96, Rolf G. Karlsson and Andrzej Lingas, eds., Lecture Notes in Comput. Sci. 1097, Springer-Verlag, Berlin, 1996, pp. 100–111.

[7] M. Bellare, S. Goldwasser, C. Lund, and A. Russell, *Efficient probabilistically checkable proofs and applications to approximation*, in Proceedings of the 25th Annual ACM Symposium on Theory of Computing, San Diego, 1993, pp. 294–304.

[8] M. Bellare, O. Goldreich, and M. Sudan, *Free bits, PCPs, and nonapproximability—Towards tight results*, SIAM J. Comput., 27 (1998), pp. 804–915.

[9] R. Boppana and M. Halldórsson, *Approximating maximum independent sets by excluding subgraphs*, BIT, 32 (1992), pp. 180–196.

[10] D. P. Bovet and P. Crescenzi, *Introduction to the Theory of Complexity*, Prentice-Hall, New York, 1994.

[11] N. Creignou, *A dichotomy theorem for maximum generalized satisfiability problems*, J. Comput. System Sci., 51 (1995), pp. 511–522.

[12] N. Creignou and M. Hermann, *Complexity of generalized satisfiability counting problems*, Inform. and Comput., 125 (1996), pp. 1–12.

[13] P. Crescenzi, V. Kann, R. Silvestri, and L. Trevisan, *Structure in approximation classes*, SIAM J. Comput., 28 (1999), pp. 1759–1782.

[14] P. Crescenzi and A. Panconesi, *Completeness in approximation classes*, Inform. and Comput., 93 (1991), pp. 241–262.

[15] P. Crescenzi, R. Silvestri, and L. Trevisan, *To weight or not to weight: Where is the question?,* in Proceedings of the 4th IEEE Israel Symposium on Theory of Computing and Systems, Jerusalem, 1996, pp. 68–77.

[16] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multicuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[17] T. Feder and M. Y. Vardi, *The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory*, SIAM J. Comput., 28 (1998), pp. 57–104.

[18] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy, *Interactive proofs and the hardness of approximating cliques*, J. ACM, 43 (1996), pp. 268–292.

[19] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[20] N. Garg, V. V. Vazirani, and M. Yannakakis, *Approximate max-flow min-(multi)cut theorems and their applications*, SIAM J. Comput., 25 (1996), pp. 235–251.

[21] M. X. Goemans and D. P. Williamson, *New $\frac{3}{4}$-approximation algorithms for the maximum satisfiability problem*, SIAM J. Discrete Math., 7 (1994), pp. 656–666.

[22] M. Goemans and D. Williamson, *Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming*, J. ACM, 42 (1995), pp. 1115–1145.

[23] J. Håstad, *Clique is hard to approximate within $n^{1-\epsilon}$*, in Proceedinsg of the 37th Annual Symposium on Foundations of Computer Science, Burlington, VT, IEEE, 1996, pp. 627–636.

[24] J. Håstad, *Some optimal inapproximability results*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 1–10.

[25] D. S. Hochbaum, N. Megiddo, J. Naor, and A. Tamir, *Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality*, Math. Programming, 62 (1993), pp. 69–83.

[26] H. B. Hunt III, M. V. Marathe, and R. E. Stearns, *Generalized CNF satisfiability problems and non-efficient approximability*, in Proceedings of the Ninth Annual Structure in Complexity Theory Conference, Amsterdam, The Netherlands, IEEE Computer Society Press, Los Alamitos, CA, 1994, pp. 356–366.

[27] H. Karloff and U. Zwick, *A 7/8-approximation algorithm for MAX 3SAT?*, in Proceedings of the 38th Annual Symposium on Foundations of Computer Science, Miami Beach, FL, IEEE, 1997, pp. 406–415.

[28] S. Khanna and R. Motwani, *Towards a syntactic characterization of PTAS*, in Proceedings of the 28th Annual ACM Symposium on the Theory of Computing, Philadelphia, 1996, pp. 329–337.

[29] S. Khanna, R. Motwani, M. Sudan, and U. Vazirani, *On syntactic versus computational views of approximability*, SIAM J. Comput., 28 (1998), pp. 164–191.

[30] S. Khanna, M. Sudan, and L. Trevisan, *Constraint satisfaction: The approximability of*

minimization problems, in Proceedings of the 12th Annual IEEE Conference on Computational Complexity, Ulm, Germany, IEEE Computer Society Press, Los Alamitos, CA, 1997, pp. 282–296.

[31] S. Khanna, M. Sudan, and D. P. Williamson, *A complete classification of the approximability of maximization problems derived from Boolean constraint satisfaction*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 11–20.

[32] P. Klein, A. Agarwal, R. Ravi, and S. Rao, *Approximation through multicommodity flow*, in Proceedings of the 31st Annual Symposium on Foundations of Computer Science, Vol. II, St. Louis, MO, IEEE, 1990, pp. 726–737.

[33] P. N. Klein, S. A. Plotkin, S. Rao, and É. Tardos, *Approximation algorithms for Steiner and directed multicuts*, J. Algorithms, 22 (1997), pp. 241–269.

[34] P. G. Kolaitis and M. N. Thakur, *Approximation properties of NP minimization classes*, J. Comput. System Sci., 50 (1995), pp. 391–411.

[35] R. Ladner, *On the structure of polynomial time reducibility*, J. ACM, 22 (1975), pp. 155–171.

[36] C. Lund and M. Yannakakis, *On the hardness of approximating minimization problems*, J. ACM, 41 (1994), pp. 960–981.

[37] C. Lund and M. Yannakakis, *The approximation of maximum subgraph problems*, in Automata, Languages and Programming, 20th International Colloquium, Svante Carlsson, Andrzej Lingas, and Rolf G. Karlsson, eds., Lecture Notes in Comput. Sci. 700, Springer-Verlag, Lund, Sweden, 1993, pp. 40–51.

[38] R. Panigrahy and S. Vishwanathan, *An O(log* n) approximation algorithm for the asymmetric p-center problem*, J. Algorithms, 27 (1998), pp. 259–268.

[39] C. Papadimitriou and M. Yannakakis, *Optimization, approximation and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.

[40] E. Petrank, *The hardness of approximation: Gap location*, Comput. Complexity, 4 (1994), pp. 133–157.

[41] R. Raz and S. Safra, *A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 475–484.

[42] T. Schaefer, *The complexity of satisfiability problems*, in Proceedings of the 10th Annual ACM Symposium on Theory of Computing, San Diego, 1978, pp. 216–226.

[43] L. Trevisan, G. B. Sorkin, M. Sudan, and D. P. Williamson, *Gadgets, approximation, and linear programming*, SIAM J. Comput., 29 (2000), pp. 2074–2097.

[44] M. Yannakakis, *On the approximation of maximum satisfiability*, J. Algorithms, 17 (1994), pp. 475–502.

[45] D. Zuckerman, *On unapproximable versions of NP-complete problems*, SIAM J. Comput., 25 (1996), pp. 1293–1304.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1921–1941

DUAL-ISSUE SCHEDULING FOR BINARY TREES WITH SPILLS AND PIPELINED LOADS*

WALEED M. MELEIS†

Abstract. We describe an algorithm that finds a minimum cost schedule, including spill code, for a register-constrained machine that can issue up to one arithmetic operation and one memory access operation at a time, under the restrictions that the dependence graph is a full binary tree, all arithmetic and store operations have unit latency, and all load operations have a latency of 1 or all load operations have a latency of 2. This problem is a generalization of two problems whose efficient solutions are well understood: optimal dual-issue scheduling without spills for binary expression trees, solved by Bernstein, Jaffe, and Rodeh [*SIAM J. Comput.*, 18 (1989), pp. 1098–1127], and optimal single-issue scheduling with spill code and delayed loads, solved by Kurlander, Proebsting, and Fischer [*ACM Transactions on Programming Languages and Systems*, 17 (1995), pp. 740–776], both assuming a fixed number of registers. We show that the algorithm's complexity is $O(nk)$ where n is the number of operations to be scheduled and k is the number of spills in the schedule. The cost of a "contiguous" schedule (i.e., its length) is shown to be $\rho + 2k + g + |A|$, where ρ is the number of registers used, $|A|$ is the number of arithmetic operations, k is the number of spills, and g is the number of empty slots in the associated single processor schedule. Therefore all contiguous schedules formed from optimal single processor schedules have minimum cost.

Key words. scheduling algorithms, code generations, parallel functional units, registers

AMS subject classifications. 68Q20, 68Q22, 68N20, 68R05, 90B35

PII. S009753979834610X

1. Introduction.

In modern processors, memory bandwidth is a major performance bottleneck due to increased processor issue widths and clock speeds relative to the limited number of ports and access latency to memory. Reducing memory traffic by improving data reuse at all levels of the memory hierarchy is therefore critical to improve system performance. In addition, the latency of pipelined memory operations is often larger than that of most arithmetic operations. One source of memory traffic is spilling, which moves data values from registers to memory and back. This data movement is necessary because the number of registers that instructions can access is limited. Reducing this traffic can be achieved through better register allocation and register sensitive scheduling and through the judicious use of spill code to minimize performance degradation. Processors that can issue more than one instruction per cycle can hide the latency of spill operations and other memory operations by overlapping them with instructions that do useful work.

Many scheduling problems for related processor architectures can be solved efficiently. Minimizing the number of registers needed to schedule operations without spill code on a single-issue machine when the dependence graph is a tree was first solved by Nakata [30] and Redziejowski [34]. These algorithms use a postorder evaluation of the expression tree and execute in time proportional to the number of operations to be scheduled. Sethi and Ullman [36] extended this result to minimize the amount of spill code needed to evaluate an expression tree. This algorithm also uses a postorder traversal of the expression tree and executes in linear time. Aho and John-

*Received by the editors October 26, 1998; accepted for publication (in revised form) September 25, 2000; published electronically March 20, 2001. A preliminary version of this paper appeared in the Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms.
http://www.siam.org/journals/sicomp/30-6/34610.html
†Department of Electrical and Computer Engineering, Northeastern University, Boston, MA 02115 (meleis@ece.neu.edu).

son [1] describe a dynamic programming solution to this problem that applies to a wider range of architectures. While the preceding results only apply to operations with unit latencies, Kurlander, Proebsting, and Fischer [28] describe an efficient algorithm for optimally scheduling expressions trees, with spills, for a delayed load architecture. This architecture assumes that all load operations have a latency of two cycles.

When the dependence graph is a directed acyclic graph (DAG), the related scheduling problems are usually NP-complete. In this case, more than one operation can depend on a particular operation. Minimizing the number of registers needed to schedule operations without spill code on a single-issue machine when the dependence graph is a DAG was shown to be NP-complete by Sethi [35]. Bruno and Sethi [11] show that when the dependence graph is a DAG, minimizing spill code is NP-complete even when only one register is available. This result assumes an accumulator-based architecture which is different than the load/store architecture we consider. The easier problem of inserting the minimum amount of spill code into a fixed schedule is also NP-complete when the underlying dependence graph is a DAG [13, 19]. Scheduling operations with arbitrary latencies on multiple identical machines was shown by Ullman to be NP-complete when the dependence graph is a DAG [37], by Bernstein, Rodeh, and Gertner when the dependence graph is a tree [5], and by Palem and Simons when the dependence graph is a collection of chains [31].

However, a number of related problems with unit-time operations can be solved efficiently in the absence of register constraints. If all latencies equal one, polynomial algorithms are described by Coffman and Graham that schedule operations on two processors when the dependence graph is a DAG [15] and by Hu for arbitrarily many processors when the dependence graph is a tree [27]. If all latencies are equal and the dependence graph is a tree, Bruno, Jones, and So give an efficient algorithm to schedule operations on multiple processors [10]. Finally, if all latencies equal 1 or 2, Bernstein and Gertner describe an efficient algorithm to schedule a single processor with arbitrary precedence [3].

Heuristics for the spill insertion problem based on dynamic programming and pruning rules are described by Horwitz et al. [25] and Hsu, Fischer, and Goodman [26]. More practical heuristics for register allocation and spill insertion are described and evaluated by Chaitin et al. [17, 18] and Chow and Hennessy [14]. More recently, improved allocation algorithms were evaluated by Callahan and Koblenz [12], Pinter [32], Briggs et al. [8, 9], Proebsting and Fischer [33], George and Appel [20], Lueh and Gross [29], and Bergner et al. [2]. These algorithms attempt to minimize the number of spills within a basic block or over an entire program, but without explicitly handling pipelines or instruction-level parallelism. The interdependency of code scheduling and register allocation in practical applications was described in [7] and [22]. These papers address the problem of efficient code generation for basic blocks on single-issue RISC processors and document the tradeoffs that exist between good data reuse in the register file and the minimization of pipeline interlocks. Other scheduling algorithms for pipelined processors were described by Hennessy and Gross [23, 24] and by Gibbons and Muchnick [21].

The algorithms described in this paper are the first to simultaneously optimize a schedule in the presence of multiple processors, and either spill code or instruction pipelines. (In fact, we solve the scheduling problem in the presence of both spills and instruction pipelines.) These results therefore increase the space of scheduling problems that can be solved efficiently along these important dimensions. Bernstein, Jaffe, and Rodeh [4] first described an efficient algorithm to schedule a dual-issue

processor with a limit on the number of available registers. They show that an optimal dual-issue schedule can be derived from an optimal single-issue schedule. However, this result does not apply when spill code must be inserted and the authors state that "... more work concerning store operations is needed."

In this paper we describe an efficient optimal scheduling algorithm that allocates registers and inserts spill code for a dual-issue processor with pipelined memory access operations. We show that an optimal single-issue schedule can be transformed into an optimal dual-issue schedule without increasing the number of spill operations. We then show that this transformation can be further generalized to apply to processors with pipelined memory operations, and we completely describe the structure of an optimal solution to this general problem. The dual-issue machine assumed in this paper is a simplified representation of a range of pipelined RISC microprocessors and supercomputers that have a limited number of registers. By solving the problem of scheduling with spills for a dual-issue processor with pipelined memory operations, we have solved the following two previously open subproblems: optimal scheduling for a dual-issue processor with spills and optimal scheduling for a dual-issue processor without spills with pipelined memory operations. Our main result represents a generalization of optimal algorithms described by Sethi and Ullman [36] and Bernstein, Jaffe, and Rodeh [4]. This result also represents an extension of the algorithm described by Kurlander, Proebsting, and Fischer [28]; however, their machine model also includes unary operations and literals which our algorithm does not handle. We summarize the main results of this paper below.

1. A sequential schedule with arbitrary load latencies and spills can be transformed into a dual-issue schedule with the same number of spills and idle cycles, and visa versa.

2. A dual-issue schedule with arbitrary load latencies and spills can be transformed into a contiguous dual-issue schedule by moving load operations earlier in the schedule and store operations later, without increasing the schedule's length or the number of spill operations.

3. We give an efficient algorithm that solves the dual-issue scheduling problem with spills for load latencies = 1 or load latencies = 2.

Our paper is organized as follows. In section 2 we describe our machine and computation model and define the sequential and parallel scheduling problems. In section 3 we describe some basic properties of parallel schedules. In sections 4 and 5 we show that we only need to consider contiguous schedules. In section 6 we describe the structure and cost of a contiguous schedule, and in section 7 we describe an algorithm that solves the parallel scheduling problem.

2. The machine/computation model. The notation used by Bernstein, Jaffe, and Rodeh in [4] formalizes the sequential and parallel scheduling problems without spills. We extend this notation to allow the use of spill code and nonunit latencies for loads. We consider a machine model that has access to R registers: $r_1\ r_2, \ldots, r_R$ and an arbitrarily large memory space. The machine has three types of operations: load operations which copy a data value from memory into a register, store operations which copy a data value from a register into memory, and binary arithmetic operations which read two data values from registers and write a new data value to a register. Arithmetic and store operations have unit latencies, and each load operation has a latency that is some positive integer. In the definitions below, we assume that an operation's latency gives the minimum number of cycles that must elapse before a dependent operation is scheduled. We also assume that all functional units are fully

pipelined, which allows an operation to be issued to each functional unit in each cycle. We let NOP indicate that no operation is scheduled at some time.

The dependence graph F is a rooted binary tree that contains leaf nodes that represent load operations and internal nodes that represent arithmetic operations. We assume that every internal node has exactly two predecessors. An edge between two nodes represents a data dependence between the two operations, and an operation must be scheduled after its predecessors. The set of *arithmetic operations* in F is denoted $A(F) = \{A_1, A_2, \ldots, A_{|A(F)|}\}$, where A_i may denote either the operation or its *arithmetic result* (or simply, *value*). An arithmetic operation is said to *use* a value that it directly depends on. The set of *load operations* is $L(F) = \{L_1, L_2, \ldots, L_{|L(F)|}\}$, where L_i represents a load of A_i if $1 \leq i \leq |A(F)|$, and L_i represents a leaf node of F if $|A(F)| < i \leq |L(F)|$. The latency of load operation L_i is denoted $latency(L_i)$. The set of *store operations* is denoted $S(F) = \{S_1, S_2, \ldots, S_{|A(F)|}\}$, where S_i represents a store of A_i. A load operation L_i or a store operation S_i is referred to as *spill code* if $1 \leq i \leq |A(F)|$. Note that if value A_i is spilled in an evaluation of F, S_i and later L_i will appear in the evaluation somewhere between A_i and the use of A_i; if A_i is not spilled, S_i and L_i will not appear at all.

For a given node v of F, F_v denotes the subtree of F with v as its root. If u is a predecessor of v in F, we say u is *used* by v.

A *sequential evaluation* P of a computation F is an ordered list of operations (including spill code and NOPs): $P = P_1, P_2, \ldots, P_n$, where P_i is executed at time i, and the following three conditions are satisfied:

1. Every arithmetic operation and load leaf is executed, i.e., for every node v, there exists a unique j such that $P_j = v$.

2. No dependences of F are violated, i.e., if P_i, P_j are nodes in F, and P_j is an arithmetic operation, then $P_j \in F_{P_i}$ implies $j \leq i - 1$. If P_j is a load operation, then $P_j \in F_{Pi}$ implies $j \leq i - latency(P_j)$. This implies that the latency of each arithmetic operation is one, and the latency of each load operation P_j is $latency(P_j)$.

3. If the value A_v corresponding to a nonleaf node P_i is used by P_j $(j > i)$, then A_v may be stored at time k_1 and loaded at time k_2, provided that $i < k_1 < k_2 \leq j - latency(P_{k_2})$. In this case, $P_{k_1} = S_v$ and $P_{k_2} = L_v$, and S_v and L_v are spill code.

Conditions 2 and 3 are collectively referred to as the *precedence constraints* of a sequential evaluation. An operation $P_i = NOP$ is called an *empty slot* of a sequential evaluation. The cost of P, $c(P)$, equals the number of operations in P, i.e., we define an evaluation's cost to be equal to its length.

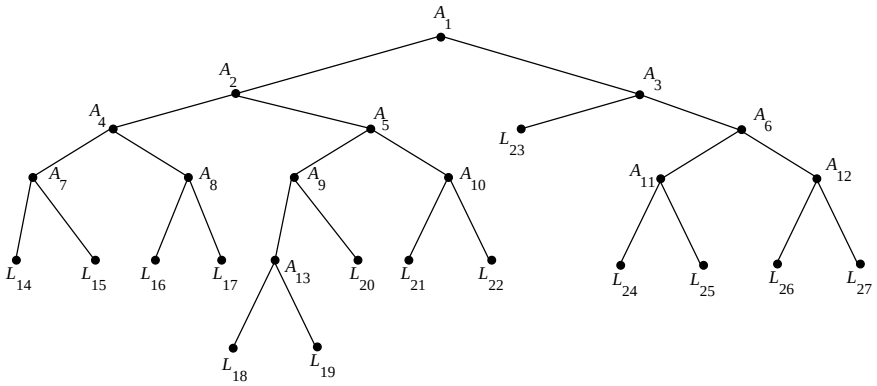
We now extend the definitions in [4] to handle dual-issue processors. A *parallel evaluation* PQ of a computation F is an ordering of operation pairs: $PQ = PQ_1, PQ_2, \ldots, PQ_n$ where $PQ_i = (p_i, q_i)$ such that $p_i \in L(F) \cup S(F) \cup \{NOP\}$ and $q_i \in A(F) \cup \{NOP\}$. NOPs are used to indicate that no operation is scheduled in either or both operation pairs at some time. A parallel evaluation PQ must satisfy the following four conditions:

1. For every nonleaf node v, there exists a unique PQ_j such that $q_j = v$.

2. For every leaf node v, there exists a unique PQ_j such that $p_j = v$.

3. For every PQ_i, PQ_j, if $q_i \neq NOP$, then $p_j \in F_{q_i}$ implies $j \leq i - latency(q_i)$. Furthermore, $q_j \in F_{q_i}$ implies $j \leq i - 1$. Once again, the latency of each arithmetic operation is one, and the latency of load operation q_i is $latency(q_i)$.

4. For every PQ_i, PQ_j, if $q_i = A_v$ and q_j uses A_v $(j > i)$, then A_v may be stored at time k_1 and loaded at time k_2, provided that $i < k_1 < k_2 \leq j - latency(k_2)$. In this case, $PQ_{k_1} = (S_v, q_{k_1})$ and $PQ_{k_2} = (L_v, q_{k_2})$ are spill code.



(a) *A computation F.*

Slot $i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$P_i =$	L_{18}	L_{19}	–	A_{13}	L_{20}	–	A_9	L_{21}	L_{22}	–	A_{10}	A_5	S_5	L_{14}	L_{15}	–	A_7	L_{16}	L_{17}	–
$\rho_i =$	1	2	2	1	2	2	1	2	3	3	2	1	0	1	2	2	1	2	3	3

Slot $i =$	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
$P_i =$	A_8	A_4	L_5	–	A_2	S_2	L_{24}	L_{25}	–	A_{11}	L_{26}	L_{27}	–	A_{12}	A_6	L_{23}	–	A_3	L_2	–	A_1
$\rho_i =$	2	1	2	2	1	0	1	2	2	1	2	3	3	2	1	2	2	1	2	2	1

(b) *A serial evaluation of F.*

Slot $i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$p_i =$	L_{14}	L_{15}	–	L_{16}	L_{17}	–	–	L_{18}	L_{19}	–	L_{20}	–	L_{21}	S_4	L_{22}	–	–	–	L_4	–
$q_i =$	–	–	–	A_7	–	–	A_8	A_4	–	–	A_{13}	–	A_9	–	–	–	A_{10}	A_5	–	–
$\rho_i =$	1	2	2	2	3	3	2	2	3	3	3	3	3	2	3	3	2	1	2	2

Slot $i =$	21	22	23	24	25	26	27	28	29	30	31	32	33
$p_i =$	L_{24}	L_{25}	–	L_{26}	S_{11}	L_{27}	–	L_{11}	–	L_{23}	–	–	–
$q_i =$	A_2	–	–	A_{11}	–	–	–	A_{12}	–	A_6	–	A_3	A_1
$\rho_i =$	2	3	3	3	2	3	3	3	3	3	3	2	1

(c) *A parallel evaluation of F.*

FIG. 1. *Examples of sequential and parallel evaluations for latency $= 2$ and $R = 3$.*

Conditions 3 and 4 are collectively referred to as the precedence constraints of a parallel evaluation. A pair (p_i, NOP) is called an *empty arithmetic slot* if p_i is not a NOP, and (NOP, q_i) is called an *empty load/store slot* if q_i is not a NOP. A pair (NOP, NOP) is called an *empty slot* of a parallel evaluation. Note that by definition the sets of empty arithmetic slots, empty load/store slots, and empty slots in an evaluation are disjoint. The cost of PQ, $c(PQ)$, equals the number of operation pairs in PQ, and note that we again define an evaluation's cost to be equal to its length.

Sequential and parallel evaluations defined in this way ensure that every operation is executed, the operations' latencies are not violated, and spills (if any) are properly placed in the evaluation. The limitations imposed by a register file of finite size are described below. The concatenation $P|P'$ (or simply PP') of two sequential evaluations represents the sequential evaluation formed by the operations in P followed by the operations in P'.

For a sequential or parallel evaluation E, $inreg_i$ is the set of all values that are in a register at time i, namely the set of all values v such that v is defined or loaded at time j, where $j \leq i$, and is neither used nor stored at any time k, where $j < k \leq i$.

A nonleaf value v is said to be *spilled* at time i if it is stored at time j, where $j \leq i$, and is loaded at time k, where $k > i$. Thus if v is spilled or used at time i, then v is not in a register at time i.

Each load operation adds one element to $inreg$, each store removes one, and each arithmetic operation adds one and removes two. Therefore, $|inreg_i|$ = the total number of loads, minus stores, minus arithmetic operations at time i or earlier. The number of values in registers at time i of an evaluation E, $|inreg_i|$, is denoted $\rho_i(E)$, and $\rho(E)$ denotes the maximum over all i of $\rho_i(E)$. That is, $\rho_i(E) = |inreg_i|$ and $\rho(E) = \max_i \rho_i(E)$. When all load latencies are equal, we let $latency = latency(L_i)$. An example is shown in Figure 1 in which $latency = 2$ and $R = 3$.

Given a computation F, the number of available registers R, and the latencies of the load operations, the sequential scheduling problem is to construct a sequential evaluation P such that $c(P)$ is minimized subject to $\rho(P) \leq R$. The parallel scheduling problem is to construct a parallel evaluation PQ such that $c(PQ)$ is minimized subject to $\rho(PQ) \leq R$. In each case, the evaluation constructed is said to *use R registers*. In this paper we describe algorithms that solve the parallel scheduling problem for $latency = 1$ and $latency = 2$.

3. Minimizing spill operations plus empty slots in a parallel evaluation. Given a tree F, the number of available registers R, and the latencies of the load operations, a solution to the parallel scheduling problem is found by using the sequential scheduling algorithm to construct a sequential evaluation that minimizes the number of spill operations plus empty slots, transforming this evaluation into a parallel evaluation, and then transforming this evaluation into a "contiguous" form. At each stage of the transformation the number of spill operations plus empty slots remains unchanged and the cost of the evaluation does not increase. In this section we show that for arbitrary load latencies, no parallel evaluation of F can have fewer spill operations plus empty slots than an optimal sequential evaluation of F, and that a minimum cost sequential evaluation can be transformed into a parallel evaluation that minimizes the number of spill operations plus empty slots. In the following sections, we show that every parallel evaluation PQ of F in contiguous form that contains k spilled values and g empty slots has a cost of $\rho(PQ) + 2k + g + |A(F)|$, and, for $latency = 1$ or $latency = 2$, that no parallel evaluation can cost less.

3.1. Minimizing spills in a sequential evaluation. An efficient algorithm that solves the sequential scheduling problem with $latency = 1$ was described by Aho, Sethi, and Ullman [1] and Sethi and Ullman [36]. In addition to the three types of operations described above, these algorithms include another binary arithmetic operator that reads one operand from a register and one operand from memory and writes its result to a register, causing no net change in $inreg_i$. Bose [6] describes an adaptation of the algorithm in [1] that solves the sequential scheduling problem without this fourth instruction class.

For a given vertex v, the algorithm first computes the minimum number of registers needed to evaluate the subtree rooted at v without spill code. The scheduling routine uses these quantities to optimally schedule the entire tree. Spill code is inserted when neither subtree can be scheduled using R or fewer registers. The operations that compute a particular subtree are executed consecutively. That is, the computation of a subtree is never interrupted by the computation of another subtree. The subtree with the higher register requirement at a particular vertex is always executed first. As argued in [6], this sequential scheduling algorithm finds a minimum cost evaluation of F that uses no more than R registers assuming $latency = 1$. Furthermore, it can be

shown that if the cheapest evaluation contains no spills, the algorithm minimizes the maximum number of values in registers. The proof of this, which we omit, is similar to the one given in [36].

An efficient algorithm to solve the sequential scheduling problem with $latency = 2$ was described by Kurlander, Proebsting, and Fischer [28]. They show that all nontrivial trees can be scheduled without empty slots or spill code if the number of available registers is at least one more than the minimum number of registers needed to evaluate the tree without spills. In this case the sequential scheduling algorithm for $latency = 1$ is first used to create an evaluation, and the ordering of the load operations and the ordering of the arithmetic operations becomes an invariant. The new evaluation then consists of a series of load operations, followed by an alternating series of arithmetic and load operations, and ending with a series of arithmetic operations. This algorithm therefore solves the sequential scheduling problem with $latency = 2$ for sufficiently large values of R.

For smaller values of R, spill code may need to be inserted in the evaluation. For each vertex v, the algorithm computes the *register pressure* caused by v by applying the unit load latency sequential scheduling algorithm to the subtree rooted at v. The register pressure caused by v is equal to the number of times during this evaluation that the number of live values equals the maximum number of live values. A vertex v is then spilled if the minimum number of registers needed to evaluate the subtree rooted at v equals R and v's register pressure is greater than two. If the minimum number of registers needed to evaluate v is greater than R, then a predecessor of v with the largest register pressure is spilled.

The cost of a sequential evaluation equals the number of nodes in F plus the number of spill operations and empty slots. Therefore an optimal sequential evaluation must contain the fewest spill operations plus empty slots among all sequential evaluations. Furthermore, if k values are spilled, there are $2k$ spill operations, i.e., k store/load pairs.

3.2. A lower bound on spill operations plus empty slots in a parallel evaluation. A simple transformation allows us to transform a k-spill, g-empty slot parallel evaluation with arbitrary load latencies and that uses R registers into a k-spill, g-empty slot sequential evaluation that uses R registers. Recall that for a parallel evaluation PQ and a time i, $PQ_i = (p_i, q_i)$.

Serial transformation. Given a k-spill parallel evaluation PQ with arbitrary load latencies that uses R registers, we construct a serial evaluation P as follows. For each time i, if $PQ_i = (L, A)$ for some load L and arithmetic operation A, let $J_i = A, L$. Similarly, if $PQ_i = (L, NOP)$, (S, NOP), or (NOP, A), let $J_i = L$, S, or A, respectively. If $PQ_i = (S, A)$, let $J_i = A, S$. Finally if $PQ_i = (NOP, NOP)$, let $J_i = NOP$. Then let $P' = J_1 J_2 \dots J_{|PQ|}$. In [4] a similar transformation is applied to parallel evaluations that do not contain spills and with $latency = 1$. We first show that when P is constructed in this manner it represents a sequential evaluation. Then we show that $\rho(P) \leq \rho(PQ)$ (Lemma 3.1).

If two operations v and w are scheduled $d \geq 1$ cycles apart in parallel evaluation PQ, these operations are scheduled d or more cycles apart after PQ is transformed into P. Therefore these operations cannot violate the precedence constraints in the serial evaluation P. If operations v and w are scheduled at the same time in PQ then by definition of a parallel evaluation they are not dependent on one another. Therefore these operations cannot violate the precedence constraints in P, and it follows that P represents a sequential evaluation.

LEMMA 3.1. *Given a k-spill parallel evaluation PQ of a computation F with arbitrary load latencies and R available registers, and given the serial evaluation P formed by applying the serial transformation to PQ, then $\rho(P) \leq \rho(PQ)$.*

Proof. Consider some arbitrary time j of the evaluation P. Suppose first that P_j contains an arithmetic operation, i.e., $P_j = q_i$ for some time i of PQ. By definition of the serial transformation, all operations scheduled earlier than q_i in P must have been scheduled at time $i-1$ or earlier in PQ. Therefore, the number of values in registers at time $j-1$ of P and at time $i-1$ of PQ is the same, i.e., $\rho_{j-1}(P) = \rho_{i-1}(PQ)$. Since q_i is an arithmetic operation, $\rho_j(P) = \rho_{j-1}(P) - 1 \leq \rho_{j-1}(P)$, which implies that $\rho_j(P) \leq \rho_{i-1}(PQ)$.

Now suppose that P_j contains a load/store operation, i.e., $P_j = p_i$ for some time i of PQ. By definition of the serial transformation, all operations scheduled at time j or earlier in P must have been scheduled at time i or earlier in PQ. Therefore, $\rho_j(P) \leq \rho_i(PQ)$. Finally if P_j contains a NOP, then $\rho_j(P) \leq \rho_{j-1}(P)$. □

Since P is a sequential evaluation of F that contains the same non-NOP operations as PQ, evaluations P and PQ contain the same number of spills. Furthermore, by construction both evaluations contain the same number of empty slots. We can now state the following lemmas whose proofs follow from the discussion above.

LEMMA 3.2. *Consider a computation F with arbitrary load latencies and R available registers. If an optimal sequential evaluation P of F that uses R registers contains $2k$ spill operations and g empty slots, then the number of spill operations plus empty slots in any parallel evaluation of F that uses R registers is not less than $2k+g$.*

LEMMA 3.3. *Consider a computation F with arbitrary load latencies and R available registers, and let minreg be the minimum number of registers needed to evaluate F without spills in a serial evaluation. Then all parallel evaluations of F that use fewer than minreg registers have at least one spill.*

3.3. Creating a minimum-spill operation plus empty slot parallel evaluation. An optimal serial evaluation of F thus directly yields a lower bound $2k+g$ on the number of spill operations plus empty slots in any parallel evaluation of F. We can achieve this lower bound through a straightforward transformation of an optimal sequential evaluation.

Parallel transformation. Given a k-spill, g-empty slot sequential evaluation $P = P_1, \ldots, P_n$ with arbitrary load latencies, then for each time i,

(i) if $P_i \in A(F)$, let $PQ_i = (NOP, P_i)$, and
(ii) if $P_i \in L(F) \cup S(F)$, let $PQ_i = (P_i, NOP)$,
(iii) if $P_i = NOP$, let $PQ_i = (NOP, NOP)$.

No precedence is violated by this transformation since the operations are executed in the same order, and all pairs of operations are the same number of cycles apart in both evaluations. Therefore PQ represents a k-spill, g-empty slot parallel evaluation of F that uses R registers. Note, however, that PQ is degenerate since there is a NOP scheduled in PQ for every time i; thus PQ is unlikely to have the minimum cost. The result of applying the parallel transformation to the sequential evaluation in Figure 1(b) is shown in Figure 2.

4. Load-contiguous parallel evaluations. We have shown that a parallel evaluation with the minimum number of spill operations plus empty slots can be constructed by applying the parallel transformation to an optimal sequential evaluation. We now describe the process of transforming this parallel evaluation into a contiguous form. We define a load-contiguous parallel evaluation and show that

Slot $i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$p_i =$	L_{18}	L_{19}	–	–	L_{20}	–	–	L_{21}	L_{22}	–	–	–	S_5	L_{14}	L_{15}	–	–	L_{16}	L_{17}	–
$q_i =$	–	–	–	A_{13}	–	–	A_9	–	–	–	A_{10}	A_5	–	–	–	–	A_7	–	–	–
$\rho_i =$	1	2	2	1	2	2	1	2	3	3	2	1	0	1	2	2	1	2	3	3

Slot $i =$	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
$p_i =$	–	–	L_5	–	–	S_2	L_{24}	L_{25}	–	–	L_{26}	L_{27}	–	–	–	L_{23}	–	–	L_2	–	–
$q_i =$	A_8	A_4	–	–	A_2	–	–	–	–	A_{11}	–	–	–	A_{12}	A_6	–	–	A_3	–	–	A_1
$\rho_i =$	2	1	2	2	1	0	1	2	2	1	2	3	3	2	1	2	2	1	2	2	1

FIG. 2. *The result of transforming the sequential evaluation in Figure* 1(b) *into a parallel evaluation for latency* $= 2$ *and* $R = 3$.

every parallel evaluation can be made load-contiguous without increasing its cost or the maximum number of values in registers. Furthermore, for load latencies ≤ 2 we show that the number of spill operations plus empty slots also does not increase. We then show in section 5 that a load-contiguous evaluation can be made contiguous, i.e., both load-contiguous and store-contiguous, without increasing its cost, the maximum number of values in registers, or the number of spill operations plus empty slots. Finally in sections 6 and 7 we show that for *latency* $= 1$ or *latency* $= 2$, a contiguous evaluation that minimizes the number of spill operations plus empty slots is necessarily optimal. That is, applying the contiguity transformations to a parallel evaluation with the minimum number of spill operations plus empty slots produces a minimum cost evaluation.

A parallel evaluation of a computation F is said to be *load-contiguous* if and only if no load operation is scheduled at a later time than an empty load/store slot, i.e., a pair (NOP, q) where q is not a NOP. Using this definition we can prove the following theorem and lemma.

THEOREM 4.1. *Given any* k*-spill parallel evaluation* PQ *of a computation* F *with arbitrary load latencies, there exists a* k'*-spill load-contiguous parallel evaluation* PQ' *of* F *such that*

(i) $c(PQ') \leq c(PQ)$,
(ii) $\rho(PQ') \leq \rho(PQ)$, *and*
(iii) $k' \leq k$.

Proof. Let PQ be a k-spill, g-empty slot parallel evaluation of a computation F that uses R registers, but is not load-contiguous. Consider the first empty load/store slot i. Let j be the first time after i where a load operation is scheduled. We create an evaluation PQ' that is the same as PQ, except one of of the following two transformations has been applied.

1. If the load scheduled at time j loads a value that is stored at time m such that $i < m < j$, both the load and store are deleted.

2. Otherwise, the load operation at time j is moved to i.

In both cases, $c(PQ') \leq c(PQ)$ and PQ' has no more spill operations than PQ. It remains to show that $\rho(PQ') \leq \rho(PQ)$, and that precedence is preserved.

Since no load/store operation is scheduled at time i, $\rho_i(PQ) = \rho_{i-1}(PQ) - 1$. From time i through $j-1$ by assumption there are no load operations, so the number of registers used as we progress forward can only stay the same or decrease. In particular, for all time s such that $i \leq s < j$, $\rho_s(PQ) \leq \rho_{i-1}(PQ) - 1$. We tentatively create a new evaluation PQ' that is identical to PQ except the load operation at time j is moved to time i. That is, $p'_i = p_j$, and $p'_j = NOP$. For all $m < i$, $\rho_m(PQ') = \rho_m(PQ)$. Similarly, for all $m \geq j$, $\rho_m(PQ') = \rho_m(PQ)$. However, $\rho_i(PQ') = \rho_i(PQ) + 1$ due to the new load operation at time i. Similarly, for all time

Slot $i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$p_i =$	L_{18}	L_{19}	–	L_{20}	–	L_{21}	L_{22}	–	L_{14}	L_{15}	S_5	–	L_{16}	L_{17}	–	L_5	L_{24}	–	L_{25}	S_2
$q_i =$	–	–	–	A_{13}	–	A_9	–	–	A_{10}	A_5	–	–	A_7	–	–	A_8	A_4	–	A_2	–
$\rho_i =$	1	2	2	2	2	2	3	3	3	3	2	2	2	3	3	3	3	3	3	2

Slot $i =$	21	22	23	24	25	26	27	28	29	30
$p_i =$	–	L_{26}	L_{27}	–	L_{23}	L_2	–	–	–	–
$q_i =$	–	A_{11}	–	–	A_{12}	A_6	–	A_3	–	A_1
$\rho_i =$	2	2	3	3	3	3	3	2	2	1

FIG. 3. *The result of applying the load-contiguous transformation to the parallel evaluation in Figure* 2 *for latency* $= 2$ *and* $R = 3$.

s such that $i \leq s < j$, $\rho_s(PQ') = \rho_s(PQ) + 1$. But since $\rho_s(PQ) \leq \rho_{i-1}(PQ) - 1$ from above, we can conclude that for $i \leq s < j$, $\rho_s(PQ') = \rho_s(PQ) + 1 \leq \rho_{i-1}(PQ)$. Therefore if the load operation at time j is moved to i, $\rho(PQ') \leq \rho(PQ) \leq R$. By a similar argument we can show that if the load at time j and a store at time m are deleted, $\rho(PQ') \leq \rho(PQ) \leq R$.

Moving a load operation earlier in the evaluation cannot violate its precedence with respect to an arithmetic operation that uses the loaded value. If the load operation scheduled at time j loads a value that is stored at time m such that $i < m < j$, both the load and store operations are deleted. Note that the number of empty slots in the evaluation will increase if the load operation at time j or the store operation at time m is in an empty arithmetic slot.

The transformation described above can be applied iteratively until the evaluation is load-contiguous. The transformation does not affect any operation earlier than i, and each time the transformation is applied either a load operation is scheduled at time i or a spill is eliminated. Therefore the iterations will eventually terminate, giving an evaluation in which the initial portion of the evaluation contains load operations, store operations and empty slots, until the load operations are exhausted. □

We now extend this result by showing that for *latency* ≤ 2, the number of spill operations plus empty slots also does not increase.

LEMMA 4.2. *Given any* k*-spill,* g*-empty slot, parallel evaluation* PQ *of a computation* F *with latency* ≤ 2*, there exists a* k'*-spill,* g'*-empty slot, load-contiguous parallel evaluation* PQ' *of* F *such that*

(i) $c(PQ') \leq c(PQ)$,
(ii) $\rho(PQ') \leq \rho(PQ)$,
(iii) $k' \leq k$, *and*
(iv) $g' = g$.

Proof. Let PQ be a k-spill, g-empty slot parallel evaluation of a computation F that uses R registers, but is not load-contiguous. Consider the first empty load/store slot i. Let j be the first time after i where a load operation is scheduled. We apply the transformation described in Theorem 4.1 to either move the load operation at time j to i or delete the load operation at time j along with the store of the same value. In each case we show that an evaluation can be created that contains g empty slots.

If PQ_j originally contained an empty arithmetic slot, moving the load operation scheduled there to time i will create a new empty slot at time j. Deleting this empty slot does not violate precedence. Deleting an empty slot cannot violate precedence between two arithmetic operations, between an arithmetic operation and a store operation, or between a store operation and a load operation, because arithmetic and store operations have a latency of 1. Since the load latency is not larger than 2,

deleting an empty slot at time j could only violate precedence if PQ_{j-1} contains a load operation and PQ_{j+1} contains an arithmetic operation dependent on that load. However, by assumption there are no load operations scheduled between time i and time j in PQ. Therefore PQ_{j-1} does not contain a load operation, and precedence is not violated if an empty slot at time j is deleted.

A similar argument applies if the load operation scheduled at time j loads a value that is stored at time m such that $i < m < j$. Deleting both the load and store operations will create new empty slots if either operation was originally scheduled in an empty arithmetic slot. Precedence would be violated by deleting these empty slots if either operation was preceded by a load operation and followed by an arithmetic operation dependent on that load. However, by assumption there cannot be a load operation scheduled between the first empty load/store slot at time i and time j. Therefore any new empty slots can be deleted without violating precedence.

Furthermore by arguments similar to those used in the proof of Theorem 4.1 we can show that $c(PQ') \leq c(PQ)$, $\rho(PQ') \leq \rho(PQ)$, $k' \leq k$, and that this transformation can be applied iteratively until the evaluation is load contiguous. □

Load-contiguous transformation for load latencies ≤ 2. Given any parallel evaluation PQ of a computation F, we create a load-contiguous parallel evaluation PQ' using the transformation described in Lemma 4.2: we apply the transformation from Theorem 4.1 and delete any slots that are empty as a result of each phase of the transformation.

The result of applying the load-contiguous transformation to the parallel evaluation in Figure 2 is shown in Figure 3. We now extend the notion of load-contiguity to stores.

5. Store-contiguous parallel evaluations. We define a store-contiguous parallel evaluation and show that every load-contiguous parallel evaluation can be made store-contiguous as well. Given R available registers, a parallel evaluation of a computation F is *store-contiguous* if and only if for any i such that a store operation is scheduled at time i, $\rho_{i+1} = R$. This implies that store operations are scheduled as late as possible. Using this definition we can prove the following theorem.

THEOREM 5.1. *Given a k-spill, g-empty slot, load-contiguous, parallel evaluation PQ of a computation F with arbitrary load latencies and with $\rho(PQ) \leq R$, there exists a k'-spill, g'-empty slot, parallel evaluation PQ' of F such that*

(i) $c(PQ') \leq c(PQ)$,
(ii) $\rho(PQ') \leq R$,
(iii) $k' \leq k$,
(iv) $g' \geq g$,
(v) $2k' + g' \leq 2k + g$,
(vi) *PQ' is load-contiguous, and*
(vii) *PQ' is store-contiguous.*

Proof. Let PQ be a load-contiguous, k-spill, g-empty slot parallel evaluation of a computation F that uses R registers, but is not store-contiguous. We show that PQ can be transformed into an evaluation that is both load-contiguous and store-contiguous without increasing the evaluation's cost, the number of spill operations, or the number of spill operations plus empty slots, and without causing the number of empty slots to decrease or the maximum number of values in registers to exceed R. We begin by showing that the last store operation in the evaluation that violates the store-contiguous property cannot immediately precede another store operation. We then show that the parallel evaluation can be transformed so that the store operation

no longer violates the store-contiguous property. Finally we show that both load-contiguity and store-contiguity can be achieved for the entire evaluation.

Consider the largest time i such that a store operation is scheduled at time i and $\rho_{i+1}(PQ) < R$. We show by contradiction that there cannot be a store operation scheduled at time $i+1$. Suppose there is a store operation scheduled at time $i+1$. Then $\rho_{i+1}(PQ) < \rho_i(PQ)$ because the number of values in registers must decrease by at least 1 between time i and time $i+1$. Furthermore, since a store operation is scheduled at time i, $\rho_i(PQ) < \rho_{i-1}(PQ)$. Thus $\rho_{i+1}(PQ) \leq \rho_{i-1}(PQ) - 2$. The number of values in registers at any time can increase by at most one in the next time period, which occurs when the next time period contains *only* a load operation. Therefore, $\rho_{i+2}(PQ) \leq \rho_{i-1}(PQ) - 1$. Since $\rho_{i-1}(PQ) \leq R$, we have $\rho_{i+2} \leq R-1$. In this case, the store scheduled at time $i+1$ contradicts the assumption that the store scheduled at time i is the last store that precedes a time period with fewer than R values in registers. Therefore there is not a store operation scheduled at time $i+1$.

We now show that since $\rho_{i+1}(PQ) < R$, we can construct a new evaluation PQ' in which the store scheduled at time i is moved one time period later. We consider seven cases depending on the operations scheduled at times i and $i+1$. The transformation from PQ to PQ' is shown in Figure 4. For each case, the upper pairs of operations represent PQ_i and PQ_{i+1}, while the lower pairs represent PQ'_i and PQ'_{i+1}. For conciseness we let X and Y each represent either an arithmetic operation or an empty slot, and we let Z represent either a load operation or an empty slot. In each case it can be seen that $c(PQ') \leq c(PQ)$, $k' \leq k$, and $g' \geq g$.

The new evaluation does not violate the precedence constraints because in each case one or more of the following transformations is performed: a load operation is moved earlier in the evaluation, a store operation is moved later in the evaluation, an arithmetic operation is moved one slot later into a previously empty slot, or an arithmetic operation is moved before a store operation. None of these transformations can violate the precedence constraints, unless a load operation is moved earlier than a store operation of the same value. In this case deleting both operations does not violate precedence. We now show that in each case $\rho(PQ') \leq R$ and $2k' + g' \leq 2k + g$.

The transformations affect only operations scheduled at time i or $i+1$, so for all $j \leq i-1$ and $j \geq i+1$, $\rho_j(PQ) = \rho_j(PQ')$. As a result, showing that $\rho_i(PQ') \leq \rho_{i-1}(PQ')$ allows us to conclude that $\rho(PQ') \leq R$.

(i) Case 1 in Figure 4 such that S and L reference the same value: Delete the load and store operations, i.e., let $PQ'_i = (NOP, X)$ and $PQ'_{i+1} = (NOP, Y)$. Since $PQ'_i = (NOP, A)$ or (NOP, NOP), $\rho_i(PQ') \leq \rho_{i-1}(PQ')$. The number of empty slots does not decrease. The quantity $2k+g$ does not increase because the number of spills decreases by one and the number of empty slots increases by no more than two.

(ii) Cases 2–6 in Figure 4 such that S and L, if present, do not reference the same value: Since $\rho_{i-1}(PQ) \leq R$ and $\rho_{i-1}(PQ) = \rho_{i-1}(PQ')$, it can be seen in each case that $\rho_i(PQ') \leq R$. Also, the number of spill operations and empty slots is unchanged.

(iii) Case 7 in Figure 4 such that S and L do not reference the same value: In this case, $\rho_{i-1}(PQ) \leq R-1$. If this were not true, then $\rho_{i+1}(PQ) = R$, which contradicts the assumption that $\rho_{i+1}(PQ) < R$. Therefore $\rho_i(PQ') \leq R$. Again, the number of spill operations and empty slots is unchanged.

In each case we have shown that if a store operation is scheduled at time i and $\rho_{i+1}(PQ) < R$, it is always possible to move the store one time period later without violating precedence, without increasing the evaluation's cost, the number of spill

Case =	1	2	3	4	5	6	7
$p =$	$S\ L$	$S\ -$	$S\ L$	$S\ Z$	$S\ -$	$S\ -$	$S\ L$
$q =$	$X\ Y$	$-\ -$	$A\ X$	$-\ A$	$A\ A'$	$A\ -$	$-\ -$
$p' =$	$-\ -$	$-\ S$	$L\ S$	$Z\ S$	$-\ S$	$-\ S$	$L\ S$
$q' =$	$X\ Y$	$-\ -$	$A\ X$	$A\ -$	$A\ A'$	$-\ A$	$-\ -$

FIG. 4. *The contents of PQ_i, PQ_{i+1}, PQ'_i, PQ'_{i+1} for each case of the store-contiguous transformation. X and Y each represent either an arithmetic operation or an empty slot, and Z represents either a load operation or an empty slot. In Case* 1 *the store and load operations reference the same value. In the remaining cases the store and load operations, if present, reference different values.*

Slot $i =$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$p_i =$	L_{18}	L_{19}	$-$	L_{20}	$-$	L_{21}	L_{22}	$-$	L_{14}	L_{15}	$-$	L_{16}	S_5	L_{17}	$-$	L_5	L_{24}	$-$	L_{25}	$-$
$q_i =$	$-$	$-$	$-$	A_{13}	$-$	A_9	$-$	$-$	A_{10}	A_5	$-$	A_7	$-$	$-$	$-$	A_8	A_4	$-$	A_2	$-$
$\rho_i =$	1	2	2	2	2	2	3	3	3	3	3	3	2	3	3	3	3	3	3	3

Slot $i =$	21	22	23	24	25	26	27	28	29	30
$p_i =$	L_{26}	S_2	L_{27}	$-$	L_{23}	L_2	$-$	$-$	$-$	$-$
$q_i =$	A_{11}	$-$	$-$	$-$	A_{12}	A_6	$-$	A_3	$-$	A_1
$\rho_i =$	3	2	3	3	3	3	3	2	2	1

FIG. 5. *The result of applying the store-contiguous transformation to the load-contiguous evaluation in Figure* 3 *with $R = 3$.*

operations, or the number of spill operations plus empty slots, and without causing the number of empty slots to decrease or the maximum number of values in registers to exceed R. The transformation described above can be applied repeatedly to the same store, moving it one time period later in each case, until either it does precede a time period in which R values are in registers, or the store can be deleted (if it is moved later than the load of the same value). Then this transformation can be applied to the preceding store. Since the evaluation is of finite length, each store will eventually either precede a time period containing R values in a register or be deleted.

The movement of one store cannot affect the prior placement of a later store since, as we have shown, the last store to violate the store-contiguous property cannot immediately precede another store operation. Since there are finitely many stores, the sequence of transformations will eventually terminate with each remaining store preceding a time period with R values in registers.

It remains to show that it is possible to satisfy both load-contiguity and store-contiguity in an optimal evaluation. Notice that up to the point of deleting load/store pairs, any transformation that moves a load interchanges the load with a store. Thus load-contiguity is preserved. But empty load/store slots can be introduced wherever a load/store pair is deleted and these may destroy load-contiguity. In this case, however, the load-contiguous transformation can be reapplied. At worst, the transformation described above and the load-contiguous transformation are reapplied k times, i.e., each time a spill is eliminated. □

Store-contiguous transformation. Given a load-contiguous parallel evaluation PQ of a computation F, we create a parallel evaluation PQ' that is both load-contiguous and store-contiguous using the transformation described in Theorem 5.1.

An evaluation PQ' is said to be *contiguous* if it is both load-contiguous and store-contiguous, i.e., if no load operation is scheduled in a later time period than an empty load/store slot, and for any i such that a store operation is scheduled at

time i, $\rho_{i+1} = R$. The result of applying the store-contiguous transformation to the load-contiguous parallel evaluation in Figure 3 is shown in Figure 5.

6. Cost of a *k*-spill parallel evaluation. We have shown that given a k-spill, g-empty slot parallel evaluation of a computation F that uses R registers, there exists another parallel evaluation of F that costs no more, uses no more than R registers, and is contiguous. Furthermore, for load latencies ≤ 2 the contiguous evaluation has no more than $2k + g$ spill operations plus empty slots. We now examine the structure of such a k-spill, g-empty slot contiguous evaluation with a sequence of lemmas and show in Theorem 6.5 that its cost is $\rho(PQ)+2k+\mathrm{g}+|A(F)|$. The results in this section apply with arbitrary load latencies.

LEMMA 6.1 (each store operation and the following operation). *Consider a k-spill, g-empty slot contiguous parallel evaluation PQ of a computation F with arbitrary load latencies and R available registers. If PQ_i contains a store operation S, then $PQ_i = (S, NOP)$, and $PQ_{i+1} = (L, NOP)$ for some load operation L.*

Proof. Let PQ be a k-spill, g-empty slot contiguous parallel evaluation of a computation F such that $\rho(PQ) \leq R$, and consider an arbitrary store operation S in PQ_i. We show by contradiction that $PQ_i = (S, NOP)$, and show that $PQ_{i+1} = (L, NOP)$ for some load operation L. Assume that $PQ_i \neq (S, NOP)$, i.e., $PQ_i = (S, A)$. Then $\rho_i(PQ) = \rho_{i-1}(PQ) - 2$. The number of values in registers at some time can increase by at most one in the next time period, so $\rho_{i+1}(PQ) \leq \rho_i(PQ) + 1 = \rho_{i-1}(PQ) - 1 \leq R - 1$. Thus PQ is not store-contiguous, which is a contradiction. We conclude that $PQ_i = (S, NOP)$.

Now from the store contiguity of PQ we know that $\rho_{i+1}(PQ) = R$. We also know that $\rho_{i-1}(PQ) \leq R$ and therefore, since $PQ_i = (S, NOP)$, $\rho_i(PQ) \leq R - 1$. Thus $\rho_{i+1}(PQ) - \rho_i(PQ) \geq 1$ which implies that $PQ_{i+1} = (L, NOP)$ for some load operation L. ∎

LEMMA 6.2 (operations between two successive store operations). *Consider a k-spill, g-empty slot contiguous parallel evaluation PQ of a computation F with arbitrary load latencies and R available registers. If $PQ_i = (S, NOP)$ for some store operation S and the following store operation exists and is scheduled at time j, $j > i$, then for all m, $i + 1 < m < j$, PQ_m is of the form (L, A) or (NOP, NOP), and $\rho_m(PQ) = R$.*

Proof. Consider a store operation scheduled at time i of a contiguous parallel evaluation PQ with $\rho(PQ) \leq R$, and the following store operation scheduled at time j. We know from Lemma 6.1 that $PQ_{i+1} = (L, NOP)$ and from store-contiguity that $\rho_{i+1}(PQ) = R$. We now show by contradiction that for all m such that $i+1 < m < j$, $\rho_m(PQ) = R$. Assume that for some time m, $i+1 < m < j$, $\rho_m(PQ) < R$. Therefore the number of values in registers decreases by at least one between time $i + 1$ and m. Since there are, by assumption, no stores scheduled at these times, there must be at least one time m', $i + 1 < m' \leq m$ such that $PQ_{m'} = (NOP, A)$. However, there must be a load operation after time j that loads the value stored in slot j. Thus the empty load/store slot m' causes PQ not to be load-contiguous, which is a contradiction. Therefore, for all $i + 1 < m < j$, $\rho_m(PQ) = R$. This implies that for all m, $i + 1 < m < j$, PQ_m is of the form (L, A) or (NOP, NOP). ∎

LEMMA 6.3 (operations before the first store operation). *Consider a k-spill, g-empty slot contiguous parallel evaluation PQ of a computation F with arbitrary load latencies and R available registers. If the first store operation of PQ is scheduled at time i, then a load operation or an empty slot is scheduled at each time preceding time i, and exactly R empty arithmetic slots precede time i.*

Proof. Consider the first time i when a store operation S, if one exists, is scheduled in a contiguous parallel evaluation PQ with $\rho(PQ) \leq R$. Load-contiguity and the fact that there is a load operation scheduled at time $i+1$ imply that there can be no empty load/store slots before slot i, and since there are no stores there, there must be a load operation or an empty slot scheduled at each time before time i.

Store-contiguity implies that $\rho_{i+1}(PQ) = R$, so $\rho_{i-1} = R$ because $PQ_i = (S, NOP)$ and $PQ_{i+1} = (L, NOP)$ for some store operation S and load operation L. Therefore there are R more load operations than arithmetic operations scheduled before time i, which implies that there are exactly R empty arithmetic slots before time i. ∎

LEMMA 6.4 (operations after the last store operation). *Consider a k-spill, g-empty slot contiguous parallel evaluation PQ of a computation F with arbitrary load latencies and R available registers. If the last store operation of PQ is scheduled at time m, then there are no empty arithmetic slots after time $m+1$.*

Proof. Consider the last time m when a store operation S, if one exists, is scheduled in a contiguous parallel evaluation PQ with $\rho(PQ) \leq R$. We claim that there are no empty arithmetic slots after time $m+1$, and show this by contradiction. Note first that there can be no empty load/store slots between time m and the last load operation at time t. As argued above, if there is such an empty load/store slot, PQ would not be load-contiguous. Assume that PQ does contain an empty arithmetic slot m' such that $m+1 < m'$. If there is more than one such slot, we let m' be the first one. Therefore, there is an arithmetic operation or an empty slot scheduled at every time j, where $m+2 \leq j \leq m'-1$. (Recall that the operations scheduled at time $m+1$ have the form (L, NOP).) Since PQ is contiguous, there must be a load operation or an empty slot scheduled at every time j, where $m+1 \leq j \leq m'$. Therefore, $\rho_{m'-1}(PQ) = R$. Since $PQ_{m'} = (L, NOP)$ for some load operation L, $\rho_{m'}(PQ) = R+1$, which is a contradiction. Therefore, the empty arithmetic slot m' cannot exist. ∎

THEOREM 6.5 (cost of a contiguous parallel evaluation). *A k-spill, g-empty slot contiguous parallel evaluation PQ of a computation F with arbitrary load latencies and with $\rho(PQ) \leq R$ has cost $c(PQ) = \rho(PQ) + 2k + g + |A(F)|$.*

Proof. We have shown that in a contiguous parallel evaluation there are two empty arithmetic slots for each store operation (spill), namely, the slot with the store operation and the one immediately after. Furthermore there are no other empty arithmetic slots between a pair of store operations. If any stores exist, there are R empty arithmetic slots before the first store operation and there are no empty arithmetic slots after the last store, except for the slot immediately following the last store which is of the form (L, NOP). If there is at least one spill, $\rho(PQ) = R$ since PQ is contiguous. Therefore, the cost of a k-spill contiguous evaluation, where $k \geq 1$, is the sum of the number of slots containing arithmetic operations, plus the number empty arithmetic slots, plus the number of empty slots, i.e., $\rho(PQ) + 2k + g + |A(F)|$.

If there are no spills in a contiguous parallel evaluation PQ, the number of empty arithmetic slots can be seen to be $\rho(PQ)$. The number of values in registers only increases when operation pairs scheduled at some time are of the form (L, NOP). Since PQ contains no spills, the number of values in registers only decreases when operation pairs scheduled have the form (NOP, A). The number of values in registers is unchanged when operation pairs scheduled have the form (L, A) or (NOP, NOP). Since PQ is load-contiguous, each operation pair of the form (NOP, A) must follow all operation pairs of the form (L, NOP). Therefore, $\rho(PQ) =$ the number of

operation pairs of the form (L, NOP). Since arithmetic operations or empty slots are scheduled at all other times, the cost of a 0-spill contiguous evaluation is $\rho(PQ) + g + |A(F)|$. □

As a consequence of Theorem 6.5 and the preceding lemmas, a k-spill, g-empty slot contiguous evaluation that uses R registers has the following general form. The first store operation, if it exists, is preceded by a series of operation pairs of the form (L, A) interspersed with empty slots and R operation pairs of the form (L, NOP). Each of the k store operations is scheduled in an empty arithmetic slot, (S, NOP), and is followed by an operation pair of the form (L, NOP). Such pairs of load/store operations are followed by a series of operation pairs of the form (L, A) interspersed with empty slots. The evaluation ends with a series of operation pairs of the form (NOP, A), also interspersed with empty slots. It can be seen that the contiguous evaluation in Figure 5 satisfies Lemmas 6.1, 6.2, 6.3, and 6.4 and Theorem 6.5.

Note also that there are $R - 1$ final operations ($\rho(PQ) - 1$ operations if there are no stores) of the form (NOP, A) interspersed with empty slots. This will always be the case since there are R values in registers in the time period after the last store, schedules of dependence trees leave one final value in a register, operation pairs of the form (L, A) or (NOP, NOP) do not change the number of values in registers, and each operation pair of the form (NOP, A) reduces the number of values in registers by one. Any empty slots that follow the first empty arithmetic slot can be deleted without violating precedence. Furthermore the evaluation can be made to begin with R operation pairs of the form (L, NOP), followed by only (L, A) and (NOP, NOP) operation pairs up to the first store operation by simply pushing the leading A operations to the right until they are scheduled consecutively up to the first store operation (or consecutive slots up to the last (L, A) operation pair if there are no stores), and deleting any empty slots that precede the first arithmetic operation by more than one cycle.

7. Optimal parallel evaluations for latency = 1 or latency = 2. We describe an algorithm that solves the parallel scheduling problem given a computation F and R available registers. We first construct a k-spill, g-empty slot sequential evaluation P using an optimal sequential scheduling algorithm, such as the two described in section 3.1. Using the parallel transformation of section 3.3, we construct the parallel evaluation PQ_1. Using the load-contiguous transformation of section 4, we construct the evaluation PQ_2. Using the store-contiguous transformation of section 5, we construct the evaluation PQ_3. If $latency = 1$, we show that using the $latency = 1$ sequential scheduling algorithm to produce P in the first step gives an evaluation PQ_3 that is a solution to the parallel scheduling problem. Similarly, if $latency = 2$ we show that using the $latency = 2$ sequential scheduling algorithm to produce P in the first step gives an evaluation PQ_3 that is a solution to the parallel scheduling problem.

Note that the sequential evaluations produced by the two sequential scheduling algorithms can be quite different since they make different assumptions about the underlying processor. We begin by describing two properties that are independent of the optimal sequential scheduling algorithm used.

LEMMA 7.1. *Given a computation F with arbitrary load latencies and R available registers, there exists a solution to the parallel scheduling problem that is contiguous.*

Proof. By Theorem 4.1 and Theorem 5.1, given any parallel evaluation PQ of F that uses R registers, it is possible to construct another contiguous evaluation that uses R registers, contains no additional spills, and costs no more than PQ. □

As a result of Lemma 7.1, we consider only contiguous evaluations of F.

LEMMA 7.2. *Given a computation F with latency ≤ 2 and R available registers, evaluations P and PQ_3 produced by the parallel scheduling algorithm have the same number of spill operations plus empty slots, and PQ_3 has no fewer empty slots than P.*

Proof. By Lemma 3.2, no parallel evaluation of F that uses R registers can contain fewer spill operations plus empty slots than an optimal sequential evaluation. Since P is optimal, the number of spill operations plus empty slots in the parallel evaluations PQ_3 is not less than $2k + g$. But the transformations used to produce PQ_3 never increase the number of spill operations plus empty slots in the evaluation, as shown in Lemma 4.2 and Theorem 5.1. Therefore the number of spill operations plus empty slots in the parallel evaluation PQ_3 is exactly $2k+g$. Similarly, the number of empty slots in evaluation PQ_3 is not less than g. ∎

We now show that the parallel scheduling algorithm for *latency* $= 1$ gives an optimal evaluation.

THEOREM 7.3. *Given a computation F with latency $= 1$ and R available registers, the parallel scheduling algorithm for latency $= 1$ gives an evaluation, PQ_3, that is a solution to the parallel scheduling problem.*

Proof. By Lemma 7.2, the number of spill operations plus empty slots in parallel evaluation PQ_3 is $2k + g$. Since load latencies equal one we can assume that all evaluations contain no empty slots. Therefore, PQ_3 contains exactly k spills. We first show that only k-spill contiguous evaluations need to be considered. We then show that PQ_3 is an optimal k-spill contiguous evaluation of F.

Consider two contiguous evaluations of F, PQ and PQ', that contain k and k' spills, respectively, such that both use R registers and $k \neq k'$. We show that $c(PQ) \leq c(PQ')$. By Lemma 3.2 and since $g = 0$, no parallel evaluation of F that uses R registers can contain fewer than k spills, so $k < k'$. We first consider the case where $k > 0$. Since PQ is contiguous, and since it contains at least one spill, $\rho(PQ) = R$. Since $k < k'$, PQ' also contains at least one spill. Therefore $\rho(PQ) = \rho(PQ')$. Since both PQ and PQ' are contiguous, $c(PQ) = \rho(PQ) + 2k + |A(F)|$, and $c(PQ') = \rho(PQ') + 2k' + |A(F)|$. Since $k < k'$, $c(PQ) < c(PQ')$. Therefore if $k \neq 0$, no contiguous evaluation with more than k spills is optimal.

We now consider the case where $k = 0$. Since PQ uses R registers, $\rho(PQ) \leq R$ and $c(PQ) = \rho(PQ) + |A(F)|$. Since $k < k'$, PQ' contains at least one spill, so $\rho(PQ') = R$ and $c(PQ') = R + 2k' + |A(F)|$. Therefore, $c(PQ) < c(PQ')$. We have shown that given an arbitrary contiguous evaluation PQ' that uses R registers, if PQ' does not contain k spills, then PQ' cannot be an optimal evaluation of F. We will therefore only consider k-spill contiguous evaluations of F.

By Theorem 6.5, every k-spill contiguous evaluation PQ of F that uses R registers has cost $c(PQ) = \rho(PQ) + 2k + |A(F)|$. If $k > 0$, $c(PQ) = R + 2k + |A(F)|$, which is a constant. That is, all k-spill contiguous evaluations of F have the same cost and every one of them is optimal. In particular, PQ_3 is optimal.

If $k = 0$, $\rho(PQ)$ must be minimized in order to minimize $c(PQ) = \rho(PQ)+|A(F)|$. In [36] it is shown that the sequential scheduling algorithm minimizes the number of registers used. By Lemma 3.1, any parallel evaluation of a computation F can be transformed into a serial evaluation of F without increasing the maximum number of values in registers. Therefore, no parallel evaluation of F uses fewer than $\rho(P)$ registers. The evaluation P is transformed by our algorithm into PQ_2 by applying the parallel transformation followed by the load-contiguous transformation. Since $k = 0$ by assumption, the load-contiguous evaluation is also store-contiguous, and therefore contiguous. As described in section 3.3, the parallel transformation does

not change the maximum number of values in registers. By Theorem 4.1, the load-contiguous transformation also does not increase the maximum number of values in registers. Therefore, $\rho(PQ) = \rho(P)$ and $\rho(PQ)$ is minimized. We conclude therefore that PQ_3 is an optimal no-spill contiguous evaluation of F. ∎

The following theorem shows that the parallel scheduling algorithm for *latency* = 2 gives an optimal evaluation. The theorem relies on results regarding the optimal sequential scheduling algorithm for *latency* = 2 [28]. In particular we use the fact that if the number of available registers is large enough, the algorithm produces an evaluation containing no empty slots, and otherwise the evaluation produced must contain at least one empty slot. While the proof of Theorem 7.3 considered cases based on the number of spills, the proof of Theorem 7.4 considers cases based on the relative sizes of R and the minimum number of registers needed to avoid the use of spill code. With *latency* = 2, it is the number of available registers that affects the existence of empty slots in an optimal parallel evaluation.

THEOREM 7.4. *Given a computation F with latency = 2 and R available registers, the parallel scheduling algorithm for latency = 2 gives an evaluation, PQ_3, that is a solution to the parallel scheduling problem.*

Proof. We prove that no contiguous evaluation of F costs less than PQ_3. Let PQ' be an arbitrary k'-spill, g'-empty slot contiguous evaluation of F that uses R registers, and let $\rho' = \rho(PQ')$. We further let *minreg* equal the minimum number of registers needed to evaluate F without spill code on one processor with load latencies = 1. We first consider the case where $R \geq minreg + 1$. Since R is larger than *minreg*, evaluation P contains no spills and therefore PQ_3 contains no spills. Furthermore, when R is larger than *minreg* the sequential scheduling algorithm produces an evaluation containing no empty slots [28]. Since by Lemma 3.2 the transformations do not change $2k + g$, $c(PQ_3) = minreg + 1 + |A(F)|$. We now compare this cost to the possible costs of PQ'. If $\rho' \geq minreg + 1$, then by Theorem 6.5, $c(PQ_3) \leq c(PQ')$. If $\rho' = minreg$, then the sequential scheduling algorithm produces an evaluation containing at least one empty slot [28]. Since the transformations do not decrease the number of empty slots, $g \geq 1$, and again $c(PQ_3) \leq c(PQ')$. Finally if $\rho' < minreg$, then by assumption $\rho' < R$ and therefore k' must equal zero since PQ' is contiguous. This is a contradiction by Lemma 3.3 because all parallel evaluations that use fewer than *minreg* registers must contain at least one spill.

We next consider the case where $R = minreg$. The cost of evaluation PQ_3 is $R+2k+g+|A(F)|$. If $k' > 0$, then $\rho' = R$ since PQ' is contiguous. Therefore the cost of PQ' is $R+2k'+g'+|A(F)|$ which is not less than $c(PQ_3)$ because $2k+g \leq 2k'+g'$ by Lemma 3.2. If $k' = 0$, then it must be the case that $\rho' = R$, because if $\rho' < minreg$ then PQ' must contain at least one spill by Lemma 3.3. Therefore $c(PQ_3) \leq c(PQ')$ because $2k + g \leq 2k' + g'$.

Finally, we consider the case where $R < minreg$. The cost of evaluation PQ_3 is again $R + 2k + g + |A(F)|$. If $k' > 0$, then $\rho' = R$ and $c(PQ_3) \leq c(PQ')$ as before. If $k = 0$, then this contradicts Lemma 3.3 because $R < minreg$. Therefore in each case $c(PQ_3) \leq c(PQ')$, and we conclude that PQ_3 is an optimal evaluation of F. ∎

8. Computational complexity. We show in this section that the worst-case computational complexity of the parallel scheduling algorithms is $O(nk)$, where $n = |A| + |L|$ is the number of nodes in F and k is the number of spills in the evaluation.

The labeling routine of the sequential scheduling algorithms performs a depth-first traversal of the dependence graph and does a constant amount of work at each vertex. The worst-case complexity of a depth-first traversal of a binary tree is $O(n)$

[16]. Similarly, the worst-case complexity of the scheduling routine with load latencies $= 2$ is $O(n)$.

The parallel transformation of section 3.3 visits every time period in the sequential evaluation once and does a constant amount of work per visit. A sequential evaluation containing k spills is of length $n + 2k$, so the transformation's worst-case complexity is $O(n + k)$.

The load-contiguous transformation in section 4 can be implemented with two pointers. The first pointer points to each load operation in turn, starting from the beginning of the evaluation, and the second pointer always points to the latest empty load/store slot that is earlier than the first pointer. In the worst case, each pointer scans the $n + 2k$ time periods in the parallel evaluation once and is bounded by some constant time per time period. Thus, the transformation's complexity is $O(n + k)$.

The store-contiguous transformation in section 5 can be implemented by considering each store operation in turn, starting from the end of the evaluation. The transformation is repeatedly applied to the same store, moving it one time period later each time, until the transformation can no longer be applied. Applying the transformation once takes constant time. No store is moved more than $n + 2k$ time periods, i.e., from the beginning of the evaluation to the end. Since there are k stores, the transformation will take time $nk+2k^2$ in the worst case. Each value can be spilled at most once, so $k \leq n$. Thus, the complexity of the store-contiguous transformation is $O(nk)$. Since k is the minimum number of spills in any parallel evaluation of F, no stores will be deleted by the transformation and the load-contiguous transformation will not need to be reapplied. Unnecessary Empty slots can be deleted from the evaluation in $O(n + k)$ time. Thus, the worst-case complexity of the parallel scheduling algorithms is $O(nk)$.

9. Summary. We have presented an $O(nk)$ algorithm that solves the parallel scheduling problem for $latency = 1$ or $latency = 2$ by applying a series of transformations to an optimal sequential evaluation. We have shown that only contiguous evaluations need to be considered in searching for optimal parallel evaluations, and that optimal serial and optimal parallel evaluations have the same number of spill operations plus empty slots. We have fully described the structure of contiguous parallel evaluations and shown that the cost of the k-spill parallel evaluation PQ found by the algorithm is $\rho(PQ) + 2k + g + |A(F)|$.

These results may encourage further research on other, related problems. For example, extensions of our results may apply to trees with arbitrary arity, and to processors that can issue more operations simultaneously. The new transformations we describe may also form the basis for worst-case results for more general problems that cannot be efficiently solved.

Acknowledgments. We thank the anonymous referees for their detailed suggestions which have greatly improved this paper.

REFERENCES

[1] A. V. Aho and S. C. Johnson, *Optimal code generation for expression trees*, J. ACM, 23 (1976), pp. 488–501.

[2] P. Bergner, P. Dahl, D. Engebretsen, and M. O'Keefe, *Spill code minimization via interference region spilling*, in ACM SIGPLAN Conference on Programming Language Design and Implementation, Las Vegas, NV, ACM Press, 1997.

[3] D. Bernstein and I. Gertner, *Scheduling expressions on a pipelined processor with a maximal delay of one cycle*, ACM Trans. Prog. Lang. Syst., 11 (1989), pp. 57–66.

[4] D. Bernstein, J. Jaffe, and M. Rodeh, *Scheduling arithmetic and load operations in parallel with no spilling*, SIAM J. Comput., 18 (1989), pp. 1098–1127.
[5] D. Bernstein, M. Rodeh, and I. Gertner, *On the complexity of scheduling problems for parallel/pipeplined machines*, IEEE Trans. Comput., 38 (1989), pp. 1308–1313.
[6] P. Bose, *Optimal code generation for expressions in super scalar machines*, in Proceedings of the Fall Joint Computer Conference, IEEE Computer Society Press, Los Alamitos, CA, 1986, pp. 372–379.
[7] D. Bradlee, S. Eggers, and R. Henry, *Integrating register allocation and instruction scheduling for RISCS*, in Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems, ACM Press, 1991.
[8] P. Briggs, K. D. Cooper, K. Kennedy, and L. Torczon, *Coloring heuristics for register allocation*, SIGPLAN Notices, 24 (1989), pp. 275–284.
[9] P. Briggs, K. D. Cooper, and L. Torczon, *Improvements to graph coloring register allocation*, ACM Trans. Prog. Lang. Syst., 16 (1994), pp. 428–455.
[10] J. Bruno, J. Jones, and K. So, *Deterministic scheduling with pipelined processors*, IEEE Trans. Comput., C-29 (1980), pp. 308–316.
[11] J. Bruno and R. Sethi, *Code generation for a one-register machine*, J. ACM, 23 (1976), pp. 502–510.
[12] D. Callahan and B. Koblenz, *Register allocation via hierarchical graph coloring*, in ACM SIGPLAN '91 Conference on Programming Language Design and Implementation, Toronto, Canada, ACM Press, 1991.
[13] M. Carlisle, *On Local Register Allocation*, Thesis, Computer Science Department, University of Delaware, Newark, DE, 1991.
[14] F. C. Chow and J. L. Hennessy, *The priority-based approach to register allocation*, ACM Trans. Prog. Lang. Syst., 12 (1990), pp. 503–536.
[15] E. Coffman and R. Graham, *Optimal scheduling for two-processor systems*, Acta Inform., 1 (1972), pp. 200–213.
[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*, MIT Press, Cambridge, MA, 1991.
[17] G. J. Chaitin, et al., *Register allocation via coloring*, Computer Languages, 6 (1981), pp. 47–57.
[18] G. J. Chaitin, et al., *Register allocation and spilling via graph coloring*, in Proceedings of the ACM SIGPLAN '86 Symposium on Compiler Construction, New York, 1982, pp. 98–105.
[19] M. Farach and V. Liberatore, *On local register allocation*, in ACM-SIAM Symposium on Discrete Algorithms, ACM Press, 1998.
[20] L. George and A. Appel, *Iterated register coalescing*, ACM Trans. Prog. Lang. Syst., 18 (1996), pp. 300–324.
[21] P. B. Gibbons and S. S. Muchnick, *Efficient instruction scheduling for a pipelined architecture*, in Proceedings of the ACM SIGPLAN '86 Symposium on Compiler Construction, New York, 1986, pp. 11–16.
[22] J. R. Goodman and W. C. Hsu, *Code scheduling and register allocation in large basic blocks*, in Proceedings of the 1988 International Conference on Supercomputing, Orlando, FL, ACM Press, 1988, pp. 442–452.
[23] J. Hennessy and T. Gross, *Code generation and reorganization in the presence of pipeline constraints*, in 9th Annual ACM Symposium on Principles of Programming Languages, Albuquerque, NM, ACM Press, 1982, pp. 120–127.
[24] J. Hennessy and T. Gross, *Postpass code optimization of pipeline constraints*, ACM Trans. Prog. Lang. Syst., 5 (1983), pp. 422–448.
[25] L. P. Horwitz, R. M. Karp, R. E. Miller, and S. Winograd, *Index register allocation*, J. ACM, 13 (1966), pp. 43–61.
[26] W. Hsu, C. N. Fischer, and J. R. Goodman, *On the minimization of loads/stores in local register allocation*, IEEE Transactions on Software Engineering, 15 (1989), pp. 1252–1260.
[27] T. Hu, *Parallel sequencing and assembly line problems*, Oper. Res., 9 (1961), pp. 841–848.
[28] S. Kurlander, T. Proebsting, and C. Fischer, *Efficient instruction scheduling for delayed-load architectures*, ACM Trans. Prog. Lang. Syst., 17 (1995), pp. 740–776.
[29] G. Lueh and T. Gross, *Call-cost directed register allocation*, in ACM SIGPLAN Conference on Programming Language Design and Implementation, Las Vegas, NV, ACM Press, 1997.
[30] I. Nakata, *On compiling algorithms for arithmetic expressions*, Communications of the ACM, 10 (1967), pp. 492–494.
[31] K. Palem and B. Simons, *Scheduling time-critical instructions on RISC machines*, ACM Trans. Prog. Lang. Syst., 15 (1993), pp. 632–658.
[32] S. S. Pinter, *Register allocation with instruction scheduling: A new approach*, SIGPLAN

Notices, 28 (1993), pp. 248–257.
[33] T. Proebsting and C. Fischer, *Demand-driven register allocation*, ACM Trans. Prog. Lang. Syst., 18 (1996), pp. 683–710.
[34] R. R. Redziejowski, *On arithmetic expressions and trees*, Communications of the ACM, 12 (1969), pp. 81–84.
[35] R. Sethi, *Complete register allocation problems*, SIAM J. Comput., 4 (1975), pp. 226–248.
[36] R. Sethi and J. D. Ullman, *The generation of optimal code for arithmetic expressions*, J. ACM, 17 (1970), pp. 715–728.
[37] J. D. Ullman, *NP-complete scheduling problems*, J. Comput. System Sci., 10 (1975), pp. 384–393.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1942–1961

A POLYNOMIAL TIME APPROXIMATION SCHEME FOR INFERRING EVOLUTIONARY TREES FROM QUARTET TOPOLOGIES AND ITS APPLICATION*

TAO JIANG†, PAUL KEARNEY‡, AND MING LI‡

Abstract. Inferring evolutionary trees has long been a challenging problem for both biologists and computer scientists. In recent years research has concentrated on the *quartet method* paradigm for inferring evolutionary trees. Quartet methods proceed by first inferring the evolutionary history for every set of four species (resulting in a set Q of inferred quartet topologies) and then recombining these inferred quartet topologies to form an evolutionary tree. This paper presents two results on the quartet method paradigm. The first is a polynomial time approximation scheme (PTAS) for recombining the inferred quartet topologies optimally. This is an important result since, to date, there have been no polynomial time algorithms with performance guarantees for quartet methods. To achieve this result the natural *denseness* of the set Q is exploited. The second result is a new technique, called *quartet cleaning*, that detects and corrects errors in the set Q with performance guarantees. This result has particular significance since quartet methods are usually very sensitive to errors in the data. It is shown how quartet cleaning can dramatically increase the accuracy of quartet methods.

Key words. dense instance, evolutionary tree, approximation algorithm, quartet method, smooth polynomial

AMS subject classifications. 68Q25, 92B99

PII. S0097539799361683

1. Introduction.

A fundamental problem in computational biology is inferring an evolutionary tree from biological data. Virtually all formulations of this problem (maximum likelihood, maximum parsimony, minimum distance, etc. [10, 15]) are NP-hard, and so, methods tend to be either heuristic (and seldom with performance guarantees) or prohibitively exhaustive. This is a real conundrum for evolutionary biologists as datasets can be very large forcing them to use heuristic methods that can lead to erroneous results.[1] The difficulties of inferring evolutionary trees that hamper biologists have also catalyzed a surge of algorithmic research in the computer science community. Despite this attention, efficient algorithms with performance guarantees have been slow in coming.

In recent years the computational biology community has focused on the *quartet method* paradigm for inferring evolutionary trees [3, 5, 8, 11, 14]. Quartet methods utilize topological information on sets of four labels[2] to infer an evolutionary tree. To illustrate, consider the four possible trees labeled by $\{a, b, c, d\}$ as depicted in Figure

*Received by the editors September 27, 1999; accepted for publication (in revised form) October 12, 2000; published electronically March 20, 2001. A preliminary version of this paper appeared in *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, IEEE, 1998.
http://www.siam.org/journals/sicomp/30-6/36168.html

†Department of Computer Science, University of California, Riverside, CA 92521 (jiang@cs.ucr.edu). This author was supported in part by NSERC research grant OGP0046613, a CITO grant, and a UCR startup grant. He is on leave from McMaster University.

‡Department of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada (pkearney@math.uwaterloo.ca, mli@math.uwaterloo.ca). The work of the second author was supported in part by a CITO grant and by NSERC research grant 160321. The work of the third author was supported in part by NSERC research grant OGP0046506, CITO, a CGAT grant, and the Steacie Fellowship.

[1] As exemplified by the *Out of Africa* fiasco [16, 17].

[2] A label may represent a species or, more generally, a DNA or protein sequence.

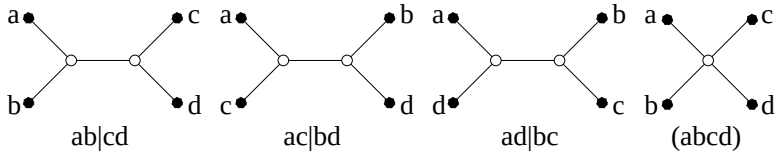


FIG. 1.1. *The four possible trees labeled by* $\{a, b, c, d\}$.

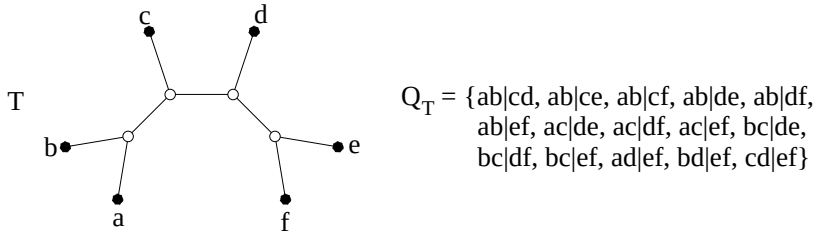


FIG. 1.2. *A tree* T *labeled by* $\{a, b, c, d, e, f\}$ *and* Q_T.

FIG. 1.3. *The quartet method paradigm.*

1.1. These labeled trees are denoted $ab|cd$, $ac|bd$, $ad|bc$, and $(abcd)$, respectively. The topology induced by the quartet $\{a, b, c, d\} \subseteq S$ in a tree T labeled by S is $ab|cd$ if a, b and c, d reside in disjoint subtrees of T. Q_T is defined to be the set of all topologies induced in T by quartets taken from S. For example, Figure 1.2 depicts a labeled degree-3 (i.e., fully resolved) tree T and its set of induced quartet topologies Q_T. We note that an evolutionary tree is usually represented by a labeled degree-3 tree with weighted edges. However, often the edge weights are determined after the shape of the tree is determined, as is the case for quartet methods. For the duration of the paper let evolutionary tree be synonymous with degree-3 labeled tree. Note that the set Q_T does not contain topologies of the form $(abcd)$ if T is a degree-3 tree.

Quartet methods are motivated by the following theorem that reveals the strong relationship between Q_T and T [7].

THEOREM 1.1. *Let* T *be an evolutionary tree.* Q_T *is unique to* T*, and furthermore,* T *can be recovered from* Q_T *in polynomial time.*

In other words, induced quartet topology provides much information about an evolutionary tree. This motivates the quartet method paradigm which is the following two step approach to inferring evolutionary trees (see Figure 1.3). Let T denote the evolutionary tree being estimated whose leaves are labeled by the elements of S:

1. A *quartet topology inference method* is used to infer the topology of each quartet in S from the data (typically DNA or protein sequence data). This results in a set Q of inferred quartet topologies.
2. A *quartet recombination method* is used to recombine the quartet topologies in Q to form an estimate T' of T.

There are several quartet topology inference methods including neighbor joining [12], the ordinal quartet method [11], maximum likelihood [9], and maximum

parsimony [15]. Quartet recombination methods attempt to solve variations of the following optimization problem.

MAXIMUM QUARTET CONSISTENCY (MQC).
Instance: A set Q of quartet topologies over label set S.
Goal: Find an evolutionary tree T labeled by S that maximizes $|Q_T \cap Q|$.

At this point we make an important distinction between two versions of MQC: *complete* MQC and *incomplete* MQC. A set Q of quartet topologies is *complete* if Q contains a quartet topology for each quartet over label set S. Incomplete MQC permits the input Q to be incomplete whereas the input to complete MQC is complete. Incomplete MQC is NP-hard [13]. In section 4 a proof of the NP-completeness of complete MQC is presented. Due to these results, most quartet recombination methods are heuristic or solve weaker optimization requirements. Examples are the Q^* method [5], the short quartet method [8], a semidefinite programming method [3], and quartet puzzling [14]. None of these methods give a performance guarantee. Despite the popularity of quartet methods, the absence of performance guarantees has been a legitimate criticism of the quartet method approach.

The distinction between complete and incomplete MQC is important for two reasons. First, in practice one almost always can obtain complete Q. Second, in this paper we present a polynomial time approximation scheme (PTAS) for complete MQC. In fact, this is the first PTAS for inferring an evolutionary tree under the quartet method paradigm and thus is a significant advancement in the development of algorithms for inferring evolutionary trees. A PTAS is desirable since it allows the approximation of an optimal solution with arbitrary precision (at the cost of efficiency). In contrast, Steel's proof that incomplete MQC is NP-hard can be adapted to show that incomplete MQC is MAX-SNP–hard; hence there is no PTAS for incomplete MQC unless NP=P, by the results of [2]. It should be pointed out that one may wish to weight quartets in practice. However, the weighted version of MQC is MAX-SNP–hard, as it generalizes the incomplete MQC, and thus has no PTAS.

Instances of complete MQC are *dense* relative to instances of incomplete MQC. Recently, the examination of dense versions of such MAX-SNP problems as Max–Cut, Betweenness, and Max–k–Sat has yielded PTASs for these problems [1, 2]. Dense instances of problems such as Max–Cut are graphs with $\Omega(n^2)$ edges whereas dense instances of Max–k–Sat are boolean k–Sat formulae with $\Omega(n^k)$ clauses. MQC is an example of an applied problem that motivates the examination of dense problems. How the natural denseness of an instance of MQC can be exploited to obtain a PTAS is explored further in section 2. In section 5 it is shown that the MQC PTAS can be extended to an important weighted variation of the problem and that this weighted version of the PTAS can be utilized to solve a consensus problem. For the duration of the paper MQC can be assumed to mean complete MQC.

Our second result is a new technique, called *quartet cleaning*, that can detect and correct quartet errors in the set Q of inferred quartet topologies. The quartet topology $ab|cd \in Q$ is called a quartet error if $ab|cd \notin Q_T$, where T is the evolutionary tree being estimated.

The practical motivation for quartet cleaning is that the accuracy of most quartet recombination methods depends critically upon the quality of the set Q. To illustrate, consider the sensitivity of the Q^* method and the short quartet method to quartet errors in Q. In particular, if e is an edge of T then $\{a, b, c, d\}$ is a quartet *across* e if a and b are in a separate component of $T - \{e\}$ than c and d (see Figure 1.4(i)). Both the Q^* method and the short quartet method have the property that a single

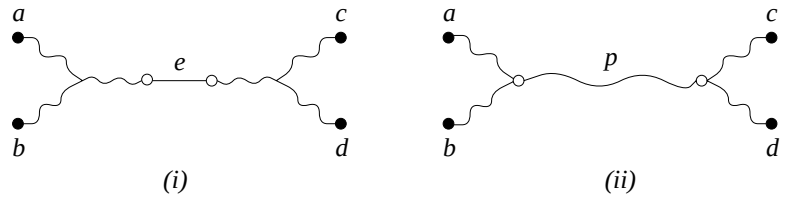


FIG. 1.4. (i) *ab|cd is a quartet across edge e.* (ii) *ab|cd is a quartet across path p.*

quartet error in Q involving a quartet across e can result in the edge e of T not being recovered. (These methods return a contraction of T at e.)

If there are m quartets across e in T and Q contains no more than $\alpha\sqrt{m}$ quartet errors involving quartets across e, where α is a constant, then quartet cleaning applied to Q returns a set Q' of quartet topologies where all of these quartet errors across e have been corrected. This results in a dramatic decrease in the sensitivity of quartet recombination methods such as Q^* method and the short quartet method to quartet errors; they can now tolerate up to $\alpha\sqrt{m}$ quartet errors across edge e, where before they could tolerate none. The bound $\alpha\sqrt{m}$ is shown to be asymptotically optimal in that no algorithm can correct more than $\alpha\sqrt{m}$ quartet errors across an edge.

A polynomial algorithm for quartet cleaning is presented in section 3. It makes nontrivial use of the PTAS for the MQC problem described above and thus serves as another motivation for this PTAS. This suggests that the ideas and techniques developed here are powerful and may find wider use for inferring evolutionary trees.

2. A PTAS for MQC. Let Q be an instance of MQC with label set S. Our discussion begins with the observation that $|Q_{T_{OPT}} \cap Q| \geq \binom{n}{4}/3$ where T_{OPT} is an optimal solution [3, 4]. This follows since a randomly selected tree has a 1/3 chance of inducing $ab|cd \in Q$, for each quartet $\{a, b, c, d\}$. Hence, for some constant c, $|Q_{T_{OPT}} \cap Q| \geq cn^4$. Our goal is then to find an approximation algorithm such that

$$|Q_{T_{APX}} \cap Q| \geq |Q_{T_{OPT}} \cap Q| - \epsilon n^4,$$

where T_{APX} is the result of the approximation algorithm.

The approximation algorithm is founded upon two concepts: a k–bin decomposition of T_{OPT} and smooth integer polynomial programs.

DEFINITION 2.1. *T_k is a k-bin decomposition of T_{OPT} if there is a partition of S into* bins *S_1, S_2, ..., S_k such that the following hold.*

- *For each S_i, $|S_i| \leq 6n/k$. Furthermore, there is a vertex v_i of degree $|S_i| + 1$, called the* bin root*, that is adjacent to each vertex in S_i.*
- *For all quartets $\{a, b, c, d\}$ where a, b, c, and d are in different bins of T_k, $ab|cd \in Q_{T_{OPT}}$ if and only if $ab|cd \in Q_{T_k}$.*

An example of a k-bin decomposition appears in Figure 2.1.

Section 2.1 will discuss k-bin decompositions in detail. In particular, it is shown that there is a k-bin decomposition T_k of T_{OPT} that is a good approximation of T_{OPT}, i.e., $|Q_{T_k} \cap Q| \geq |Q_{T_{OPT}} \cap Q| - (c'/k)n^4$, for some constant c'. Our approach is to approximate T_{OPT} indirectly by approximating T_k.

Consider a fixed k. Let K be T_k with all leaves removed (and thus the leaves of K are the bin roots of T_k). K is called the *kernel* of T_k and T_k is called a *completion* of K. K is completed to T_k by providing a label-to-bin assignment.

If the kernel K of T_k is known, then, to approximate T_k, it suffices to determine an approximately optimal label-to-bin assignment for K. This problem is formalized as follows.

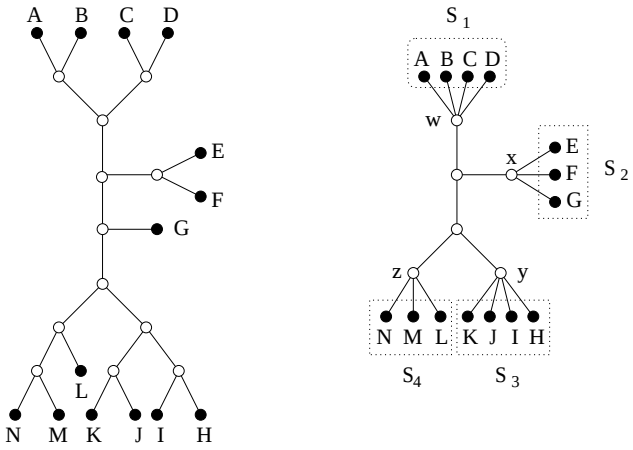


FIG. 2.1. *A tree (left) and a 4-bin decomposition of the tree (right) with bin roots w, x, y, and z.*

LABEL-TO-BIN ASSIGNMENT (LBA).
Instance: Set Q of quartet topologies and (degree-3) kernel K with k leaves.
Goal: Find a completion T' of K that maximizes $|Q_{T'} \cap Q|$.

It follows from the result in [6] that LBA is NP-hard. In section 2.2, LBA is formulated as a smooth integer polynomial problem and a PTAS for LBA is presented. In particular, it is shown that, for any $\epsilon > 0$, $|Q_{T'} \cap Q| \geq |Q_{\hat{T}} \cap Q| - \epsilon n^4$, where Q and K denote the instance of LBA, T' is the completion of K produced by our PTAS and $\hat{T}$ is an optimal completion of K.

The approximation algorithm solves (approximately) an instance of LBA for every tree with k leaves. Since k is a constant, this can be done in polynomial time using the PTAS for LBA. Let T_{APX} be the completed tree obtained that maximizes $|Q_{T_{APX}} \cap Q|$. Since the kernel K of T_k is one of the trees completed, it follows that $|Q_{T_{APX}} \cap Q| \geq |Q_{T'} \cap Q|$ where T' is the completion of K.

2.1. Decomposing the optimal tree. First we demonstrate that there is a k-bin decompositions that is a good approximation of the optimal evolutionary tree.

THEOREM 2.2. *There is a k-bin decomposition T_k of T_{OPT} such that $|Q_{T_k} \cap Q| \geq |Q_{T_{OPT}} \cap Q| - (c'/k)n^4$ for some constant c'.*

Theorem 2.2 is proven algorithmically. For convenience, let $T = T_{OPT}$ and assume that T is degree-3, without loss of generality. Let T_k denote the result of the following algorithm.

ALGORITHM k-BIN DECOMPOSITION(T).

1. Root T at an arbitrary internal vertex. Let $T(v)$ denote the subtree of T rooted at v.
2. Traverse T, beginning at the root, such that for each vertex v visited:
   If $|T(v)| \leq 6n/k$ then
   - contract all internal edges of $T(v)$,
   - label v as a bin root if v is not a leaf, and
   - continue traversal at v's parent.

   Otherwise, continue traversal at an unvisited child of v.
3. For each bin root v with parent v' and sibling u':
   If $|T(v)| \leq 3n/k$ and u' has a child u with $|T(u)| \leq 3n/k$ then
   - contract $\{u', v'\}$,

- transfer the leaves in $T(u)$ to the bin of v, and delete u.

4. For each leaf u of T not assigned to a bin, bisect the edge between u and its parent with a new vertex v, and mark v as a bin root.

Note that the last step of the algorithm is necessary since a leaf cannot be a bin root. Call a bin of T_k *small* if it has size less than $3n/k$. A bin root is *small* if its bin is small. Call a bin of T_k *large* if it has size between $3n/k$ and $6n/k$, inclusive. A bin root is *large* if its bin is large. Let s denote the number of small bins in T_k and l the number of large bins in T_k.

LEMMA 2.3. $s < 2l$.

Proof. We prove the following by induction: For every h, if u is a vertex of height h and is not a bin root then the lemma holds for the subtree $T(u)$.

For the base case, assume that u has children p and q. It cannot be that both p and q are small otherwise $|T(u)| < 6n/k$ and the algorithm would not have visited p and q. Hence, the lemma is true for $T(u)$.

In general, assume that u has children p and q. If both p and q are bin roots then the argument for the base case applies. If neither p nor q are bin roots then the inductive hypothesis applies to $T(p)$ and $T(q)$, and hence the lemma holds for $T(u)$. Otherwise, suppose that p is not a bin root but q is.

If q is large then the induction can be applied to $T(p)$ and the claim follows. Otherwise, suppose that q is small. Let p_1 and p_2 be the children of p. Neither p_1 nor p_2 are small, since otherwise the algorithm would have merged one of $T(p_1)$ and $T(p_2)$ with $T(q)$ in step 3. By the inductive hypothesis, $s_1 < 2l_1$ and $s_2 < 2l_2$ where s_1, l_1, s_2, and l_2 are the numbers of small and large bins in $T(p_1)$ and $T(p_1)$, respectively. It follows that $s_1 + s_2 + 1 < 2(l_1 + l_2)$, and hence the claim holds for $T(u)$. ∎

Since each large bin of T_k has size at least $3n/k$, $l \leq k/3$. T_k has $l + s$ bins, and so, $l + s < l + 2l = 3l \leq k$. We conclude that T_k has less than k bins each with size bounded by $6n/k$. Furthermore, since T_k was obtained from T by contracting edges and transferring leaves to neighboring bins, it follows that, for all quartets $\{a, b, c, d\}$ where a, b, c, and d are in different bins of T_k, $ab|cd \in Q_T$ if and only if $ab|cd \in Q_{T_k}$. The following lemma completes the proof of Theorem 2.2.

LEMMA 2.4. $|Q_{T_k} \cap Q| \geq |Q_{T_{OPT}} \cap Q| - (c'/k)n^4$ *for some constant* c'.

Proof. Observe that $|Q_{T_k} \cap Q| \geq |Q_{T_{OPT}} \cap Q| - |Q'|$ where Q' is the set of all quartets with an induced topology in T_{OPT} that is not induced in T_k. Let $\{a, b, c, d\} \in Q'$. It follows that at least two of these labels are in the same bin of T_k.

There are at most $k(6n/k)^2 n^2 = 36n^4/k$ quartets with two labels in the same bin. Similarly, there are at most $36n^4/k^2$ and $36n^4/k^3$ quartets with three and four labels in the same bin, respectively. It follows that

$$\begin{aligned} |Q_{T_k} \cap Q| &\geq |Q_{T_{OPT}} \cap Q| - 3(36n^4/k) \\ &\geq |Q_{T_{OPT}} \cap Q| - (108/k)n^4. \end{aligned}$$ ∎

2.2. A PTAS for LBA. Let Q and K be an instance of the LBA problem where K has k leaves. $\hat{T}$ is an optimal completion of K if $\hat{T}$ maximizes $|Q_{\hat{T}} \cap Q|$ over all completions of K. To formalize this optimization problem *smooth polynomials* are used. A degree d polynomial $p(x)$ is *t-smooth*, where t is a constant, if the coefficient of each degree i term is in the interval $[-tn^{d-i}, tn^{d-i}]$ for $0 \leq i \leq d$.

Define the 0-1 label-to-bin assignment $x = (x_{sb})$ as follows: $x_{sb} = 1$ if label s is assigned to bin b; otherwise x_{sb} is 0. For each quartet $\{a, b, c, d\} \in Q$, create a degree

4 polynomial

$$p_{ab|cd}(x) = \sum_{ij|kl \in Q_K} x_{ai}x_{bj}x_{ck}x_{dl},$$

and define

$$p(x) = \sum_{ab|cd \in Q} p_{ab|cd}(x).$$

Observe that $p(x)$ is 1-smooth. To ensure that each label is assigned to exactly one bin, the following constraints are added: for each label s, $\sum_{b=1}^{k} x_{sb} = 1$. We also require that none of the bins are too large: for each bin b, $\sum_{s=1}^{n} x_{sb} \leq 6n/k$.

Clearly, if x satisfies these constraints then $p_{ab|cd}(x) = 1$ if $ab|cd \in Q_{T'}$, and $p_{ab|cd}(x) = 0$ if $ab|cd \notin Q_{T'}$, where T' is the completion of K specified by the label-to-bin assignment x. Hence, our optimization problem is to find a 0-1 label-to-bin assignment $x = (x_{sb})$ so that

$$\begin{aligned} &p(x) \text{ is maximized,} \\ &\sum_{b=1}^{k} x_{sb} = 1 \text{ for each label } s, \\ &\sum_{s=1}^{n} x_{sb} \leq 6n/k \text{ for each bin } b. \end{aligned}$$

Arora, Frieze, and Kaplan [1] present a PTAS for solving t-smooth integer polynomial programs.

THEOREM 2.5. *For each $\epsilon > 0$ there is a polynomial time algorithm that produces a 0-1 assignment x such that $p(x) \geq m - \epsilon n^d$ where $p(x)$ is a t-smooth polynomial of degree d with maximum value m.*

The PTAS of Theorem 2.5 first solves the fractional version of the problem to obtain a solution $(\hat{x}_{sb})$. Randomized rounding is then used to obtain a 0-1 solution (x_{sb}). However, the rounding procedure used rounds each $\hat{x}_{sb}$ individually. This does not quite work here because of the additional constraints $\sum_{b=1}^{k} x_{sb} = 1$ for each label s. Hence, the following randomized rounding procedure is used instead: with probability $\hat{x}_{sb}$, $x_{sb} = 1$ and $x_{sj} = 0$ for all $j \neq b$. This ensures that exactly one of $x_{s1}, \ldots, x_{sk}$ is assigned 1 and the rest are assigned 0. This modification can be easily incorporated so that Theorem 2.5 holds. Following from the above discussion and Theorem 2.5, we have the following theorem.

THEOREM 2.6. *For each $\epsilon > 0$, there is a polynomial time algorithm that, for each instance Q and K of LBA, produces a completion T' of K such that $|Q_{T'} \cap Q| \geq |Q_{\hat{T}} \cap Q| - \epsilon n^4$ where $\hat{T}$ is an optimal completion of K.*

Combining the above results we can establish the following approximation result.

THEOREM 2.7. *For each $\epsilon > 0$, there is a polynomial time algorithm that, for each instance Q of* MQC*, produces a tree T_{APX} such that $|Q_{T_{APX}} \cap Q| \geq (1-\epsilon)|Q_{T_{OPT}} \cap Q|$.*

Proof. Let T_{OPT} and T_{APX} be defined as before and T_k be a k-bin decomposition of T_{OPT} that satisfies Theorem 2.2 for some constant k to be determined. Combining Theorems 2.6 and 2.2, we have that

$$\begin{aligned} |Q_{T_{APX}} \cap Q| &\geq |Q_{T_{OPT}} \cap Q| - (c'/k + \epsilon_1)n^4 \\ &\geq (1 - c'/(ck) - \epsilon_1/c)|Q_{T_{OPT}} \cap Q| \end{aligned}$$

for any constant $\epsilon_1 > 0$, since $|Q_{T_{OPT}} \cap Q| \geq cn^4$. The theorem result follows by choosing ϵ_1 sufficiently small and k sufficiently large. □

3. Quartet cleaning. Let T be an evolutionary tree and Q an estimate of Q_T. In order to correct quartet errors in Q we assume the following quartet error model: For each edge e of T there are at most $\alpha\sqrt{|Q_T(e)|}$ quartet errors in Q involving quartets across e where α is a constant to be determined and $Q_T(e)$ denotes the set of quartet topologies across the edge e of T (see Figure 1.4(i)). In this section we present a polynomial algorithm for correcting all quartet errors in Q under this error model. It is also shown that the above upper bound on quartet errors is asymptotically tight by proving a matching lower bound. More precisely, we prove that no algorithm can correctly infer the tree T when the set Q contains more than $\sqrt{|Q_T(e)|}$ errors across some edge e. Therefore, our algorithm is (asymptotically) optimal in terms of its power to correct quartet errors across an edge of T.

The section is organized as follows. We first define a variant of MQC, called the *minimum inconsistent balanced bipartition* (MIBB) problem, and devise a polynomial time approximation algorithm for MIBB with an additive error of ϵn^4 for any constant $\epsilon > 0$, using the same technique utilized by the PTAS for MQC. This approximation algorithm is then used recursively to clean quartets. The lower bound on quartet errors is given in section 3.3.

3.1. MIBB and its approximation. Let $S = \{1, \ldots, n\}$ denote the set of leaf labels. Each edge e of the evolutionary tree T induces a bipartition $X|Y$ of the labels. The quartets across the edge e are also referred to as the quartets induced by the bipartition $X|Y$. The bipartition $X|Y$ is called a *balanced bipartition* if $|X| \leq 2n/3$ and $|Y| \leq 2n/3$. An *edge separator* of the tree T is any edge that induces a bipartition $X|Y$ with the property that $|X| \leq 8n/9$ and $|Y| \leq 8n/9$. It is easy to see that T has at least one edge separator. We consider the following variant of MQC.

Minimum Inconsistent Balanced Bipartition (MIBB).
Instance: Set Q containing a quartet topology for each quartet of labels in S.
Goal: Find a balanced bipartition $A|B$ that induces the minimum number of quartet topologies inconsistent with the set Q. That is, we want to minimize the number of quartets $\{a, b, c, d\} \subseteq S$ such that $a, b \in A$, $c, d \in B$, and $ac|bd \in Q$.

MIBB is known to be NP-hard [6]. By formulating MIBB as a 2-bin variant of LBA, an approximation algorithm for MIBB with additive error ϵn^4 can be derived for any constant $\epsilon > 0$. This results in the following theorem.

Theorem 3.1. *For each $\epsilon > 0$, there is a polynomial time algorithm that produces a balanced bipartition $A|B$ that induces at most ϵn^4 more quartet topologies inconsistent with the set Q than an optimal balanced bipartition.*

3.2. A recursive algorithm for cleaning quartets. In this section we prove the following.

Theorem 3.2. *For some $\alpha > 0$, there is a polynomial time algorithm that produces the correct evolutionary tree T given a (complete) set Q of quartet topologies which contains at most $\alpha\sqrt{|Q_T(e)|}$ errors across any edge e of T.*

Before describing the algorithm in detail, we sketch its basic idea. First, we observe that the bipartition $A|B$ obtained by the approximation algorithm for MIBB on input Q is also a good approximation of a minimum inconsistent balanced bipartition for the set Q_T, since Q contains at most a total of $\alpha n^3 = o(n^4)$ erroneous quartet topologies. Moreover, we show that the bipartition $A|B$ in fact comes very close to the bipartition $X|Y$ induced by some edge separator of T, i.e., the symmetric differences

$A \oplus X$ and $B \oplus Y$ are very small. We *bootstrap* the bipartition $A|B$ by repeatedly swapping and joining incorrectly placed pairs of labels until it actually becomes the correct bipartition $X|Y$. Then we recursively bipartition sets X and Y independently, taking into account the labels in the set Y and X, respectively. Consider the case of bipartitioning X with the presence of Y. Let T_X denote the subtree of T induced by X. Observe that we can still approximately bipartition the set X as long as the errors in Q across an edge separator of T_X is significantly smaller than $|X|^4$. That is, we should have $|X|^4 \gg \alpha\sqrt{n^2|X|^2} = \alpha n|X|$, i.e., $|X| \gg n^{1/3}$. Hence, we stop the recursion at $|X| = \sqrt{n}$ and switch to a different algorithm which attempts to reconstruct the subtree T_X by directly taking advantage of the existence of a large "reference set" $Y = S - X$.[3] The details of the three main parts of the quartet cleaning algorithm are given below.

• **Inferring the first bipartition of *T*.** Fix a sufficiently small constant $\epsilon > 0$ so that all the inequalities below involving ϵ will hold. We run the approximation algorithm for MIBB on set Q to get a balanced bipartition $A|B$ of S. By Theorem 3.1, $A|B$ induces at most ϵn^4 quartet topologies inconsistent with Q and thus at most $\epsilon n^4 + O(n^3)$ quartet topologies inconsistent with Q_T. The following lemma shows that $A|B$ is "almost correct" in the sense that it is actually very close to the bipartition induced by some edge separator of T.

LEMMA 3.3. *Let e_0 be an edge separator of T inducing a bipartition $X|Y$ such that* (i) $|A \cap X| \geq |A|/3 \geq n/9$, (ii) $|B \cap Y| \geq |B|/3 \geq n/9$, *and* (iii) $\max\{|A \cap Y|, |B \cap X|\}$ *is minimized. (It is easy to see that such an edge e_0 exists.) Then* $\max\{|A \cap Y|, |B \cap X|\} \leq 11\epsilon^{1/3}n$.

Proof. Without loss of generality, assume that $|A \cap Y| \geq |B \cap X|$. Let $\epsilon_1 = |B \cap X|/n$ and $\epsilon_2 = |A \cap Y|/n$. Then

$$|Y| \geq |B \cap Y| = |B| - |B \cap X| \geq (1 - \epsilon_1)|B|.$$

Since each quartet $\{a, b, c, d\}$, where $a \in A \cap X$, $b \in A \cap Y$, $c \in B \cap X$, and $d \in B \cap Y$, yields a topology $ac|bd$ in T which is inconsistent with the bipartition $A|B$, we have

$$(n/9) \cdot (\epsilon_1 n) \cdot (\epsilon_1 n) \cdot (n/9) \leq \epsilon n^4 + O(n^3),$$

which implies that

$$\epsilon_1 \leq 9\sqrt{\epsilon}. \tag{3.1}$$

Suppose that the set Y is further partitioned into subsets Y_1 and Y_2, as illustrated in Figure 3.1. Without loss of generality, assume that $|B \cap Y_1| \leq |B \cap Y_2|$. Suppose that the constant ϵ (and thus the constant ϵ_1) is so small that

$$|B \cap Y_2| \geq (1 - \epsilon_1)|B| - |B \cap Y_1| \geq (1 - \epsilon_1)|B| - |B|/2 \geq |B|/3.$$

Thus, the edge e_1 inducing the partition $X \cup Y_1|Y_2$, as illustrated in Figure 3.1, is also an edge separator satisfying the conditions (i) and (ii) of the lemma. Then, we must have

$$|B \cap X| + |B \cap Y_1| \geq |A \cap Y|,$$

[3] There is certain analogy between this algorithm and the idea of using an *out-group* to root an evolutionary tree in biology.

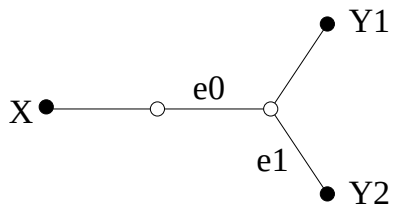


FIG. 3.1. *The edge separator e_0 and its induced bipartition.*

because otherwise the edge e_1 would have been a better choice than the edge e_0, violating the condition (iii) of the lemma. Since $|B \cap X| \leq \epsilon_1 n$,

$$|B \cap Y_1| \geq |A \cap Y| - \epsilon_1 n.$$

Since each quartet $\{a, b, c, d\}$, where $a \in A \cap X$, $b \in A \cap Y_1$, $c \in B \cap Y_1$, and $d \in B \cap Y_2$, yields a topology $ad|bc$ in T which is inconsistent with the bipartition $A|B$, we have

$$(n/9) \cdot |A \cap Y_1| \cdot (|A \cap Y| - \epsilon_1 n) \cdot (|A \cap Y| - \epsilon_1 n) \leq \epsilon n^4 + O(n^3).$$

Similarly, since each quartet $\{a, b, c, d\}$, where $a \in A \cap X$, $b \in A \cap Y_2$, $c \in B \cap Y_1$, and $d \in B \cap Y_2$, yields a topology $ac|bd$ in T which is inconsistent with the bipartition $A|B$,

$$(n/9) \cdot |A \cap Y_2| \cdot (|A \cap Y| - \epsilon_1 n) \cdot (|A \cap Y| - \epsilon_1 n) \leq \epsilon n^4 + O(n^3).$$

Summing up the above two inequalities, we have

$$(n/9) \cdot |A \cap Y| \cdot (|A \cap Y| - \epsilon_1 n)^2 \leq 2\epsilon n^4 + O(n^3).$$

Therefore,

$$\epsilon_2(\epsilon_2 - \epsilon_1)^2 \leq 18\epsilon.$$

It follows from (3.1) and the fact $\epsilon < 1$ that

$$\epsilon_2 < 11\epsilon^{1/3}. \qquad \square \tag{3.2}$$

Since the bipartition $A|B$ may still be incorrect (i.e., it may not be the bipartition induced by any edge of T), we try to revise it so it becomes eventually correct. From now on, let $\epsilon' = 11\epsilon^{1/3}$. We first try to detect the small number of pairs of labels that are "reversed" in the bipartition $A|B$. For any pair (a, b) of leaves, where $a \in A$ and $b \in B$, let us analyze how many quartet topologies $ax|by$ in the set Q, where $x \in A - \{a\}$ and $y \in B - \{b\}$, would "support" (i.e., be consistent with) the placement that the label a is in the set A and the label b is in the set B if (i) $a \in X$ and $b \in Y$ or (ii) $a \in Y$ and $b \in X$.

LEMMA 3.4. *Q has at least $((1-3\epsilon')^2 - \alpha/2)|X| \cdot |Y|$ quartet topologies supporting the placement $a \in A$ and $b \in B$ in case* (i) *(correctly), and at most $(12\epsilon' + \alpha/2)|X| \cdot |Y|$ quartet topologies supporting the placement $a \in A$ and $b \in B$ in case* (ii) *(incorrectly).*

Proof. Suppose that $a \in X$ and $b \in Y$. Let us first analyze how many quartet topologies $ax|by$ in the set Q_T, where $x \in A - \{a\}$ and $y \in B - \{b\}$, across the edge e_0 support the placement $a \in A$ and $b \in B$. Clearly, for any $x \in A \cap X - \{a\}$ and

$y \in B \cap Y - \{b\}$, the quartet topology $ax|by \in Q_T$ supports the placement. Hence the number of such supportive quartet topologies in Q_T across the edge e_0 is at least (roughly, omitting minor terms)

$$|A \cap X| \cdot |B \cap Y| \geq (1 - 3\epsilon')^2 |X| \cdot |Y|.$$

Since the set Q contains at most $\alpha|X| \cdot |Y|/2$ erroneous quartet topologies across the edge e_0, Q has at least

$$((1 - 3\epsilon')^2 - \alpha/2)|X| \cdot |Y|$$

quartet topologies supporting the placement $a \in A$ and $b \in B$.

The quartet topologies in Q supporting the placement $a \in A$ and $b \in B$ in case (ii) is at most

$$\begin{aligned} &|A| \cdot |B| - ((1 - 3\epsilon')^2 - \alpha/2)|X| \cdot |Y| \\ &\leq ((1 + 3\epsilon')^2 - (1 - 3\epsilon')^2 + \alpha/2)|X| \cdot |Y| \\ &= (12\epsilon' + \alpha/2)|X| \cdot |Y| \qquad \square \end{aligned}$$

Supposing that α is small enough and choosing ϵ sufficiently small, we can guarantee that the following inequality

$$\frac{12\epsilon' + \alpha/2}{(1 - 3\epsilon')^2 - \alpha/2} \leq \beta \tag{3.3}$$

holds for some small (but not too small) threshold β (exact value to be determined). Thus, from the set Q, we can decide if we should keep placing a in set A and b in set B, or switch them by checking the ratio between the supportive quartet topologies for each case.

We repeat the above test and correction until we cannot find any pair of labels to swap. Note that, in this process we may also swap pairs (a, b) with the property that $a \in A$, $b \in B$, and $a, b \in Y$ (or $a, b \in X$). That is, for such pairs we should (correctly) join them in the set Y (or X, respectively), but the quartet topologies in Q_T (and thus Q) tell us that they are reversed and we should swap them according to the above separation ratio. When this (e.g., $a, b \in Y$) happens, it must be the case that Y is bipartitioned into subsets Y_1 and Y_2 in the tree T, $a \in Y_1$, $b \in Y_2$, and $|Y_2| < \beta'|Y_1|$ for some constant β' depending on β, ϵ, and α. So, if we make sure that β is so small that $\beta' < 1/2$, then we won't switch such a pair (a, b) back and forth. Hence the process will converge in $O(n)$ swaps.

When the above process terminates, the bipartition $A|B$ may still not be consistent with any edge (separator) of T. But we must now have the property that either $X \subseteq A$ or $Y \subseteq B$, although we do not know which situation holds. So, in the following we try to further improve the bipartition $A|B$ assuming each situation separately.

Consider the case $X \subseteq A$ (the other case is symmetric). Let e be the edge of T whose induced bipartition, denoted $X'|Y'$, has the largest set X' that is completely contained in A. It is easy to see that $(1-3\epsilon')|Y'| \leq (1-3\epsilon')|Y| \leq |B \cap Y| \leq |B| \leq |Y'|$. Observe that e is in fact an edge separator. We will try to modify $A|B$ so it becomes the bipartition $X'|Y'$. Suppose that Y' is further bipartitioned into subsets Y_1' and Y_2' in the tree T, where $|Y_1'| \leq |Y_2'|$. By the choice of the edge e, $Y_1' \not\subseteq A$ and $Y_2' \not\subseteq A$. Moreover, from the above discussion on convergence, we know that if $|Y_1'| < \beta'|Y_2'|$ then $Y_2' \subseteq B$. Again, we take pairs of labels (a, b), where $a \in A$ and $b \in B$, and

analyze the support from Q for joining a and b in the set B if (i) $a \in X'$, or (ii) $a \in Y_i'$ and $b \in Y_i'$, for some i.

LEMMA 3.5. *Q contains at least* $(\min\{1/2 - 3\epsilon', \beta'/(1+\beta')\} - \alpha/2)|X'| \cdot |Y'|$ *supportive quartet topologies in case* (i) *and at most* $(6\epsilon' + \alpha/2)|X'| \cdot |Y'|$ *supportive quartet topologies in case* (ii).

Proof. Suppose that $a \in X'$. Q_T may contain a supportive quartet topology $ab|xy$, where $x \in A$ and $y \in B$, only if $x \in A \cap Y'$. Thus, the number of supportive quartet topologies in this case is at most

$$|Y' - Y' \cap B| \cdot |B| \leq 3\epsilon'|Y'|^2 \leq 6\epsilon'|X'| \cdot |Y'|.$$

Observe that, out of the $|X'|^2 \cdot |Y'|^2/4$ quartet topologies in Q_T across the edge e, only $\alpha|X'| \cdot |Y'|/2$ of them can go wrong in Q and become supportive; we claim Q contains at most

$$(6\epsilon' + \alpha/2)|X'| \cdot |Y'|$$

supportive quartet topologies.

Now suppose that $a \in Y_i'$ and $b \in Y_i'$ for some i. We consider two subcases. If $a \in Y_1'$ and $b \in Y_1'$, then all quartet topologies $ab|xy$ in Q_T, where $x \in A \cap X'$ and $y \in B \cap Y_2'$, would certainly be supportive. Hence, the number of supportive quartet topologies (across the edge connecting the sets $X \cup Y_2'$ and Y_1' in T) is at least

$$|X'| \cdot |Y_2'| \geq |X'| \cdot |Y'| \cdot (1/2 - 3\epsilon')$$

If $a \in Y_2'$ and $b \in Y_2'$, then $Y_2' \not\subseteq B$ and thus $|Y_1'| \geq \beta'|Y_2'|$. Since all quartet topologies $ab|xy$ in Q_T, where $x \in A \cap X'$ and $y \in B \cap Y_1'$, are supportive in this case, we have at least

$$|X'| \cdot |Y_1'| \geq |X'| \cdot \beta'|Y'|/(1+\beta')$$

supportive quartet topologies across the edge connecting the sets $X \cup Y_1'$ and Y_2' in T. Hence, Q contains at least

$$(\min\{1/2 - 3\epsilon', \beta'/(1+\beta')\} - \alpha/2)|X'| \cdot |Y'|$$

supportive quartet topologies in either subcase. ∎

Therefore, if we assume that α is sufficiently small, choose ϵ small enough and keep β' sufficiently large relative to α, then we can ensure that the inequality

$$\frac{6\epsilon' + \alpha/2}{\min\{1/2 - 3\epsilon', \beta'/(1+\beta')\} - \alpha/2} < \gamma \tag{3.4}$$

for some small threshold $\gamma < 1$. In other words, the set Q would contain sufficient information for us to tell if we should move a from the set A to the set B or not.

So we repeat the above step until (i) we cannot find any label to move or (ii) the size of A is getting below $|X| \geq n/9$. Observe that if we do not move anything at all in the whole process, then $A|B = X'|Y'$. In case (ii), we know that we have been moving in the wrong direction (i.e., $Y \subseteq B$ before the process started). In case (i), we could either get a correct bipartition $A|B = X'|Y'$ if $X \subseteq A$, or possibly an incorrect bipartition if $Y \subseteq B$. To check if the resulting bipartition is indeed one induced by some edge separator of T, all we need is to check if any pair (a, b), where $a \in A$ and

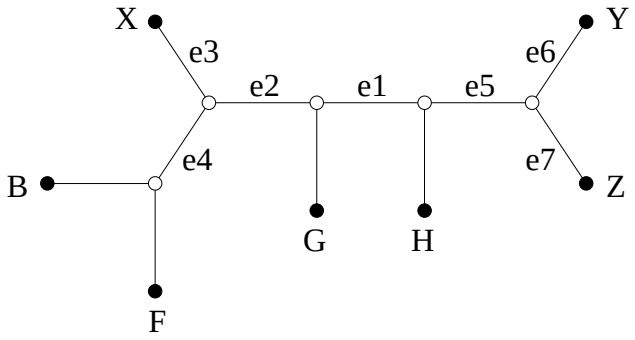


FIG. 3.2. *The placement of the labels $b \in B, x \in X, y \in Y$, and $z \in Z$ in T.*

$b \in B$, should be reversed under the criterion in inequality (3.3). Therefore, we can obtain at least one correct bipartition induced by some edge separator e' (which may or may not be e) of T, and clean up the erroneous quartets in Q across the edge e' safely.

• **The recursive step.** To continue cleaning up all errors in Q (or, equivalently, constructing the full T), we proceed in two stages. In the first stage, we recursively bipartition sets of sizes larger than $\sqrt{n}$. The general form of the problem is, given a correct bipartition $A|B$ of S, where $\sqrt{n} \leq |A| \leq 2n/3$, we try to find a bipartition of the set A consistent with T. The basic idea is the same as before, but we have to take into account the set B. Observe that since in the subtree of T induced by A, denoted as T_A, there can be at most $\alpha\sqrt{|A|^2 \cdot |B|^2} = \alpha|A| \cdot |B| = o(|A|^4)$ erroneous quartets across any edge of T_A, the approximation algorithm for MIBB will work on input Q_A (i.e., Q restricted to A), and will give us a good approximation of the bipartition induced by some edge separator of T_A as claimed in Lemma 3.3.

Let this approximate bipartition be $A_1|A_2$. We "hypothesize" if B should be placed in A_1 or A_2 by considering two bipartitions $A_1 \cup B|A_2$ and $A_1|A_2 \cup B$. For the bipartition $A_1 \cup B|A_2$, we swap and join pairs (a, b), where $a \in A_1$ and $b \in A_2$, as before, but by considering quartets $\{a, b, x, y\}$ of the special form $x \in B$ and $y \in A_2$. Similarly, for the bipartition $A_1|A_2 \cup B$, we swap and join pairs (a, b), where $a \in A_1$ and $b \in A_2$, by considering quartets $\{a, b, x, y\}$ of the form $x \in A_1$ and $y \in B$. It is straightforward to show that Lemmas 3.4 and 3.5 can be easily extended to work for these bipartitions. At the end we can again check which of the bipartitions yields a correct bipartition consistent with some edge separator of T_A as before.

• **Terminating the recursion.** We use the above recursive step to produce correct bipartitions $A|B$, where $|A| \leq \sqrt{n}$. Now we cannot continue the recursion since the errors in Q may jeopardize the performance of the approximation algorithm for MIBB when $|A|$ gets below $n^{1/3}$. So, we turn to a more direct cleaning approach which makes essential use of the large size of the set B.

Consider a quartet $\{b, x, y, z\}$, where $x, y, z \in A$ and $b \in B$. Suppose that the labels are placed in the tree T as illustrated in Figure 3.2.

Let's consider all quartets consisting of a label $b' \in B$, two of the labels x, y, z, and a label $a \in A - \{x, y, z\}$, and analyze the supports from the sets Q_T and Q for each of the three possible topologies $bx|yz$, $by|xz$, and $bz|xy$. For each edge e, let $f_Q(e)$ denote the number of errors in Q across edge e.

LEMMA 3.6. *The set Q contains at least*

$$\begin{aligned}&(1-6\alpha/2)|B|\cdot(|H|+|G|+|Y|+|Z|)-f_Q(e_3)-f_Q(e_4)\\&\leq(1-7\alpha/2)|B|\cdot(|H|+|G|+|Y|+|Z|)-\alpha|B|\cdot|X|\end{aligned}$$

more supportive quartet topologies for $bx|yz$ than for $by|xz$ or for $bz|xy$.

Proof. We first calculate the number of such quartet topologies in Q_T across an (arbitrarily chosen) edge e_1 as illustrated in the figure that support the correct topology $bx|yz$. A quartet $\{b', x, y, a\}$ would yield a supportive topology across e_1 if $a \in H \cup Z \cup Y - \{y\}$. So Q_T has $|B|\cdot(|H|+|Y|+|Z|-1)$ such supportive quartet topologies. Similarly, we know that Q_T has $|B|\cdot(|H|+|Y|+|Z|-1)$ supportive quartet topologies across e_1 of the form $b'x|az$ and $|B|\cdot(|F|+|G|+|X|)$ supportive quartet topologies across e_1 of the form $b'a|yz$. Hence, Q_T has a total of

$$|B|\cdot(2|H|+2|Y|+2|Z|+|F|+|G|+|X|-2)$$

supportive quartet topologies across the edge e_1. This implies that Q has at least

$$\begin{aligned}&|B|\cdot(2|H|+2|Y|+2|Z|+|F|+|G|+|X|-2)\\&-(\alpha/4)(|B|+|X|+|F|+|G|)\cdot(|Y|+|Z|+|H|)\\&\geq\ |B|\cdot(2|H|+2|Y|+2|Z|+|F|+|G|+|X|-\alpha(|Y|+|Z|+|H|)/4)\end{aligned}$$

supportive quartet topologies.

Let's calculate the number of quartet topologies in Q_T supporting the topology $by|xz$. A quartet $\{b', x, y, a\}$ would yield a supportive topology only if $a \in X - \{x\}$, and thus Q_T has $|B|\cdot(|X|-1)$ such supportive quartet topologies. Similarly, Q_T has $|B|\cdot|F|$ supportive quartet topologies of the form $b'a|xz$ and $|B|\cdot(|Z|-1)$ supportive quartet topologies of the form $b'y|az$. Hence, Q_T has a total of

$$|B|\cdot(|X|+|Z|+|F|-2)$$

supportive quartet topologies. This implies that Q has at most

$$|B|\cdot(|X|+|Z|+|F|-2)+\sum_{i=1}^{7}f_Q(e_i)$$

supportive quartet topologies, where $f_Q(e_i) \leq \alpha\sqrt{|Q_T(e_i)|}$ denotes the number of errors across the edge e_i in the current set Q.

Using the same idea, we claim that the set Q has at most

$$|B|\cdot(|X|+|Y|+|F|-2)+\sum_{i=1}^{7}f_Q(e_i)$$

members supporting the topology $bz|xy$.

So the difference between the support for the correct topology $bx|yz$ and that for incorrect ones $by|xz$ or $bz|xy$ is at least

$$\begin{aligned}&(1-\alpha/4)|B|\cdot(2|H|+|G|+|Y|+|Z|)-\sum_{i=1}^{7}f_Q(e_i)\\&\leq(1-\alpha/4)|B|\cdot(|H|+|G|+|Y|+|Z|)\end{aligned}$$

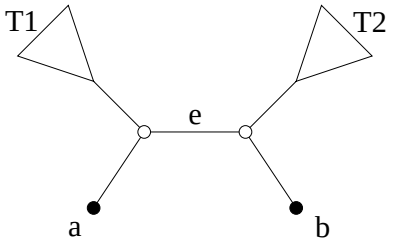


FIG. 3.3. *A hard case for quartet cleaning.*

$$\begin{aligned}
&- 5 \cdot (\alpha/4)|B| \cdot (|H| + |G| + |Y| + |Z|) - f_Q(e_3) - f_Q(e_4) \\
&\leq (1 - 6\alpha/4)|B| \cdot (|H| + |G| + |Y| + |Z|) - f_Q(e_3) - f_Q(e_4) \\
&\leq (1 - 6\alpha/4)|B| \cdot (|H| + |G| + |Y| + |Z|) \\
&\quad - \alpha|B| \cdot |X|/4 - \alpha|B| \cdot (|X| + |H| + |G| + |Y| + |Z|)/4 \\
&\leq (1 - 7\alpha/4)|B| \cdot (|H| + |G| + |Y| + |Z|) - \alpha|B| \cdot |X|/2. \qquad \square
\end{aligned}$$

The above difference between the supports is at least $-\alpha|B| \cdot |X|/2$ and would in fact be at least $(1 - 9\alpha/2)|B| \cdot (|H| + |G| + |Y| + |Z|)$ if (i) $|X| \leq |H| + |G| + |Y| + |Z|$ or (ii) the erroneous quartet topologies in Q across the edges e_3 and e_4 have already been fixed. This suggests that we should first work on the quartet $\{b, x, y, z\}$ which yields the largest difference.

More precisely, our algorithm finds the quartet $\{b, x, y, z\}$, where $x, y, z \in A$ and $b \in B$, with the largest margin in the supports from Q for each of its three possible topologies, and correct Q according to the topology with the highest support. We then consider the remaining quartets $\{b, x, y, z\}$ of the form $x, y, z \in A$ and $b \in B$, and repeat the same operation until correct topologies for all such quartets have been found.

3.3. An upper bound for quartet cleaning. The following theorem establishes an upper bound on the number of quartet errors across an edge that can be corrected.

THEOREM 3.7. *No algorithm can correctly reconstruct the evolutionary tree T if Q contains $\sqrt{|Q_T(e)|}$ or more erroneous quartet topologies across some edge e of T.*

Proof. Consider the tree in Figure 3.3 and quartets of the form $\{a, b, x, y\}$ where $x \in T_1$ and $y \in T_2$. If half of these quartets have topology $ax|by$ in Q whereas the other half have topology $ay|bx$ in Q then it cannot be decided which of these quartet topologies are erroneous. It follows that there must be no more that $|T_1||T_2|/2$ quartet errors across e. The theorem follows from

$$|T_1||T_2|/2 \leq \sqrt{|Q_T(e)|}. \qquad \square$$

4. Complete MQC is NP-complete. In this section we demonstrate that the following problem is NP-complete.

MAXIMUM QUARTET CONSISTENCY (DECISION) (MQCD).
Instance: A complete set Q of quartet topologies over label set S and nonnegative integer k.
Question: Is there an evolutionary tree T labeled by S such that $|Q_T \cap Q| \geq k$?

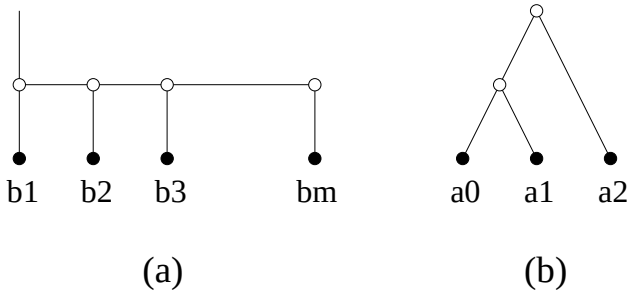


FIG. 4.1. (a) *The caterpillar tree M with $m \gg n^5$ feet (leaves).* (b) *Each label $a \in S$ is replaced by three $a_0, a_1, a_2 \in S'$.*

It is clear that MQC is in NP. To complete the proof we reduce Quartet Compatibility to MQC.

QUARTET COMPATIBILITY.
Instance: A (possibly incomplete) set Q of quartet topologies on label set S.
Question: Is Q compatible, i.e., does there exist an evolutionary tree T labelled by S such that $Q \subseteq Q_T$?

This problem is known to be NP-hard [13]. Given an instance of **Quartet Compatibility** with a quartet topology set Q defined on a label set S, where $n = |S|$, we will construct an instance of MQC with a *complete* quartet topology set Q' defined on a new label set S' such that there is a evolutionary tree T realizing Q with no error if and only if there is a evolutionary tree T' realizing Q' with at most $g(n)m + f(n)$ errors, where $f(n) = O(n^4)$, $g(n) = O(n^3)$ and $m \gg n^5$ will be specified later.

The basic idea behind this proof is to add topologies to Q for quartets that are not specified in Q to create a complete set Q' such that with respect to any optimal evolutionary tree T' for Q', precisely one third of the added quartet topologies are correct. In order to do this, we will need a large evolutionary tree M on m leaves with a fixed (e.g., caterpillar) structure (see Figure 4.1 (a)), where m is a number that is both divisible by three and much larger than the number of missing quartet topologies in Q, e.g., $m \gg n^5$. M will be embedded as a subtree in the optimal evolutionary tree T' and will be used to enforce certain useful structures in T'.

We construct the sets S' and Q' as follows. Add the m leaf-labels $b_1, \ldots, b_m$ in M to S', and for each $a \in S$, create three new labels a_0, a_1, a_2 and add them to S'. Note that we do not add the original labels in S to S'. We want to specify Q' such that (i) each triplet of labels a_0, a_1, a_2 appear together in a subtree of T' as in Figure 4.1(b) and (ii) the optimal evolutionary tree T' for Q' is formed by attaching M to some branch of a evolutionary tree $\hat{T}$ that is obtained from an optimal evolutionary tree T for Q by replacing every leaf of T with a subtree containing three leaves as shown in Figure 4.1(b).

Intuitively, Q' must be constructed to enforce the following conditions:

- M appears intact in T' as in Figure 4.1(a).
- Each created triplet of labels a_0, a_1, a_2 appear together in a subtree of T' as in Figure 4.1(b), with nothing else inserted between them.
- The quartet topologies in Q extend naturally to Q'. That is, if $ab|cd \in Q$, then $a_i b_j | c_k d_l$ is included in Q' for all $0 \le i, j, k, l \le 2$.
- For each quartet involving one label from M and three labels corresponding to three distinct elements of S, its topology is related to the (unknown) structure

of T (or $\hat{T}$) and the branch of $\hat{T}$ where M is attached. Hence, we should make sure that the number of erroneous topologies induced from such quartets is independent of the structure of T and the location where M is attached.

- For all quartets on S' that correspond to labels of Q with missing topologies in Q, we add quartet topologies in Q' such that precisely one third of these new quartet topologies are satisfied in T'. This is the difficult part since we do not know the structure of T'.

Here are the details of the construction. Let $w, x, y, z \in S'$ be four distinct labels.

1. If the labels are all in M, specify the quartet topology according to the structure of M as shown in Figure 4.1(a).
2. If $w = a_i$ for some $a \in S$, $x = b_j, y = b_k$, and $z = b_l$ with $j < k < l$, specify the topology as $wx|yz$.
3. If $w, x \in M$ and $y, z \notin M$, specify the topology as $wx|yz$.
4. If $w = b_i \in M$ and $x, y, z \notin M$, we consider two subcases.
   (a) If at least two of x, y, z correspond to the same label in S, then the quartet topology can be specified according to the required structure of T' described above.
   (b) If x, y, z correspond to distinct labels in S, then specify the topology as $b_i x|yz$ if $i \leq m/3$, or as $b_i y|xz$ if $m/3 < i \leq 2m/3$, or as $b_i z|xy$ if $i > 2m/3$. Intuitively, here we are partitioning M into three subsets $\{b_1, \ldots, b_{m/3}\}$, $\{b_{m/3+1}, \ldots, b_{2m/3}\}$, and $\{b_{2m/3+1}, \ldots, b_m\}$ so that quartets of the above form will introduce the same number of errors no matter how T looks and where M is attached in $\hat{T}$.
5. Finally, if none of the labels are from M, we consider three subcases.
   (a) If at least two of them correspond to the same label in S, then this quartet topology can be specified as before.
   (b) If they correspond to a quartet on S that has a resolved topology in Q, then specify the same topology in Q'.
   (c) If they correspond to a quartet on S whose topology is missing in Q, we take care of all such quartets collectively. Recall that each such quartet of labels $w, x, y, z \in S$ corresponds to $3^4 = 81$ different quartets on S'. We divide them into $81/3 = 27$ disjoint *groups*. Each group contains three quartets:

$$\begin{gathered} w_0, x_i, y_j, z_k, \\ w_1, x_{i+1 \bmod 3}, y_{j+1 \bmod 3}, z_{k+1 \bmod 3}, \\ w_2, x_{i+2 \bmod 3}, y_{j+2 \bmod 3}, z_{k+2 \bmod 3}. \end{gathered}$$

   For each group, we specify quartet topologies as follows:

$$\begin{gathered} w_0 x_i | y_j z_k, \\ w_1 y_{j+1 \bmod 3} | x_{i+1 \bmod 3} z_{k+1 \bmod 3}, \\ w_2 z_{k+2 \bmod 3} | x_{i+2 \bmod 3} y_{j+2 \bmod 3}. \end{gathered}$$

Obviously, the quartet topologies defined in items 1, 2, 3, 4(a), 5(a), and 5(b) do not introduce any errors in T' if M stays as shown in Figure 4.1(a) and for each label $a \in S$, its associated triplet of labels $a_0, a_1, a_2 \in S'$ appear together in a subtree of T' as shown in Figure 4.1(b). The following lemmas give exact error bounds for quartet topologies defined in items 4(b) and 5(c).

LEMMA 4.1. *If M does not split, item* 4(b) *introduces precisely $g(n)m$ quartet errors, where $g(n) = 18\binom{n}{3}$.*

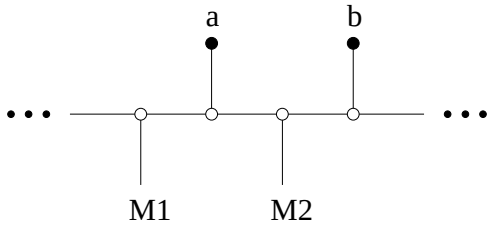


FIG. 4.2. *M is split by label a.*

Proof. The number of quartets considered in item 4(b) is $\binom{n}{3} \cdot 3^3 \cdot \frac{2m}{3} = 18\binom{n}{3}m = g(n)m$. ∎

LEMMA 4.2. *Let $q(n) = \binom{n}{4} - |Q|$ be the number of missing quartet topologies in Q. If M does not split, then for each $a \in S$, its associated triplet of labels $a_0, a_1, a_2 \in S'$ appear together in a subtree of T' as shown in Figure* 4.1(b) *and item* 5(c) *introduces precisely $f(n) = 54q(n)$ quartet errors.*

Proof. Suppose that for some $a \in S$, the triplet $a_0, a_1, a_2 \in S'$ do not appear together in a subtree as shown in Figure 4.1(b). Assume for the worst case that there is a single leaf x separating these three elements in T' as shown below: Then, for any label $y \in M$, the quartet topology $a_0x|a_1y$ in Q' would be an error. As there are at least m such y, this gives rise to at least m quartet errors. Since $m \gg n^5$, we can improve T' by moving x away from the triplet a_0, a_1, a_2, which contradicts to the fact that T' is optimal for Q'.

Given that each triplet of labels corresponding to a label in L appear together in a subtree in T' as shown in Figure 4.1(b), precisely one third of the 81 quartet topologies specified in item 5(c) for each missing quartet in Q are correct. Hence, item 5(c) introduces precisely $81 \cdot \frac{2}{3}q(n) = 54q(n) = f(n)$ quartet errors. ∎

Finally, we have to show that M does not split in the optimal evolutionary tree T'.

LEMMA 4.3. *In the optimal evolutionary tree T' realizing Q', M stays intact as the caterpillar shown in Figure* 4.1(a).

Proof. Splitting M may reduce the error terms in Lemma 4.1 and Lemma 4.2. We argue that such splittings are not worthwhile because they introduce more quartet errors to T' than they can save.

To illustrate the idea of the argument, suppose that some labels $a, b \notin M$ split M into two subsets M_1 and M_2, as shown in Figure 4.2.

In this case, we may (or may not) save at most $g(n) \cdot \min\{|M_1|, |M_2|\}$ errors from item 4 that. But this splitting creates at least $\Omega(|M_1| \cdot |M_2|)$ new errors because all quartet topologies of the form

$$m_1a|m_2b,$$

where $m_1 \in M_1, m_2 \in M_2$, are erroneous. Since $m \gg n^5$ and $g(n) = O(n^3)$, it is easy to see that $|M_1| \cdot |M_2| \gg g(n) \cdot \min\{|M_1|, |M_2|\}$. Hence, the splitting in fact creates more errors than it can save.

Therefore, in the optimal evolutionary tree T', M stays in one piece as a caterpillar subtree. ∎

From the above lemmas, we conclude that Q is compatible if and only if there exists an evolutionary tree T' that is inconsistent with Q' on at most (in fact, exactly) $g(n)m + f(n)$ quartets.

5. Discussion. In practice, the inference of quartet topology is not reliable, and so, confidence levels are assigned to quartet topologies. For example, for quartet $\{a, b, c, d\}$ the quartet topologies $ab|cd$, $ac|bd$, and $ad|bc$ may be assigned confidence levels 80%, 15%, and 5% indicating that we have the most confidence in the inference $ab|cd$ but that this confidence is not 100%. Given this information, the weighted MQC problem is to obtain an evolutionary tree T that maximizes

$$\sum_{ab|cd \in Q_T} w(ab|cd),$$

where $w(ab|cd)$ denotes the confidence level of quartet topology $ab|cd$. The MQC PTAS can be extended to solve this weighted variation on MQC as long as the weights are drawn from some interval of positive integers of constant range to preserve the smoothness of the polynomial integer programs. On the other hand, when the weights are allowed to be 0-1, weighted MQC becomes the incomplete MQC problem which is MAX-SNP–hard.

The PTAS for weighted MQC can also be used to solve the quartet consensus problem [6]. In the quartet consensus problem, several evolutionary trees T_1, T_2, ..., T_k compete as alternate hypotheses for the evolutionary history of a label set S. The goal is to produce an evolutionary tree T that maximizes the sum

$$\sum_{i=1}^{k} |Q_T \cap Q_{T_i}|.$$

When k is a constant this can be solved by defining $w(ab|cd)$ to be the number of evolutionary trees T_i in which $ab|cd$ is induced, for each quartet topology $ab|cd$, and then applying the weighted MQC PTAS.

In an error model that restricts the number of quartet errors across an edge, each quartet error may be "charged" to many edges. For example, in Figure 1.4(ii), a quartet error involving the labels a, b, c, and d would be charged to all edges on the path p connecting a, b with c, d. Hence, it is also natural to associate quartet errors with *paths* instead of edges. If p is a path in T then $\{a, b, c, d\}$ is a quartet across p if p contains all the edges crossed by the quartet $\{a, b, c, d\}$, as illustrated in Figure 1.4(ii). Error models that restrict the number of quartet errors across paths and those that restrict the number of quartet errors across edges are incomparable. In general, the former are good at capturing uniformly distributed errors while the latter are better suited for describing localized errors. Let $Q_T(p)$ denote the set of quartets across a path p of T. It is not hard to extend our bipartition-based quartet cleaning algorithm to work under the assumption that Q contains at most $\alpha\sqrt{|Q_T(p)|}$ quartet errors across any path p of T, for some constant $\alpha > 0$.

Several open problems present themselves. In particular, the quartet cleaning technique presented here is based upon an error model that bounds the number of quartet errors across every edge of the evolutionary tree. A method that could clean quartet errors across edges independently would be an improvement. It is hopeless to obtain a PTAS for the sparse MQC problem since it is MAX-SNP–hard. Can we obtain a better than 1/3 approximation for the sparse MQC problem?

Acknowledgment. We would like to thank the anonymous referees for their detailed comments.

REFERENCES

[1] S. Arora, A. Frieze, and H. Kaplan, *A new rounding procedure for the assignment problem with applications to dense graph arrangement problems*, in 37th Annual Symposium on Foundations of Computer Science, IEEE Computer Society Press, 1996, pp. 21–30.

[2] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy, *Proof verification and hardness of approximation problems*, in Proceedings of the 33rd IEEE Symposium on the Foundations of Computer Science, IEEE Computer Society Press, 1992, pp. 14–23.

[3] A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg, *From four–taxon trees to phylogenies: The case of mammalian evolution*, in Proceedings of the 2nd Annual International Conference on Computational Molecular Biology, ACM Press, 1998, pp. 9–19.

[4] V. Berry, *private communication*, LIRM, Montpellier, France, 1998.

[5] V. Berry and O. Gascuel, *Inferring evolutionary trees with strong combinatorial evidence*, in Proceedings of the 3rd Annual International Computing and Combinatorics Conference, Springer, 1997, pp. 111–123.

[6] D. Bryant, *Structures in Biological Classification*, Ph.D. thesis, Department of Mathematics and Statistics, University of Canterbury, Canterbury, UK, 1997.

[7] P. Buneman, *The recovery of trees from measures of dissimilarity*, in Mathematics in the Archaeological and Historical Sciences, F. R. Hodson, D. G. Kendall, and P. Tautu, eds., Edinburgh University Press, Edinburgh, 1971, pp. 387–395.

[8] P. Erdös, M. Steel, L. Székely, and T. Warnow, *Constructing big trees from short sequences*, in Proceedings of the 24th International Colloquium on Automata, Languages, and Programming, Springer, 1997.

[9] J. Felsenstein, *Evolutionary trees from DNA sequences: A maximum likelihood approach*, J. Molecular Evolution, 17 (1981), pp. 368–376.

[10] J. Felsenstein, *Numerical methods for inferring evolutionary trees*, Quarterly Review of Biology, 57 (1982), pp. 379–404.

[11] P. E. Kearney, *The ordinal quartet method*, in Proceedings of the 2nd Annual International Conference on Computational Molecular Biology, ACM Press, 1998, pp. 125–134.

[12] N. Saitou and M. Nei, *The neighbor–joining method: A new method for reconstructing phylogenetic trees*, Molecular Biology and Evolution, 4 (1987), pp. 406–425.

[13] M. Steel, *The complexity of reconstructing trees from qualitative characters and subtrees*, J. Classification, 9 (1992), pp. 91–116.

[14] K. Strimmer and A. von Haeseler, *Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies*, Molecular Biology and Evolution, 13 (1996), pp. 964–969.

[15] D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M Hillis, *Phylogenetic inference*, in Molecular Systematics, 2nd ed., D. M. Hillis, C. Moritz, and B. K. Mable, eds., Sinauer Associates, Sunderland, MA, 1996, pp. 407–514.

[16] A. Templeton, *Human origins and analysis of mitochondrial DNA sequences*, Science, 255 (1991), p. 737.

[17] L. Vigilant, M. Stoneking, H. Harpending, H. Hawkes, and A. C. Wilson, *African populations and the evolution of human mitochondrial DNA*, Science, 253 (1991), pp. 1503–1507.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1962–1975

AN EXTENSION OF PATH COUPLING AND ITS APPLICATION TO THE GLAUBER DYNAMICS FOR GRAPH COLORINGS*

MARTIN DYER†, LESLIE ANN GOLDBERG‡, CATHERINE GREENHILL§, MARK JERRUM¶, AND MICHAEL MITZENMACHER‖

Abstract. A new method for analyzing the mixing time of Markov chains is described. This method is an extension of path coupling and involves analyzing the coupling over multiple steps. The expected behavior of the coupling at a certain stopping time is used to bound the expected behavior of the coupling after a fixed number of steps. The new method is applied to analyze the mixing time of the Glauber dynamics for graph colorings. We show that the Glauber dynamics has $O(n \log(n))$ mixing time for triangle-free Δ-regular graphs if k colors are used, where $k \geq (2 - \eta)\Delta$, for some small positive constant η. This is the first proof of an optimal upper bound for the mixing time of the Glauber dynamics for some values of k in the range $k \leq 2\Delta$.

Key words. Markov chains, coupling, stopping times, graph coloring, Glauber dynamics

AMS subject classifications. 05C15, 05C85, 60J10, 68Q25, 82B20

PII. S0097539700372708

1. Introduction.

In this paper, a new method for analyzing the mixing time of Markov chains is described. This method is a nontrivial extension of path coupling, and applies in situations where path coupling is not enough to prove rapid mixing. We run the path coupling for multiple steps and use the expected behavior of the coupling at a certain *stopping time* to bound the expected behavior of the coupling after a fixed number of steps. Standard path coupling is a worst-case analysis in that it considers the expected change in the distance between the worst possible pair of states over a single step. However, in a multiple-step analysis, the choice of the initial pair of states is mitigated by the random choices made by the coupling over several steps. Hence, with some constant probability, we are not in the worst case. This is how a multiple-step analysis can improve upon one-step path coupling.

The approach of analyzing the behavior of a Markov chain over several steps has proved worthwhile in other settings. For example, it has been used to prove the stability of randomized bin-packing algorithms [7, 16, 1] and contention resolution protocols [14, 13]. Hence this approach appears to be a natural direction for coupling arguments as well.

*Received by the editors May 31, 2000; accepted for publication (in revised form) October 30, 2000; published electronically March 20, 2001. An earlier version of this paper appeared in the Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, 2000. This work was supported in part by the EPSRC Research grant "Sharper Analysis of Randomised Algorithms: A Computational Approach" and by the ESPRIT Projects RAND-APX and ALCOM-FT.

http://www.siam.org/journals/sicomp/30-6/37270.html

†School of Computer Studies, University of Leeds, Leeds LS2 9JT, UK (dyer@scs.leeds.ac.uk).

‡Department of Computer Science, University of Warwick, Coventry CV4 7AL, UK (leslie@dcs.warwick.ac.uk).

§Department of Mathematics and Statistics, University of Melbourne, Parkville, Vic 3052, Australia (csg@ms.unimelb.edu.au). This author was supported by an Australian Research Council Postdoctoral Fellowship.

¶Department of Computer Science, University of Edinburgh, Edinburgh ED9 3JZ, UK (mrj@dcs.ed.ac.uk).

‖Computer Science Department, Harvard University, 33 Oxford St, Cambridge, MA 02138 (michaelm@eecs.harvard.edu). This author was supported in part by the Alfred P. Sloan Foundation.

Czumaj et al. [8] introduced a framework for multiple-step couplings based on path coupling, which they call *delayed path coupling*. Their "delayed path coupling lemma" [8, Lemma 4.2] (reproduced below as Lemma 2.2) shows how the mixing time of a Markov chain can be bounded above in terms of the behavior of a coupling over a fixed number of steps. However, the way in which the coupling is analyzed over the fixed time interval is not specified, and Czumaj et al. give a few different applications. In some applications, they explicitly construct a non-Markovian coupling over the full time interval. The construction and analysis of such a coupling can be very complicated. However, we use straightforward path coupling to drive our multiple-step coupling, performing most of our analysis at a specially defined stopping time. The next section contains a description of this new method.

We then apply our method to the problem of analyzing the mixing time of the Glauber dynamics for graph colorings. A proper k-coloring of a graph $G = (V, E)$ is a labelling of the vertices from a set of colors $C = \{1, \ldots, k\}$ such that no two neighboring vertices have the same color. We consider the problem of sampling nearly uniformly from the set of all proper k-colorings of a graph of maximum degree Δ. Note that efficiently sampling k-colorings nearly uniformly allows one to approximately count such colorings [15]. This problem is interesting as a fundamental combinatorial problem, and it also relates to several problems in statistical physics; see [15, 23] for more details.

A standard approach to the sampling problem is to design a Markov chain whose stationary distribution is uniform over all proper k-colorings. We can then sample nearly uniformly from all proper k-colorings by running the Markov chain until the distribution of the state is sufficiently near the stationary distribution. For this approach to be efficient, the number of steps for which we must run the Markov chain must be sufficiently small. The number of steps for which we must run the Markov chain is generally called the *mixing time*, and a Markov chain for sampling proper k-colorings is *rapidly mixing* if the mixing time is bounded above by some polynomial in $|V| = n$.

Jerrum [15] (and independently Salas and Sokal [21], using different methods) showed that when $k \geq 2\Delta$, a simple Markov chain is rapidly mixing. This Markov chain is easily described as follows: choose a vertex v uniformly at random and a color c uniformly at random; recolor v to color c if doing so yields a proper coloring. This Markov chain is generally referred to as the Glauber dynamics in the statistical physics literature. Jerrum proved that the Glauber dynamics has $O(n \log(n))$ mixing time for $k > 2\Delta$, while for $k = 2\Delta$ the best known upper bound was $O(n^3)$. We use our new method to show that, for $\Delta \geq 14$, the Glauber dynamics chain has $O(n \log(n))$ mixing time for $k \geq (2-\eta)\Delta$ whenever the graph is triangle-free and Δ-regular, where η is some small, positive constant. It seems to be widely believed that $\Omega(n \log n)$ is a lower bound on the mixing time of the Glauber dynamics; however, we do not know of an existing proof. We present a simple proof of this fact in Theorem 3.1, for the special case of graphs with no edges. Therefore our $O(n \log n)$ bound on the mixing time is optimal. Our main result is the first proof of an optimal upper bound for the mixing time of the Glauber dynamics for some values of k in the range $k \leq 2\Delta$.

The 2Δ barrier has been broken using more complicated chains, but as far as we know this is the first proof that involves direct analysis of the simple Glauber dynamics chain. In [5], a rapidly mixing Markov chain was presented for the case $\Delta = 3$, $k = 5$ (and for $\Delta = 4$, $k = 7$ when the graph is triangle-free and 4-regular). The proof involves the analysis of several (in the hundreds for the $\Delta = 3$ case)

linear programming problems related to the chain. Using a comparison technique such as [6, 9, 10, 11, 20] one can conclude that the Glauber dynamics is also rapidly mixing for these values of k, Δ. However, applying a comparison technique generally increases the upper bound on the mixing time by several factors of n.

In recent work, Vigoda [23] has proven that $k \geq 11\Delta/6$ is sufficient for rapid mixing, using an entirely different Markov chain (similar to the well-known Swendsen–Wang algorithm [22]). Again, his result implies rapid mixing of the Glauber dynamics for $k \geq 11\Delta/6$, but with an $O(n^2 \log n)$ bound on the mixing time. His result clearly dominates ours in terms of the range of k for which rapid mixing is established. However, because our analysis is based directly on the Glauber dynamics chain and achieves an optimal bound, and because we use a new technique based on analyzing this chain over multiple steps, our result is of independent interest.

2. Path coupling using stopping times. Before describing the new method we present some standard definitions and notation. Let Ω be a finite set and let $\mathcal{M}$ be a Markov chain with state space Ω, transition matrix P, and unique stationary distribution π. If the initial state of the Markov chain is x then the distribution of the chain at time t is given by $P_x^t(y) = P^t(x, y)$. The *total variation distance* of the Markov chain from π at time t, with initial state x, is defined by

$$\mathrm{d_{TV}}(P_x^t, \pi) = \frac{1}{2} \sum_{y \in \Omega} |P^t(x, y) - \pi(y)|.$$

Following Aldous [3], let $\tau_x(\varepsilon)$ denote the least value T such that $\mathrm{d_{TV}}(P_x^t, \pi) \leq \varepsilon$ for all $t \geq T$. The *mixing time* of $\mathcal{M}$, denoted by $\tau(\varepsilon)$, is defined by $\tau(\varepsilon) = \max\{\tau_x(\varepsilon) : x \in \Omega\}$. A Markov chain is said to be *rapidly mixing* if the mixing time is bounded above by some polynomial in n and $\log(\varepsilon^{-1})$, where n is a measure of the size of the elements of Ω. Throughout this paper all logarithms are to base e.

There are relatively few methods available to prove that a Markov chain is rapidly mixing. One such method is *coupling*. A coupling for $\mathcal{M}$ is a stochastic process (X_t, Y_t) on $\Omega \times \Omega$ such that each of (X_t), (Y_t), considered marginally, is a faithful copy of $\mathcal{M}$. The moves of the coupling are correlated to encourage the two copies of the Markov chain to *couple*: i.e., to achieve $X_t = Y_t$. This gives a bound on the total variation distance using the *coupling lemma* (see for example, Aldous [3]), which states that

$$\mathrm{d_{TV}}(P_x^t, \pi) \leq \mathrm{Prob}[X_t \neq Y_t],$$

where $X_0 = x$ and Y_0 is drawn from the stationary distribution π. The following standard result is used to obtain an upper bound on this probability and hence an upper bound for the mixing time (the proof is omitted).

THEOREM 2.1. *Let (X_t, Y_t) be a coupling for the Markov chain $\mathcal{M}$ and let ρ be any integer valued metric defined on $\Omega \times \Omega$. Suppose that there exists $\beta \leq 1$ such that $\mathbf{E}[\rho(X_{t+1}, Y_{t+1})] \leq \beta\, \rho(X_t, Y_t)$ for all t, and all $(X_t, Y_t) \in \Omega \times \Omega$. Let D be the maximum value that ρ achieves on $\Omega \times \Omega$. If $\beta < 1$ then the mixing time $\tau(\varepsilon)$ of $\mathcal{M}$ satisfies $\tau(\varepsilon) \leq \log(D\varepsilon^{-1})/(1 - \beta)$. If $\beta = 1$ and there exists $\alpha > 0$ such that*

$$\mathrm{Prob}[\rho(X_{t+1}, Y_{t+1}) \neq \rho(X_t, Y_t)] \geq \alpha$$

for all t, and all $(X_t, Y_t) \in \Omega \times \Omega$, then $\tau(\varepsilon) \leq \lceil eD^2/\alpha \rceil \lceil \log(\varepsilon^{-1}) \rceil$.

From now on, assume that all couplings are Markovian unless explicitly stated. The path coupling method, introduced in [4], is a variation of traditional coupling

which allows us to restrict our attention to a certain subset S of $\Omega \times \Omega$, where Ω is the state space of a given Markov chain. If we view S as a relation, the transitive closure of S must equal Ω. The rate of convergence of the chain is measured with respect to a (quasi)metric ρ on $\Omega \times \Omega$, which can be defined by lifting a proximity function on S to the whole of $\Omega \times \Omega$ (see [12] for details).

In this section we present a modification of path coupling which involves stopping times. Let (X, Y) be any element of $\Omega \times \Omega$. As for ordinary path coupling, we define a path, or sequence

$$X = Z_0,\ Z_1, \ldots, Z_r = Y$$

between X and Y, where $(Z_\ell, Z_{\ell+1}) \in S$ for $0 \le \ell < r$, and

$$\sum_{\ell=0}^{r-1} \rho(Z_\ell, Z_{\ell+1}) = \rho(X, Y).$$

In ordinary path coupling we allow the coupling to evolve for *one* step, giving a new path

$$Z_0',\ Z_1', \ldots, Z_r'$$

(for a precise definition of the probability distribution of this new path, see [12]). We then define $(X', Y') = \left(Z_0', Z_r'\right)$. The path coupling lemma says the following. Let $(X, Y) \mapsto (X', Y')$ be a coupling defined on all pairs in S. Suppose there exists a constant β such that $0 < \beta \le 1$ and for all $(X, Y) \in S$ we have

$$\mathbf{E}\left[\rho(X', Y')\right] \le \beta\, \rho(X, Y). \tag{2.1}$$

Then we can conclude that (2.1) holds for all $(X, Y) \in \Omega \times \Omega$, and apply Theorem 2.1. Suppose, however, that the smallest value of β for which (2.1) holds for all $(X, Y) \in S$ satisfies $\beta > 1$. Then path coupling is not good enough to allow us to apply Theorem 2.1. However, if β is not much larger than 1, and there are some "good" initial pairs $(X, Y) \in S$ where the distance *decreases* after one step (in expected value), then we can try the following approach.

The following lemma is the "delayed path coupling lemma" [8, Lemma 4.2] of Czumaj et al., which shows how the mixing time of a Markov chain may be related to the behavior of a t-step path coupling (which may be non-Markovian). For completeness, we present a proof.

LEMMA 2.2. *Let $S \subseteq \Omega \times \Omega$ be such that the transitive closure of S is the whole of $\Omega \times \Omega$. Let ρ be an integer-valued metric on $\Omega \times \Omega$ which takes values in $\{0, \ldots, D\}$. Given $(X_0, Y_0) \in S$, let $(X_0, Y_0), (X_1, Y_1), \ldots, (X_t, Y_t)$ be the t-step evolution of a (possibly non-Markovian) coupling starting from (X_0, Y_0). Suppose that there exists a constant γ such that $0 < \gamma < 1$ and*

$$\mathbf{E}\left[\rho(X_t, Y_t)\right] \le \gamma \rho(X_0, Y_0) \tag{2.2}$$

for all $(X_0, Y_0) \in S$. Then the mixing time $\tau(\varepsilon)$ of $\mathcal{M}$ satisfies

$$\tau(\varepsilon) \le \frac{\log(D\varepsilon^{-1})}{1 - \gamma} \cdot t.$$

Proof. Using the same argument as the path coupling lemma, we know that (2.2) holds for all $(X_0, Y_0) \in \Omega \times \Omega$. Run the coupling in epochs of length t. After r epochs, we have

$$\mathbf{E}\left[\rho(X_{rt}, Y_{rt})\right] \leq \gamma^r \, \rho(X_0, Y_0) \leq \gamma^r \, D.$$

If $r \geq \log(D\varepsilon^{-1})/(1-\gamma)$ then $\mathbf{E}\left[\rho(X_{rt}, Y_{rt})\right] \leq \varepsilon$. This gives an upper bound for the number of epochs required to ensure that the distribution of the chain is at most ε away from stationarity, in terms of total variation distance. Multiplying this number by t, the number of steps per epoch, gives the mixing time of the chain. □

Therefore it suffices to show that $\mathbf{E}\left[\rho(X_t, Y_t)\right] \leq \gamma \, \rho(X_0, Y_0)$ for all $(X_0, Y_0) \in S$, where γ is some positive constant less than 1. The main contribution of this paper is to provide a new approach to bounding $\mathbf{E}\left[\rho(X_t, Y_t)\right]$, which we now describe. Let $(X, Y) \mapsto (X', Y')$ be a (one-step, Markovian) coupling for $\mathcal{M}$ defined on all initial pairs in S; that is, $(X, Y) \in S$ and $(X', Y') \in \Omega \times \Omega$. We will apply this coupling for t steps, using the path coupling machinery to drive the coupling if the trajectory of the coupling leaves the set S. This gives a multiple-step coupling $\{(X_s, Y_s)\}_{s \geq 0}$. Let T be a stopping time for this coupling, defined in such a way that

$$\rho(X_s, Y_s) = \rho(X_0, Y_0) \text{ for } 0 \leq s < T.$$

Then T is a random variable which depends only on the history of the coupling up to the present time. For example, we could define T to be the first time at which the value of ρ changes.

If $T > t$ then we know that $\rho(X_t, Y_t) = \rho(X_0, Y_0)$. Otherwise, we consider (X_T, Y_T), the state of the coupling at the stopping time T. (The pair (X_T, Y_T) need no longer belong to the set S, but the path coupling machinery drives the coupling for all pairs in $\Omega \times \Omega$.) The analysis gives an upper bound for the quantity

$$\mathbf{E}\left[\rho(X_T, Y_T) \mid T \leq t\right].$$

We hope that this quantity will be smaller than $\mathbf{E}\left[\rho(X_1, Y_1)\right]$, with the following heuristic justification. The analysis of one-step coupling is a worst-case analysis. However, after running the Markov chain for T steps, the effect of the chosen starting state is mitigated to some extent by the random choices made during the running of the coupling. In other words, with some positive probability we are not in the worst case. It is here that we can improve on one-step coupling.

We now show how to relate $\mathbf{E}\left[\rho(X_T, Y_T) \mid T \leq t\right]$ and $\mathbf{E}\left[\rho(X_t, Y_t)\right]$.

THEOREM 2.3. *Let $\mathcal{M}$ be a Markov chain with state space Ω. Let ρ be a metric on $\Omega \times \Omega$ and let S be some subset of $\Omega \times \Omega$ such that the transitive closure of S is $\Omega \times \Omega$. Suppose that we have a (one-step, Markovian) coupling $(X, Y) \mapsto (X', Y')$, defined on pairs in S such that*

$$\mathbf{E}\left[\rho(X', Y')\right] \leq \beta \, \rho(X, Y)$$

for some constant β such that $\beta \geq 1$. Let $t > 0$ be a fixed integer. Apply the coupling for t steps from initial state $(X_0, Y_0) \in S$, using the path coupling lemma. Let T be some stopping time for $\{(X_s, Y_s)\}_{s \geq 0}$ such that

$$\rho(X_s, Y_s) = \rho(X_0, Y_0)$$

whenever $0 \leq s < T$. Then

$$\mathbf{E}\left[\rho(X_t, Y_t)\right] \leq \text{Prob}\left[T > t\right] \cdot \rho(X_0, Y_0) + \text{Prob}\left[T \leq t\right] \cdot \beta^t \cdot \mathbf{E}\left[\rho(X_T, Y_T) \mid T \leq t\right]$$

for all $(X_0, Y_0) \in S$.

Proof. The coupling defined on the set S gives rise to a coupling $(X, Y) \mapsto (X', Y')$ on the entire set $\Omega \times \Omega$ such that $\mathbf{E}[\rho(X', Y')] \leq \beta\, \rho(X, Y)$ for all $(X, Y) \in \Omega \times \Omega$, by the path coupling lemma [4]. Let (X_0, Y_0), $(X_1, Y_1), \ldots, (X_t, Y_t)$ be the t-step evolution of this coupling from the starting state $(X_0, Y_0) \in S$.

If $T > t$ then $\rho(X_t, Y_t) = \rho(X_0, Y_0)$. Next suppose that $T \leq t$. Then

$$\begin{aligned}\mathbf{E}[\rho(X_t, Y_t) \mid T \leq t] &\leq \beta\, \mathbf{E}[\rho(X_{t-1}, Y_{t-1}) \mid T \leq t] \\ &\leq \mathbf{E}\left[\beta^{t-T}\, \rho(X_T, Y_T) \mid T \leq t\right] \\ &\leq \beta^t\, \mathbf{E}[\rho(X_T, Y_T) \mid T \leq t]\,.\end{aligned}$$

(By replacing $t - T$ by t we are, in effect, assuming that the stopping time occurs at the very beginning of the interval.) This proves the theorem. □

Suppose that S is the set of all pairs (X, Y) with $\rho(X, Y) = 1$. In this case, Theorem 2.3 can be rewritten to assert that

$$\mathbf{E}[\rho(X_t, Y_t) - 1] \leq \mathrm{Prob}[T \leq t]\left(\beta^t \cdot \mathbf{E}[\rho(X_T, Y_T) \mid T \leq t] - 1\right).$$

Combining Lemma 2.2 and Theorem 2.3, we see that γ can be defined to be the maximum of the values

$$1 - \mathrm{Prob}[T \leq t]\left(1 - \beta^t \cdot \mathbf{E}[\rho(X_T, Y_T) \mid T \leq t]\right) \tag{2.3}$$

over all $(X_0, Y_0) \in S$. In order to obtain a good bound on the mixing time of the chain, we aim to show that $\gamma < 1$. Clearly $\gamma < 1$ if

$$\beta^t\, \mathbf{E}[\rho(X_T, Y_T) \mid T \leq t] < 1$$

for all $(X_0, Y_0) \in S$.

3. Applying the new method to the Glauber dynamics for graph colorings. In this section we illustrate the new method by using it to analyze the mixing time of the Glauber dynamics for graph colorings.

Let $G = (V, E)$ be a given graph and let $\Omega_k(G)$ be the set of all proper k-colorings of G, where C is the set of colors. The Glauber dynamics is a Markov chain on $\Omega_k(G)$ with transitions from the current state according to the following procedure:

- choose $(v, i) \in V \times C$ uniformly at random,
- recolor v with i if this results in v being properly recolored.

This chain was analyzed by Jerrum [15] and independently by Salas and Sokal [21]. They proved that the chain is rapidly mixing for graphs with maximum degree Δ when $k > 2\Delta$. The fact that the chain is also rapidly mixing for $k = 2\Delta$ can be found in [4]. Jerrum showed that the Glauber dynamics has $O(n \log(n))$ mixing time for $k > 2\Delta$, and the best known upper bound when $k = 2\Delta$ was $O(n^3)$.

In section 3.1 we describe the standard path coupling for this chain. Section 3.2 contains the definition of the stopping time for this coupling, and gives a necessary condition for the success of the new method. In section 3.3 we perform the calculations needed to establish the necessary condition. All calculations are combined in section 3.4 to provide an $O(n \log(n))$ upper bound for the mixing time of the Glauber dynamics for Δ-regular, triangle-free graphs, when $(2 - \eta)\Delta \leq k \leq 2\Delta$, where η is a small positive constant.

Before we proceed, we present a proof of the "folklore" result that the mixing time of the Glauber dynamics is bounded below by $\Omega(n \log n)$. Our proof concerns graphs with no edges.

THEOREM 3.1. *Let G be the empty graph with n vertices, and let $k \geq 2$. Then*

$$\tau((2e)^{-1}) = \Omega(n \log n).$$

Proof. A *stopping rule* Γ (see [17]) is a map that associates every initial sequence w of Markov chain states with a number $\Gamma[w] \in [0,1]$, which is taken to be the probability that the sequence should continue. We can also think of Γ as a random variable taking values in $\{0,1,2,\ldots\}$, whose distribution depends only on $w_0,\ldots,w_\Gamma$ (and w_Γ is the state where we stop). If w_0 is drawn from the distribution σ and $\mathbf{E}[\Gamma]$ is finite and the distribution of final states is τ, then the rule is called a stopping rule from σ to τ. It is said to be *optimal* for σ and τ if $\mathbf{E}[\Gamma]$ is minimal. For each $x \in \Omega_k(G)$ let σ_x be the distribution concentrated on the state x. Define τ_2 to be the maximum, over all initial states x, of the expected length of an optimal stopping rule from σ_x to π. Since the Glauber dynamics is time-reversible, a result of Aldous [2, Lemma 12] applies, showing that

$$\tau((2e)^{-1}) \geq c\tau_2,$$

where $c = (1-e^{-1})^2/2$. Now let Γ be the stopping rule which says, "stop when you have visited every vertex of G at least once." (It may not be immediately apparent that this rule satisfies the definition of a stopping rule given in [17], since it uses information not encoded in the states of the chain. However, it is routine to formulate an equivalent randomized stopping rule which does fit the definition; see [18, p. 89].) Since G has no edges and every vertex has been randomly recolored, the coloring obtained at time Γ is distributed according to π. Hence Γ is a stopping rule from σ_x to π for all $x \in \Omega_k(G)$. Let $y \in \Omega_k(G)$ be any coloring of G such that $y(v) \neq x(v)$ for all $v \in V$. Then y is a halting state for this stopping rule (that is, the probability that the process will halt if it reaches y is 1). Since Γ has a halting state it is an optimal stopping rule, using [17, Theorem 5.1]. This shows that $\tau_2 = \mathbf{E}[\Gamma]$. Therefore $\tau((2e)^{-1})$ is bounded below by a constant times the expected number of steps required to visit every vertex at least once, and the latter is $\Theta(n \log n)$ by the well-known coupon collector's lemma (see, for example, [19, section 3.6]). ∎

3.1. Path coupling for the Glauber dynamics. We now give the standard path coupling analysis of the Glauber dynamics. The proximity function is given by Hamming distance, and we let $\mathcal{S}$ be the set of all pairs with Hamming distance 1. The state space of the Markov chain must be extended to the set of all colorings (including nonproper colorings) in order to be able to form a path of length $H(X,Y)$ between any two colorings $(X,Y) \in \Omega_k(G)$. (This approach is standard and does not cause any problems, since the nonproper colorings are transient states. The stationary distribution is uniform over all proper colorings, and zero elsewhere. Although the extended chain is no longer reversible, the path coupling lemma still applies. Moreover, the mixing time of the chain on the original state space is bounded above by the mixing time of the chain on the extended state space.)

Consider $(X,Y) \in \mathcal{S}$, so X and Y differ just at a single vertex v. Let $N(v)$ denote the set of neighbors of v in G. We can couple at (X,Y) as follows: choose (u,i) uniformly at random from $V \times C$. If $u = v$ then attempt to recolor v with i in both X and Y. This will either succeed in both or fail in both. If it succeeds then the Hamming distance decreases by 1. The only other moves which can affect the Hamming distance are when $u = w$ where $w \in N(v)$. In this case, if $i \notin \{X(v), Y(v)\}$ then attempt to recolor w with i in both X and Y. This will either succeed in both

or fail in both, and the Hamming distance is unaffected. If $i = X(v)$ then attempt to recolor w with $X(v)$ in X and attempt to recolor w with $Y(v)$ in Y. This will fail in both X and Y, so the Hamming distance is unaffected. Finally, if $i = Y(v)$ then attempt to recolor w with $Y(v)$ in X, and attempt to recolor w with $X(v)$ in Y. This may succeed or fail in either, so the Hamming distance could increase by 1 here. Thus the expected change in the Hamming distance is at most

$$-\frac{(k - |\{X(w) : w \in N(v)\}|)}{kn} + \frac{\Delta}{kn}.$$

In general, we have $|\{X(w) : w \in N(v)\}| \leq \Delta$, so that the expected change in the Hamming distance is at most $-(k - 2\Delta)/(kn)$. This gives nonincreasing Hamming distance for $k \geq 2\Delta$. The aim of the new approach is to show that, with constant positive probability, there are fewer than Δ distinct colors around v, just before the stopping time. This gives nonincreasing Hamming distance for a wider range of k.

3.2. A stopping time for the Glauber dynamics on colorings. For simplicity, assume that the given graph G is Δ-regular and triangle-free. Let η be a small positive constant which we fix later, and suppose that $(2-\eta)\Delta \leq k \leq 2\Delta$. We analyze the mixing time of the Glauber dynamics using our new method, to show that the Glauber dynamics has $O(n \log(n))$ mixing time for this range of k.

Let $(X_0, Y_0) \in S$ be given, so that X_0, Y_0 differ just at a single vertex $v \in V$. Perform the coupling described in section 3.1 with starting point (X_0, Y_0). Let $Q(X_0, Y_0)$ be the set of all moves which involve v or increase the Hamming distance; that is,

$$Q(X_0, Y_0) = \{(v, i) : i \in C\} \cup \{(w, Y_0(v)) : w \in N(v)\}.$$

Then $Q(X_0, Y_0)$ contains all the choices which may affect the Hamming distance, but also some which will not. Define the random variable T to be the first step at which a pair in $Q(X_0, Y_0)$ is chosen by the coupling. Then T is a stopping time since it depends only on the coupling up to the present time. Now (X_T, Y_T) is the state of the coupling after the Tth step, which we refer to as the state of the coupling at the stopping time. Note that $H(X_s, Y_s) = H(X_0, Y_0) = 1$ for $0 \leq s < T$ by the analysis of section 3.1. Clearly $|Q(X_0, Y_0)| = k + \Delta$ for all pairs $(X_0, Y_0) \in \mathcal{S}$. Let δ be a positive constant, and assume that δn is an integer. (Since n can grow arbitrarily large, there is not much harm in making this assumption.) An (approximately) optimal value of δ will be fixed later, which will satisfy $\delta < (2 - \eta)/3$. We run the coupling for t steps, where $t = \delta n$.

Let $\mathcal{C}$ be a random variable which denotes the number of colors which occur more than once around v *just before* the stopping time T (that is, after step $T - 1$). In the next section we prove that, when n and Δ are "big enough" and η is "small enough," we have

$$\mathbf{E}[\mathcal{C} \mid T \leq \delta n] \geq \xi\Delta$$

for some constant ξ such that $\xi \geq 2\eta$. We now show why this is sufficient.

The arguments of section 3.1 show that the expected value of the Hamming distance after one step of normal path coupling from $(X, Y) \in S$ is at most

$$1 - \frac{k - 2\Delta}{kn} \leq 1 + \frac{\eta\Delta}{kn}$$

since $(2-\eta)\Delta \le k \le 2\Delta$. Next, notice that

$$\begin{aligned}\mathbf{E}\left[H(X_T,Y_T)-1 \mid T \le \delta n\right] &\le -\frac{k-(\Delta-\mathbf{E}\left[\mathcal{C} \mid T \le \delta n\right])}{k+\Delta}+\frac{\Delta}{k+\Delta}\\ &\le -\frac{k-(1-2\eta)\Delta}{k+\Delta}+\frac{\Delta}{k+\Delta}\\ &\le -\frac{\eta}{3}.\end{aligned}$$

Therefore, using Theorem 2.3 (and in particular the remarks following the theorem),

$$\begin{aligned}\mathbf{E}\left[H(X_{\delta n},Y_{\delta n})-1\right] &\le \mathrm{Prob}\left[T \le \delta n\right]\left(\beta^{\delta n}\cdot \mathbf{E}\left[H(X_T,Y_T) \mid T \le \delta n\right]-1\right)\\ &\le \mathrm{Prob}\left[T \le \delta n\right]\left(\left(1+\frac{\eta\Delta}{kn}\right)^{\delta n}\left(1-\frac{\eta}{3}\right)-1\right)\\ &\le \mathrm{Prob}\left[T \le \delta n\right]\left(e^{\eta\delta/(2-\eta)}e^{-\eta/3}-1\right). \qquad (3.1)\end{aligned}$$

This quantity is nonpositive whenever

$$\frac{\eta\delta}{2-\eta}-\frac{\eta}{3} \le 0,$$

and this holds for $\delta \le (2-\eta)/3$.

We now calculate a lower bound for $\mathbf{E}\left[T \mid T \le \delta n\right]$, which is needed in section 3.3.

LEMMA 3.2. *Suppose that $n \ge \delta^{-1}$ and $(2-\eta)\Delta \le k \le 2\Delta$, where $0<\eta<2$. Let $\theta = 3/(2-\eta)$. Then*

$$\mathbf{E}\left[T \mid T \le \delta n\right] \ge \frac{\delta n}{2}(1-\theta\delta).$$

Proof. Let $q = 1-(k+\Delta)/(kn)$, and let p_s denote the probability that $T=s$. Then $p_s = \mathrm{Prob}\left[T=s\right] = (1-q)q^{s-1}$ and $\mathrm{Prob}\left[T \le \delta n\right] = 1-q^{\delta n}$. Now $q^{\delta n} \ge 1-(k+\Delta)\delta/k$ since $n \ge \delta^{-1}$. Therefore

$$\begin{aligned}\mathbf{E}\left[T \mid T \le \delta n\right] &= \left(1-q^{\delta n}\right)^{-1}\sum_{s=0}^{\delta n} s p_s\\ &\ge \frac{p_{\delta n}\,\delta^2\,n^2}{2\left(1-q^{\delta n}\right)}\\ &> \frac{(1-q)\,q^{\delta n}\,\delta^2\,n^2}{2\left(1-q^{\delta n}\right)}\\ &\ge \frac{(1-q)\left(1-\frac{k+\Delta}{k}\delta\right)}{2\frac{k+\Delta}{k}\delta}\,\delta^2\,n^2\\ &\ge (1-\theta\delta)\,\frac{\delta n}{2},\end{aligned}$$

as claimed. ∎

3.3. The expected number of repeated colors just before the stopping time. Let (X_0,Y_0) be a given pair in $\mathcal{S}$ and let v be the vertex which is colored differently in X and Y. Let T be the stopping time for the coupling when started at

(X_0, Y_0). Denote by $\mathcal{C}$ the number of colors which occur at least twice around v just before the stopping time T. That is,

$$\mathcal{C} = |\{i \in C : |\{w \in N(v) : X_{T-1}(w) = i\}| \geq 2\}|.$$

In this section we obtain a lower bound for $\mathbf{E}[\mathcal{C} \mid T \leq \delta n]$ which holds when Δ and n are both "large enough" and η is "small enough." Specifically, take $\Delta \geq 14$, $n \geq 120$, and $\eta < 1/210$.

Let A_w be defined by

$$A_w = C \setminus (\{X_0(u) : \{u, w\} \in E\} \cup \{Y_0(v)\})$$

for $w \in N(v)$. Then A_w is the set of colors which are acceptable at w in both X_0 and Y_0. Note that $|A_w| \geq k - \Delta - 1$ for all $w \in N(v)$. Next, let

$$B_i = \{w \in N(v) : i \in A_w\}$$

and let $b_i = |B_i|$ for each $i \in C$. So B_i is the set of vertices $w \in N(v)$ at which i is acceptable in both X_0 and Y_0.

LEMMA 3.3. *Assume that $\eta < 1/210$ and $\Delta \geq 14$. Let k satisfy $(2-\eta)\Delta \leq k \leq 2\Delta$. Then there are at least $\lceil k/5 \rceil$ colors i such that $b_i \geq \Delta/3$.*

Proof. Let $Z = |\{(i, w) : i \in A_w\}|$. Now $Z \geq \Delta(k - \Delta - 1)$. For a contradiction, suppose that fewer than $\lceil k/5 \rceil$ colors i have $b_i \geq \Delta/3$. If k is a multiple of 5 then

$$\begin{aligned} Z &\leq \left(\frac{k}{5} - 1\right)\Delta + \left(\frac{4k}{5} + 1\right)\frac{\Delta}{3} \\ &\leq \left(1 - \left(\frac{1}{15} - \frac{1}{3\Delta}\right)\right)\Delta^2 - \Delta \\ &< \Delta(k - \Delta - 1), \end{aligned}$$

giving the desired contradiction. Next, suppose that $k = 5\ell + r$, where $r \in \{1, 2, 3, 4\}$. Then

$$\begin{aligned} Z &\leq \ell\Delta + (k - \ell)\frac{\Delta}{3} \\ &\leq \left(1 - \left(\frac{1}{15} - \frac{15 - 2r}{15\Delta}\right)\right)\Delta^2 - \Delta \\ &< \Delta(k - \Delta - 1) \end{aligned}$$

since $\eta < 1/210$ and $\Delta \geq 14$. Again, this is a contradiction. □

Using this information we can prove a lower bound for the expected number of repeated colors around v just before the stopping time, given that the stopping time occurs in the first δn steps.

THEOREM 3.4. *Suppose that $n \geq 120$, $\Delta \geq 14$, $\eta < 1/210$, and $\delta < (2-\eta)/3$. Also assume that $(2-\eta)\Delta \leq k \leq 2\Delta$. Then*

$$\mathbf{E}[\mathcal{C} \mid T \leq \delta n] \geq \frac{1}{3840} \cdot \delta^2 (1 - \theta\delta)^2 \cdot e^{-4\delta} \cdot \Delta,$$

where $\theta = 3/(2-\eta)$.

Proof. By Lemma 3.3, there are at least $\lceil k/5 \rceil$ colors i such that $b_i \geq \Delta/3$. Consider ways in which such a color i can occur at least twice around v just before

the stopping time T. One way in which this can occur is as follows. Suppose that there are exactly two distinct elements u, $w \in B_i$ which were chosen with the color i during the coupling. That is, (u,i) and (w,i) were both chosen but (q,i) was not chosen for any $q \in B_i \setminus \{u,w\}$. Also suppose that u and w are never chosen at any other time, with any color, and that no neighbor of u or w is ever chosen with color i. In this situation, both u and w end up colored i. We now analyze the probability that this event occurs for a given value of T.

We know that T is the first stopping time, so there are $T-1$ steps before the stopping time step. We do not have kn possible choices at each of these $T-1$ steps, but rather $kn-(k+\Delta)$ possibilities. With this in mind, the probability that, say, w is chosen with color i is given by $1/(kn-(k+\Delta)) \geq 1/(kn)$. There are at least $\Delta^2/24$ choices for the unordered pair $\{u,w\} \subseteq B_i$ since $b_i \geq \Delta/3$ and $\Delta \geq 14$. The probability that both (u,i) and (w,i) are chosen at two distinct times in the first $T-1$ steps is at least $\binom{T-1}{2} \cdot 1/(k^2n^2)$. There are also choices which we have ruled out for all other steps, corresponding to the vertex-color pairs from the set

$$\{(q,i) : q \in (N(u) \cup N(w) \setminus \{v\}) \cup (B_i \setminus \{u,w\})\} \bigcup \{(u,j),\,(w,j) : j \in C \setminus \{Y(v)\}\}.$$

(Note that the selection of $j = Y(v)$ is ruled out because s is not a stopping time for $0 \leq s < T$.) We have ruled out at most $3\Delta + 2k - 6$ choices at each of $T-3$ steps. Thus we see that

$$\text{Prob}\left[i \text{ is repeated} \mid T,\, b_i \geq \frac{\Delta}{3}\right] \geq \frac{\Delta^2}{24} \cdot \binom{T-1}{2} \cdot \frac{1}{k^2n^2} \cdot \left(1 - \frac{3\Delta + 2k - 6}{kn - (k+\Delta)}\right)^{T-3}.$$

Let $x = 3\Delta + 2k - 6$ and $y = kn - (k+\Delta)$. Then

$$\begin{aligned}
\left(1 - \frac{x}{y}\right)^{T-3} &= \exp\left(-(T-3)\sum_{i=1}^{\infty} \frac{1}{i}\left(\frac{x}{y}\right)^i\right) \\
&= \exp\left(-\frac{Tx}{y} + \sum_{i=1}^{\infty}\left(\frac{3}{i} - \frac{Tx}{(i+1)y}\right)\left(\frac{x}{y}\right)^i\right) \\
&\geq e^{-Tx/y} \\
&\geq e^{-4\delta}.
\end{aligned}$$

The first inequality follows since $3/i \geq Tx/((i+1)y)$ for all $i \geq 1$, and the second inequality follows since $4y \geq nx$ (using the definition of x, y and the assumptions of the theorem). Plugging this back into our calculations, we obtain

$$\text{Prob}\left[i \text{ is repeated} \mid T,\, b_i \geq \frac{\Delta}{3}\right] \geq \frac{\Delta^2}{24} \cdot \binom{T-1}{2} \cdot \frac{1}{k^2n^2} \cdot e^{-4\delta}.$$

Now we shall take expectation with respect to T, conditional on $T \leq \delta n$. Using Lemma 3.2 and the fact that $n \geq 120$, we find that

$$\binom{\mathbf{E}\left[T \mid T \leq \delta n\right] - 1}{2} \geq \frac{\delta^2\, n^2\, (1-\theta\delta)^2}{16}.$$

Applying Jensen's inequality, we obtain

$$\begin{aligned}
\text{Prob}\left[i \text{ is repeated} \mid T \leq \delta n,\, b_i \geq \frac{\Delta}{3}\right] &\geq \frac{\Delta^2}{384} \cdot \delta^2\, n^2\, (1-\theta\delta)^2 \cdot \frac{1}{k^2n^2} \cdot e^{-4\delta} \\
&\geq \frac{1}{768} \cdot \delta^2\, (1-\theta\delta)^2 \cdot e^{-4\delta} \cdot \frac{\Delta}{k}. \qquad (3.2)
\end{aligned}$$

By summing (3.2) over the $\lceil k/5 \rceil$ most popular colors, the theorem is proved. □

3.4. The mixing time of the Glauber dynamics. We now calculate an upper bound for the mixing time of the Glauber dynamics, using Lemma 2.2 and Theorem 2.3. Let ξ be defined by

$$\begin{aligned}\xi(\delta,\eta) &= \frac{1}{3840}\cdot\delta^2\,(1-\theta\delta)^2\,\cdot e^{-4\delta}\\ &= \frac{1}{3840}\cdot\delta^2\left(1-\frac{3\delta}{2-\eta}\right)^2\cdot e^{-4\delta}.\end{aligned}$$

Theorem 3.4 shows that $\mathbf{E}\,[\mathcal{C} \mid T \le \delta n] \ge \xi\Delta$. Note that ξ is a decreasing function of η. Take $\delta = 1/8$ and $\eta = 8\times 10^{-7}$. Then $\xi(\delta,\eta) \ge 2\eta$. (These values of δ, η are approximately optimal.) The discussion of section 3.2 suggested that this condition was sufficient to ensure rapid mixing of the Glauber dynamics. We now give the details.

THEOREM 3.5. *Let $n \ge 120$ and $\Delta \ge 14$. Suppose that $(2-\eta)\Delta \le k \le 2\Delta$, where $\eta = 8\times 10^{-7}$. The mixing time of the Glauber dynamics for graph colorings of Δ-regular, triangle-free graphs is bounded above by*

$$\tau(\varepsilon) \le 4\times 10^6\, n\log(n\varepsilon^{-1}).$$

Proof. Let $\delta = 1/8$, as in the previous section. We bound the mixing time by finding an upper bound on the quantity γ such that

$$H(X_{\delta n}, Y_{\delta n}) \le \gamma$$

over all initial pairs $(X_0, Y_0) \in S$. Using the remark following Theorem 2.3, we can define γ by (2.3). Let $q = 1-(k+\Delta)/(kn)$, as in Lemma 3.2. Then

$$\begin{aligned}\text{Prob}\,[T \le \delta n] &= 1-q^{\delta n}\\ &\ge 1-\exp\left(-\frac{k+\Delta}{k}\delta\right) \ge 1-e^{-4\delta/3}.\end{aligned}$$

Using this, with the calculations of (3.1), we obtain

$$\begin{aligned}\gamma &\le 1-\left(1-e^{-4\delta/3}\right)\left(1-\exp\left(\frac{\eta\delta}{2-\eta}-\frac{\eta}{3}\right)\right)\\ &\le 1-3.3\times 10^{-8},\end{aligned}$$

substituting $\delta = 1/8$ and $\eta = 8\times 10^{-7}$. Now applying Lemma 2.2 we find that the mixing time of the Glauber dynamics is bounded above by

$$\begin{aligned}\tau(\varepsilon) &\le \delta n\cdot\frac{\log(n\varepsilon^{-1})}{1-\gamma}\\ &\le \frac{10^8}{26.4}\,n\log(n\varepsilon^{-1})\\ &< 4\times 10^6\, n\log(n\varepsilon^{-1}).\end{aligned}$$

This bound holds for $(2-\eta)\Delta \le k \le 2\Delta$, where $\eta = 8\times 10^{-7}$, assuming that $\Delta \ge 14$ and $n \ge 120$. □

Vigoda [23] described a new Markov chain for graph colorings which alters the coloring of up to six vertices at each transition. He showed using path coupling that this chain is rapidly mixing for $k \geq 11\Delta/6$. The mixing time of this chain is bounded above by

$$\frac{k}{k - \frac{11}{6}\Delta}\, n \log(n\varepsilon^{-1})$$

for $k > 11\Delta/6$. Vigoda also applies the comparison technique of Diaconis and Saloff-Coste [9] to show that the mixing time of the Glauber dynamics is at most

$$O\left(k \log(k)\, n^2 \log(n)\right)$$

when $k > 11\Delta/6$. In particular, this gives an upper bound of $O(n^2 \log n)$ when $k = 2\Delta$. It seems unlikely that any comparison technique could yield the optimal bound of $O(n \log n)$.

REFERENCES

[1] S. Albers and M. Mitzenmacher, *Average-case analyses of first fit and random fit bin packing*, in 9th Annual Symposium on Discrete Algorithms, ACM–SIAM, New York–Philadelphia, 1998, pp. 290–299.

[2] D. J. Aldous, *Some inequalities for reversible Markov chains*, J. London Math. Soc. 2, 25 (1982), pp. 564–576.

[3] D. Aldous, *Random walks on finite groups and rapidly mixing Markov chains*, in Séminaire de Probabilités XVII 1981/1982, Lecture Notes in Math. 986, A. Dold and B. Eckmann, eds., Springer-Verlag, New York, 1983, pp. 243–297.

[4] R. Bubley and M. Dyer, *Path coupling: A technique for proving rapid mixing in Markov chains*, in 38th Annual Symposium on Foundations of Computer Science, IEEE, Los Alamitos, 1997, pp. 223–231.

[5] R. Bubley, M. Dyer, C. Greenhill, and M. Jerrum, *On approximately counting colorings of small degree graphs*, SIAM J. Comput., 29 (1999), pp. 387–400.

[6] F. R. K. Chung and R. L. Graham, *Random walks on generating sets for finite groups*, Electron. J. Combin., 4 (1997), Research Paper 7, http://www.combinatorics.org/Volume_4/Abstracts/v4i2r7.html.

[7] E. G. Coffman, D. S. Johnson, P. W. Shor, and R. R. Weber, *Markov chains, computer proofs and average-case analysis of best fit bin packing*, in 25th Annual Symposium on the Theory of Computing, ACM, New York, 1993, pp. 412–421.

[8] A. Czumaj, P. Kanarek, M. Kutyłowski, and K. Loryś, *Delayed path coupling and generating random permutations via distributed stochastic processes*, in 10th Annual Symposium on Discrete Algorithms, ACM–SIAM, New York, Philadelphia, 1999, pp. 271–280.

[9] P. Diaconis and L. Saloff-Coste, *Comparison theorems for reversible Markov chains*, Ann. Appl. Probab., 3 (1993), pp. 696–730.

[10] P. Diaconis and L. Saloff-Coste, *Logarithmic Sobolev inequalities for finite Markov chains*, Ann. Appl. Probab., 6 (1996), pp. 695–750.

[11] M. Dyer and C. Greenhill, *On Markov chains for independent sets*, J. Algorithms, 35 (2000), pp. 17–49.

[12] M. Dyer and C. Greenhill, *Random walks on combinatorial objects*, in Surveys in Combinatorics 1999, London Math. Soc. Lecture Note Ser. 267, J. D. Lamb and D. A. Preece, eds., Cambridge University Press, Cambridge, UK, 1999, pp. 101–136.

[13] L. A. Goldberg and P. D. MacKenzie, *Analysis of practical backoff protocols for contention resolution with multiple servers*, J. Comput. System Sci., 58 (1999), pp. 232–258.

[14] J. Håstad, T. Leighton, and B. Rogoff, *Analysis of backoff protocols for multiple access channels*, SIAM J. Comput., 25 (1996), pp. 740–774.

[15] M. Jerrum, *A very simple algorithm for estimating the number of k-colorings of a low-degree graph*, Random Structures Algorithms, 7 (1995), pp. 157–165.

[16] C. Kenyon, A. Sinclair, and Y. Rabani, *Biased random walks, Lyapunov functions, and stochastic analysis of best fit bin packing*, J. Algorithms, 27 (1998), pp. 218–235.

[17] L. Lovász and P. Winkler, Efficient stopping rules for Markov chains, in 27th Annual Symposium on the Theory of Computing, ACM, New York, 1995, pp. 76–82.
[18] L. Lovász and P. Winkler, *Mixing times*, in Microsurveys in discrete probability, vol. 41 of DIMACS Ser. Discrete Math. Theoret. Comput. Sci., AMS, Providence, RI, 1998, pp. 85–133.
[19] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.
[20] D. Randall and P. Tetali, *Analysing Glauber dynamics by comparison of Markov chains*, J. Math. Phys., 41 (2000), pp 1598–1615.
[21] J. Salas and A. D. Sokal, *Absence of phase transition for antiferromagnetic Potts models via the Dobrushin uniqueness theorem*, J. Statist. Phys., 86 (1997), pp. 551–579.
[22] R. Swendsen and J. S. Wang, *Universal critical dynamics in Monte-Carlo simulation*, Phys. Rev. Lett., 58 (1987), pp. 86–88.
[23] E. Vigoda, *Improved bounds for sampling colorings*, J. Math. Phys., 41 (2000), pp. 1555–1569.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1976–1992

OPTIMAL SIMULATIONS BETWEEN UNARY AUTOMATA*

CARLO MEREGHETTI† AND GIOVANNI PIGHIZZINI‡

Abstract. We consider the problem of computing the costs—*in terms of states*—of optimal simulations between different kinds of finite automata recognizing *unary* languages. Our main result is a tight simulation of unary n-state two-way nondeterministic automata by $O(\mathrm{e}^{\sqrt{n \ln n}})$-state one-way deterministic automata. In addition, we show that, given a unary n-state two-way nondeterministic automaton, one can construct an equivalent $O(n^2)$-state two-way nondeterministic automaton performing *both* input head reversals and nondeterministic choices only at the ends of the input tape. Further results on simulating unary one-way alternating finite automata are also discussed.

Key words. formal languages; deterministic, nondeterministic, and alternating finite state automata; unary languages

AMS subject classifications. 68Q10, 68Q45, 68Q68

PII. S009753979935431X

1. Introduction.

Finite automata are probably one of the simplest and most extensively studied models of computation. First of all, their computational power is well established: they exactly define the class of regular languages. Furthermore, it is also well known that several added features, such as nondeterminism, alternation, and two-way motion of the input head, do not increase their computing ability. Equivalences are clearly obtained by simulating different kinds of automata by the original device, the one-way deterministic finite automaton (1dfa) [22]. Here, we are particularly interested in the cost—*in terms of states*—of simulations between automata.

The following are the well-known costs of simulating different automata by 1dfa's (number of states of the best 1dfa simulating any n-state automaton in the class): one-way nondeterministic finite automata (1nfa): $O(2^n)$ [22], one-way alternating finite automata (1afa): $O(2^{2^n})$ [7], two-way deterministic finite automata (2dfa): $O(n^n)$ [22, 24], two-way nondeterministic finite automata (2nfa): $O(2^{n^2})$ [22, 24]. All these bounds are tight. We refer the reader to [3] which is a valuable source for results and references.

There are several open questions concerning automata simulation, the most important being probably the one posed by Sakoda and Sipser in [23]: how many states are necessary and sufficient to simulate 2nfa's (or 1nfa's) by 2dfa's? They conjecture such a cost to be exponential, and Sipser [25] proves that this is exactly the case when 2dfa's are required to be *sweeping*, i.e., to have head reversals only at the ends of the input tape. Berman and Lingas [2] state a lower bound of $\Omega(n^2/\log n)$ for the general problem and provide an interesting connection with the celebrated open problem DLOGSPACE $\stackrel{?}{=}$ NLOGSPACE. More precisely, they show that

*Received by the editors April 20, 1999; accepted for publication (in revised form) July 21, 2000; published electronically March 20, 2001. This work was partially supported by MURST, under the project "Modelli di calcolo innovativi: metodi sintattici e combinatori." A preliminary version of this paper [20] has been presented to the 15*th Annual Symposium on Theoretical Aspects of Computer Science* 1998 *(STACS*98*)*, Paris, France, February 25–27, 1998.

http://www.siam.org/journals/sicomp/30-6/35431.html

†Dipartimento di Informatica, Sistemistica e Comunicazione, Università degli Studi di Milano–Bicocca, via Bicocca degli Arcimboldi 8, 20126 Milano, Italy (mereghetti@disco.unimib.it).

‡Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39, 20135 Milano, Italy (pighizzi@dsi.unimi.it).

if DLOGSPACE = NLOGSPACE then for some polynomial p and for all integers m and k-state 2nfa A, there is a $p(mk)$-state 2dfa accepting a subset of $\mathcal{L}(A)$, the language accepted by A. The subset consists of all strings of length no more than m in $\mathcal{L}(A)$. As a consequence of this result, Sipser [25] relates the DLOGSPACE $\stackrel{?}{=}$ NLOGSPACE question also to the existence of sweeping automata with a polynomial number of states for a certain family of regular languages. This might give added evidence that the problem of evaluating how the number of states changes when turning one automaton into another is not only motivated by the investigation on the succinctness of representing regular languages but is also related to fundamental questions in complexity.

It is important to stress that the optimality of such simulations has been established by witness languages built over alphabets of two or more symbols. As a matter of fact, the situation turns out to be quite different whenever we restrict the problems to *unary* automata, i.e., automata with a single letter input alphabet. The problem of evaluating the costs of unary automata simulations was raised in [25] and has led to emphasize some relevant differences with the general case. For instance, we know that $O(\mathrm{e}^{\sqrt{n \ln n}})$ states suffice in order to simulate a unary n-state 1nfa or 2dfa by a 1dfa. Furthermore, a unary n-state 1nfa can be simulated by a 2dfa having $O(n^2)$ many states. All these results and their optimality have been proved in 1986 by Chrobak [8].

In this paper, we further deepen the study of optimal simulations between unary automata. To this aim, we find it useful to consider some techniques from the *sublogarithmic space world* (see, e.g., [11, 12, 27]).

The first part of the paper is devoted to study unary 2nfa's. We closely analyze the structure of their computation paths. In particular, using graph theoretical and number theoretical arguments, we show that, for sufficiently large inputs, it is possible to consider only computation paths in which states are repeated in a very regular way. This allows us to state our main result concerning the optimal simulation of unary 2nfa's by 1dfa's: each unary n-state 2nfa can be optimally simulated by a 1dfa with $O(\mathrm{e}^{\sqrt{n \ln n}})$ states. Note that such a complexity is the same as the above mentioned optimal simulations of unary 1nfa's and 2dfa's by 1dfa's. Thus, we can conclude that the simultaneous elimination of both two-way motion and nondeterminism on unary automata has the same cost as the elimination of either of them.

As another consequence of our analysis of unary 2nfa's, we are able to prove that each unary n-state 2nfa can be simulated by a 2nfa with $O(n^2)$ states which reverses the input head direction and makes nondeterministic decisions *only* when the input head visits the left or the right end of the input. This result can be regarded as a further step toward the solution of the Sakoda–Sipser open problem recalled above, at least for unary inputs.

In the second part of the paper, we study the relationships between unary 1afa's, dfa's and nfa's. This problem was proposed in [8] and partly solved in [3], where it is proved that 2^n states are necessary for 2nfa's to simulate unary n-state 1afa's. Here, we point out an optimal simulation of unary n-state 2nfa's by $O(\sqrt{n \ln n})$-state 1afa's.

By combining our results with those in the literature, we are able to draw an almost complete description of the costs of optimal simulations between the different types of unary automata here considered. We do not explicitly list all the resulting optimal bounds, which can be better understood by observing Figure 1.1.

2. Preliminary notions and results. In this section, we begin by recalling basic notions on finite state automata (subsection 2.1). Then, for the reader's ease of

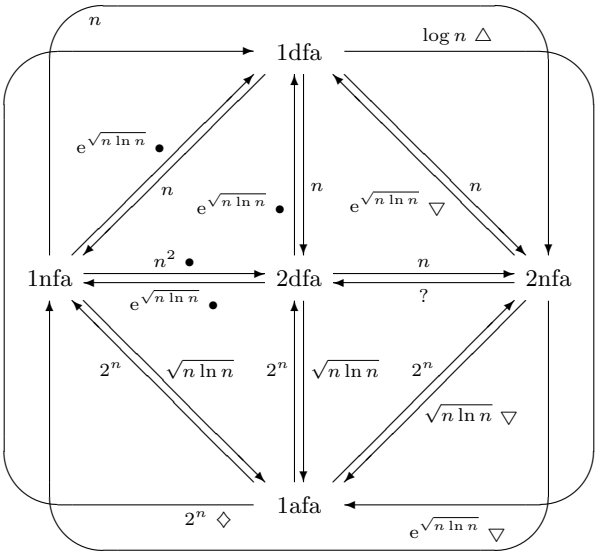


FIG. 1.1. *Costs of the optimal simulations between different kinds of unary automata. An arc labeled $f(n)$ from a vertex x to a vertex y means that a unary n-state automaton of type x can be simulated by a $O(f(n))$-state automaton of type y. Costs marked by "•" are proved in* [8]*, by "◇" in* [7]*, by "△" in* [17]*, by "▽" in this paper. The unmarked costs can be trivially obtained. The arc labeled "?" represents the open question of Sakoda and Sipser* [23].

mind, a brief outline of the main result of this paper—the $O(\mathrm{e}^{\sqrt{n \ln n}})$ simulation of 2nfa's by 1dfa's—is provided (subsection 2.2). In the following two subsections, we set up some mathematical tools to study the costs of our simulations. More precisely, we prove some new facts concerning *linear Diophantine equations* (subsection 2.3) and *directed graphs* (subsection 2.4).

2.1. Finite state automata. Given a set S, $\sharp S$ denotes its cardinality, and 2^S the family of all its subsets. Given an alphabet Σ, Σ^* denotes the set of strings on Σ, with the empty string ϵ. Given a string $x \in \Sigma^*$, $|x|$ denotes its length. A language $\mathcal{L}$ is said to be *unary* (or *tally*) whenever it can be built over a *single letter* alphabet. In this case, we let $\mathcal{L} \subseteq 1^*$.

Let us take a brief look on the computational model of finite automata. For a detailed exposition, we refer the reader to [14]. A *one-way nondeterministic finite automaton* is a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where Q is the finite set of states, Σ is the finite input alphabet, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states. The transition function δ can be extended to strings in a standard way. The language accepted by A is the set $\mathcal{L}(A) = \{x \in \Sigma^* \mid \delta(q_0, x) \cap F \neq \emptyset\}$. The one-way automaton A is *deterministic* if and only if $\sharp\delta(q, \sigma) = 1$, for any $q \in Q$, and $\sigma \in \Sigma$. (This implies that deterministic automata are assumed to be *complete*.)

It is well known that a one-way automaton can be viewed as a control unit that reads, by an input head, a tape containing input strings stored one symbol per square. At each move, the input head is shifted one square right. This model can be extended to encompass *stationary* and *left* moves of the input head. More precisely, we can define a 2nfa as a 5-tuple $A = (Q, \Sigma, \delta, q_0, F)$ in which Q, Σ, q_0, and F are defined as for 1nfa's, while $\delta : Q \times (\Sigma \cup \{\vdash, \dashv\}) \to 2^{Q \times \{-1, 0, +1\}}$ is the transition function, where

$\vdash, \dashv$ are two special symbols—the left and the right endmarker, respectively—not in Σ. Input strings are stored onto the input tape surrounded by the two endmarkers, the left endmarker being at the 0th square. The computation begins with the input head scanning the 0th square. In a move, A reads an input symbol, changes its state, and moves the input head one position forward or backward or keeps it stationary, depending on whether δ returns $+1$, -1, or 0, respectively. Left and right moves on the left and right endmarker, respectively, are forbidden. A is *deterministic* if and only if $\sharp\delta(q, \sigma) = 1$, for any $q \in Q$, and $\sigma \in \Sigma \cup \{\vdash, \dashv\}$.

Alternating finite automata[1] provide a natural generalization of nondeterministic automata. We now briefly recall their formal definition and refer the reader to [7, 10] for more details. A *one-way alternating finite automaton* is a 5-tuple $A = (Q, \Sigma, g, q_1, F)$, where $Q = \{q_1, \ldots, q_k\}$ is the set of states, Σ is the input alphabet, $F \subseteq Q$ is the set of final states, q_1 is the initial state, and $g : Q \to (\Sigma \times \mathbf{B}^k \to \mathbf{B})$ is a mapping of Q into the set of all mappings of $\Sigma \times \mathbf{B}^k$ into $\mathbf{B} = \{0, 1\}$. To explain the behavior of A, consider, for $i = 1, \ldots, k$, the function $g_i : \Sigma \times \mathbf{B}^k \to \mathbf{B}$ representing the image by g of the state q_i, i.e., $g_i = g(q_i)$. Such functions can be inductively extended to strings to obtain $G_i : \Sigma^* \times \mathbf{B}^k \to \mathbf{B}$ as: $G_i(\epsilon, \mathbf{u}) = u_i$, and $G_i(\sigma x, \mathbf{u}) = g_i(\sigma,\ G_1(x, \mathbf{u}), \ldots, G_k(x, \mathbf{u}))$, for $\sigma \in \Sigma$, $x \in \Sigma^*$, and $\mathbf{u} = (u_1, \ldots, u_k) \in \mathbf{B}^k$. The language accepted by A is the set $\mathcal{L}(A) = \{x \in \Sigma^* \mid G_1(x, \mathbf{f}) = 1\}$, where $\mathbf{f} \in \mathbf{B}^k$ denotes the characteristic vector of the set of final states. Notice that $G_i(\epsilon, \mathbf{f}) = 1$ if and only if q_i is a final state. Moreover, $G_i(\sigma x, \mathbf{f})$ can be computed as follows: a process in the state q_i reads σ from the input tape, and splits into k independent processes computing $G_j(x, \mathbf{f})$, for $j = 1, \ldots, k$. Then, by applying $g_i(\sigma, \_)$ to all these results, we get $G_i(\sigma x, \mathbf{f})$.

It is easy to see that 1nfa's are just special cases of 1afa's up to setting

$$g_i(\sigma, \mathbf{u}) = \bigvee_{q_j \in \delta(q_i, \sigma)} u_j,$$

for $i = 1, \ldots, k$, $\sigma \in \Sigma$, and $\mathbf{u} \in \mathbf{B}^k$.

We call *unary* any automaton that works with a single letter input alphabet.

2.2. Outline of the main result. The proof of the optimal $O(\mathrm{e}^{\sqrt{n \ln n}})$ simulation of 2nfs's by 1dfa's, which is the main result of this paper, is rather long and complex. To help the reader, we emphasize the main ideas here.

Let $\mathcal{C}$ be an accepting computation path of an n-state 2nfa A on input 1^m. Along $\mathcal{C}$, consider the sequence $r_0, r_1, \ldots, r_p$ of states in which the input head scans either of the endmarkers. For $j = 1, \ldots, p$, the following two possibilities arise:

(i) *U-Turn*: in both r_{j-1} and r_j, the input head scans the same endmarker.

(ii) *Left-to-right (right-to-left) traversal:* in r_{j-1} the input head scans the left (right) endmarker, while in r_j it scans the other one.

For sufficiently large inputs, U-Turns can be nondeterministically guessed without moving the input head (by Lemma 3.1). As a consequence, the computation $\mathcal{C}$ can be reduced to a sequence of left-to-right and right-to-left traversals.

Now, the key point is that in each traversal the automaton A can only measure the length of the input 1^m modulo some integer $\ell_i \leq n$. Hence, it can be argued that if there exists a traversal from r_{j-1} to r_j on 1^m, then there also exists a traversal with the same endpoints on $1^{m+\mu\ell_i}$, for every (possibly negative) integer μ exceeding

[1] Automata hereafter defined are sometimes, and perhaps more appropriately, called *boolean automata* (see, e.g., [6, 17]).

a certain value. This enables us to prove that *the language accepted by A forms an ultimately periodic set of period $\ell = \mathrm{lcm}(\ell_1, \ldots, \ell_r)$,* where the ℓ_i's depend on the cycle structure of the transition graph of A and their sum does not exceed n (Theorem 3.5).

By using a number theoretical result that will be recalled in Lemma 2.1, we get that $\ell = O(\mathrm{e}^{\sqrt{n \ln n}})$. So, the number of states in the cycle of a 1dfa A' simulating A is $O(\mathrm{e}^{\sqrt{n \ln n}})$, and this actually turns out to be an upper bound on the number of all states of A'.

As the reader will notice when we get into proof details, the study of path structure in certain weighted digraphs representing automata transitions will be crucial. Such graph theoretical problems are tackled in subsection 2.4 by using some number theoretical tools presented in the next subsection.

2.3. Number theory and Diophantine equations. As usual, we let $\mathbf{Z}^+$ ($\mathbf{Z}^-$) be the set of positive (negative) integers, and $\mathbf{Z}$ be the set of whole integers. Natural numbers are the elements of the set $\mathbf{N} = \mathbf{Z}^+ \cup \{0\}$. The absolute value of z is denoted by $|z|$. For any $z > 0$, $\ln z$ denotes the natural logarithm of z, while $\log z$ is the logarithm of z taken to the base 2. The *greatest common divisor* of integers $a_1, \ldots, a_s$ is denoted by $\gcd(a_1, \ldots, a_s)$. Their *least common multiple* is denoted by $\mathrm{lcm}(a_1, \ldots, a_s)$. Both gcd and lcm are, actually, meant to be taken on $|a_1|, \ldots, |a_s|$. We sometimes write $\gcd(A)$ to denote the greatest common divisor of the integers in the set A.

In estimating the simulation costs of unary automata, a crucial role is played by the function

$$F(n) = \max\{\mathrm{lcm}(x_1, \ldots, x_s) \mid x_1, \ldots, x_s \in \mathbf{Z}^+ \text{ and } x_1 + \cdots + x_s = n\}.$$

Evaluating the growth rate of $F(n)$ is known as Landau's problem [15, 16]. Such a problem is related to the study of the maximal order in the symmetric group S_n [26, 28]. Several approximations for $F(n)$ are given in the literature. The best one is contained in [26, Theorem I] from which we can derive the following upper bound that suffices to our purposes.

LEMMA 2.1. $F(n) = O(\mathrm{e}^{\sqrt{n \ln n}})$.

The other arithmetical tool we shall make use of is the theory of linear Diophantine equations, whose principles can be found, for instance, in [21]. We consider equations in the form

$$a_1 x_1 + \cdots + a_s x_s = z, \tag{2.1}$$

where $a_1, \ldots, a_s, z$ are given integers, and $x_1, \ldots, x_s$ are integer variables. It is a very well-known fact that (2.1) has (infinitely many) solutions in integers if and only if $\gcd(a_1, \ldots, a_s)$ divides z.

THEOREM 2.2. *For any given integers $a_1, \ldots, a_s$, the set of integers*

$$\{a_1 x_1 + \cdots + a_s x_s \mid x_1, \ldots, x_s \in \mathbf{Z}\}$$

is exactly the set of all integral multiples of $\gcd(a_1, \ldots, a_s)$.

We are sometimes interested in solving Diophantine equations in *natural numbers*. To this regard, an interesting question, first raised by Frobenius (see [4, 5]), can be stated as follows: given positive integers $a_1, \ldots, a_s$ satisfying $\gcd(a_1, \ldots, a_s) = 1$, what is the greatest number b such that the Diophantine equation $a_1 x_1 + \cdots + a_s x_s = b$ has no solution in natural numbers? For $s = 2$, such b is known to be $(a_1 - 1)(a_2 - 1) - 1$

[4, 5]. For $s \geq 3$, the problem is still open, even though several upper bounds are proved. We will refer to the following one.

THEOREM 2.3 (see [4, 5, 19]). *Let $a_1 < a_2 < \cdots < a_s$ be positive integers with $\gcd(a_1, \ldots, a_s) = 1$. Then, the greatest number b such that the Diophantine equation $a_1x_1 + \cdots + a_sx_s = b$ has no solution in natural numbers does not exceed $(a_1 - 1)(a_s - 1)$.*

More accurate estimations can be found in [9]. However, the bound in Theorem 2.3 will suffice for our purposes. By combining Theorem 2.2 and Theorem 2.3, one easily obtains the following corollary.

COROLLARY 2.4. *Let $a_1, \ldots, a_s$ be positive integers less than or equal to n, and let*

$$\mathcal{X} = \{a_1x_1 + \cdots + a_sx_s \mid x_1, \ldots, x_s \in \mathbf{N}\}.$$

Then, the set $\mathcal{X} \cap \{z \in \mathbf{Z}^+ \mid z > n^2\}$ is exactly the set of all integral multiples of $\gcd(a_1, \ldots, a_s)$ greater than n^2.

In order to consider sets defined not only by positive coefficients, but even by *both* positive and negative coefficients, we generalize Corollary 2.4 as follows.

LEMMA 2.5. *Let $A = \{a_1, \ldots, a_p\}$ and $B = \{b_1, \ldots, b_q\}$ be sets of positive integers less than or equal to n, with $\gcd(A) = \alpha$ and $\gcd(B) = \beta$. Moreover, let*

$$\mathcal{X} = \{a_1x_1 + \cdots + a_px_p \ - \ (b_1y_1 + \cdots + b_qy_q) \mid x_1, \ldots, x_p\,,\, y_1, \ldots, y_q \in \mathbf{N}\},$$

with $\gcd(A \cup B) = \gcd(\alpha, \beta) = \gamma$.

(a) *If $A = \emptyset$ ($B = \emptyset$), then $\mathcal{X} \cap \{z \in \mathbf{Z}^- \mid z < -n^2\}$ ($\mathcal{X} \cap \{z \in \mathbf{Z}^+ \mid z > n^2\}$) is exactly the set of negative (positive) integral multiples of β (α) smaller than $-n^2$ (greater than n^2).*

(b) *If $A \neq \emptyset$ and $B \neq \emptyset$, then $\mathcal{X}$ is exactly the set of integral multiples of γ.*

Proof. (a) follows trivially from Corollary 2.4. For (b) let us denote by $\mathcal{T}$ the set of integral multiple of γ. It is easy to see that $\mathcal{X} \subseteq \mathcal{T}$. In fact, each number in $\mathcal{X}$ is obtained from the Diophantine equation defining $\mathcal{X}$ itself, and hence it must be a multiple of γ.

Conversely, to show $\mathcal{T} \subseteq \mathcal{X}$, let us first define the sets

$$\mathcal{A} = \{a_1x_1 + \cdots + a_px_p \mid x_1, \ldots, x_p \in \mathbf{N}\},$$
$$\mathcal{B} = \{b_1y_1 + \cdots + b_qy_q \mid y_1, \ldots, y_q \in \mathbf{N}\}.$$

Corollary 2.4 says that $\mathcal{A} \cap \{z \in \mathbf{Z}^+ \mid z > n^2\}$ ($\mathcal{B} \cap \{z \in \mathbf{Z}^+ \mid z > n^2\}$) is exactly the set of positive integral multiples of α (β) greater than n^2. Moreover, it is easy to see that

$$\mathcal{X} = \{h - k \mid h \in \mathcal{A} \text{ and } k \in \mathcal{B}\}. \tag{2.2}$$

So, consider $\xi \in \mathcal{T}$. Since ξ is an integral multiple of $\gamma = \gcd(\alpha, \beta)$, there exist two integers $\overline{x}$ and $\overline{y}$ such that $\xi = \alpha\overline{x} - \beta\overline{y}$. Moreover, it is well known that there are infinitely many solutions of the equation $\xi = \alpha x - \beta y$ that can be obtained from the particular solution $(\overline{x}, \overline{y})$ as follows:

$$\begin{aligned} x &= \overline{x} - \tfrac{\beta}{\gamma}\, t, \\ y &= \overline{y} - \tfrac{\alpha}{\gamma}\, t, \qquad t \in \mathbf{Z}. \end{aligned}$$

At this point, it is easy to find $\hat{t} \in \mathbf{Z}$, giving the solution $(\hat{x}, \hat{y})$, such that both $\alpha\hat{x}$ and $\beta\hat{y}$ are greater than n^2, and hence belong to $\mathcal{A}$ and $\mathcal{B}$, respectively. In other words, we have found two integers $h = \alpha\hat{x} \in \mathcal{A}$ and $k = \beta\hat{y} \in \mathcal{B}$ such that $\xi = h - k$. This together with (2.2) shows that $\xi \in \mathcal{X}$ and completes the proof. □

We end this subsection by showing a further property of solutions of Diophantine equations in natural numbers.

LEMMA 2.6. *Let $a_1, \ldots, a_s$ be positive integers less than or equal to n, and let the integer $z \geq 0$. If the equation $a_1x_1 + a_2x_2 + \cdots + a_sx_s = z$ has a solution in natural numbers, then it also has a solution in natural numbers satisfying*

$$a_2x_2 + \cdots + a_sx_s \leq n^2.$$

Proof. We prove a stronger result, namely, that there exists a solution $x_1, \ldots, x_s$ satisfying $x_2 + \cdots + x_s \leq a_1$. Suppose that $(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_s)$ is a solution in natural numbers, with $\mu = \overline{x}_2 + \cdots + \overline{x}_s > a_1$. Let us form the sequence

$$\underbrace{a_2, a_2, \ldots, a_2}_{\overline{x}_2-\text{times}}, \underbrace{a_3, a_3, \ldots, a_3}_{\overline{x}_3-\text{times}}, \ \ldots\ , \underbrace{a_s, a_s, \ldots, a_s}_{\overline{x}_s-\text{times}}. \tag{2.3}$$

Now, for $i = 1, \ldots, \mu$, let S_i be the sum of the first i members in (2.3). Since the length of the sequence $S_1, \ldots, S_\mu$ exceeds a_1, there must exists a pair of indices $1 \leq i < j \leq \mu$ satisfying $S_i \equiv S_j \pmod{a_1}$, or equivalently, such that $S_j - S_i$ is a multiple of a_1. This enables us to remove from (2.3) the elements forming $S_j - S_i$, consequently augmenting the value of $\overline{x}_1$ by $(S_j - S_i)/a_1$. By iterating this process, we get the claimed result. □

2.4. Paths in directed graphs. Here, we recall some basic notions and prove—by the mathematics above developed—some results on directed graphs. Our interest in this topic is due to the fact that, as we will see in the next section, computation paths of unary automata can be represented as paths in suitable directed graphs. Few elementary notions of graph theory are required and summarized below. For more details, we refer the reader to any of the standard text on graph theory such as, e.g., [1].

Let $G = (V, E)$ be a directed graph or *digraph*, with V the set of vertices, and $E \subseteq V \times V$ the set of arcs. An *oriented path* $\mathcal{P}$ in G is a sequence of vertices $\mathcal{P} = v_0, v_1, \ldots, v_n$ where, for $i = 1, \ldots, n$, $(v_{i-1}, v_i) \in E$. Since we will be dealing with digraphs only, the attribute "oriented" will always be intended. The *length* of $\mathcal{P}$ is the number of arcs $\mathcal{P}$ consists of, and is denoted by $|\mathcal{P}|$. The path $\mathcal{P}$ is a *cycle (*or *closed path)* if $v_0 = v_n$. We call *elementary* (or, sometimes, *simple*) any cycle in which no vertex is encountered more than once (except, of course, the initial vertex which is also the terminal vertex).

A *subgraph* of G is any digraph $G' = (V', E')$ satisfying $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. The subgraph G' is said to be *induced by* V', whenever $E' = E \cap (V' \times V')$. The digraph G is *strongly connected* if there exists a path between any two vertices. A *strongly connected component* of G is a subgraph which is strongly connected and not contained in any other strongly connected subgraph.

The following two results concern length and structure of paths in digraphs.

LEMMA 2.7. *Let G be a digraph with n vertices, and let $\mathcal{P}$ be a path of length x which*

(a) *starts from vertex v_1 and ends in vertex v_2,*
(b) *visits* all *vertices in G.*

Then, there also exists a path $\mathcal{P}_0$ of length x_0 which satisfies (a) *and* (b) *and such that*

(c) $x_0 \leq n^2$,

(d) *there exist integers $x_1, x_2, \ldots, x_s \geq 0$ such that $x = x_0 + a_1x_1 + a_2x_2 + \cdots + a_sx_s$, where $a_1, a_2, \ldots, a_s$ are the lengths of the elementary cycles in G.*

Proof. We can factorize the sequence of the $x+1$ vertices visited along $\mathcal{P}$ as a concatenation of n subsequences of vertices $\sigma_1, \sigma_2, \ldots, \sigma_n$, each one starting from a *different* vertex of G. This can be done since $\mathcal{P}$ visits *all* vertices, as required in (b).

Suppose that $x > n^2$. Then, there exists $1 \leq j \leq n$ such that σ_j contains more than n vertices, i.e., $\sigma_j = v_{j_1} v_{j_2} \ldots v_{j_m}$ with $m > n$. A simple pigeonhole argument shows that there must exist $1 \leq h < k \leq m$ such that $v_{j_h} = v_{j_k}$. This clearly means that, when passing through σ_j, our path $\mathcal{P}$ presents a cycle beginning and ending in $v_{j_h} = v_{j_k}$. Call σ'_j the sequence obtained from σ_j after the elimination of the part corresponding to such a cycle, namely, $\sigma'_j = v_{j_1} v_{j_2} \ldots v_{j_h} v_{j_{k+1}} \ldots v_{j_m}$. The sequence $\sigma_1 \cdots \sigma_{j-1}\, \sigma'_j\, \sigma_{j+1} \cdots \sigma_n$ defines a new path which satisfies (a) and (b) and whose length is strictly less than the length of $\mathcal{P}$. By iterating such a path-compression, we eventually obtain a path $\mathcal{P}_0$ of length x_0 satisfying (a), (b), and (c).

In conclusion, it is not hard to see that the whole process can be carried on by eliminating *elementary cycles*. Thus, one can easily express the length x of $\mathcal{P}$ as in (d), where, for $i = 1, \ldots, s$, x_i denotes the number of times elementary cycles of length a_i have been deleted. □

THEOREM 2.8. *Let G be a digraph with n vertices, and let $\mathcal{P}$ be a path from vertex v_1 to vertex v_2. Then, there also exists a path $\mathcal{P}'$ from v_1 to v_2 with the same length as $\mathcal{P}$ which consists of*

(a) *an initial path $\mathcal{P}_1$,*

(b) *an elementary cycle $\mathcal{P}_2$, with $1 \leq |\mathcal{P}_2| \leq n$, which is repeated λ times,*

(c) *a final path $\mathcal{P}_3$,*

satisfying $|\mathcal{P}_1| + |\mathcal{P}_3| \leq 2n^2$. (Hence, notice that $|\mathcal{P}'| = |\mathcal{P}| = |\mathcal{P}_1| + \lambda\,|\mathcal{P}_2| + |\mathcal{P}_3|$.)

Proof. Without loss of generality, we suppose that $\mathcal{P}$ visits all vertices of G. Otherwise, we consider the subgraph of G induced by the set of vertices visited along $\mathcal{P}$. The new path $\mathcal{P}'$ can be obtained by suitably "reorganizing"—in the light of our result on Diophantine equations in Lemma 2.6—the way of $\mathcal{P}$ through its elementary cycles.

The first step amounts to eliminate some of the elementary cycles in $\mathcal{P}$ to obtain the path $\mathcal{P}_0$, as explained in Lemma 2.7. The total number of arcs eliminated in this step can be expressed as

$$|\mathcal{P}| - |\mathcal{P}_0| = a_1x_1 + a_2x_2 + \cdots + a_sx_s \tag{2.4}$$

for some integers $x_1, x_2, \ldots, x_s \geq 0$, as one may easily check from Lemma 2.7(d). Now, notice that the a_i's in (2.4) are positive integers not exceeding n, being lengths of elementary cycles in a digraph of n vertices. Hence, from Lemma 2.6, we know that there exist integers $\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_s \geq 0$ which are solutions for (2.4), and such that

$$a_2\overline{x}_2 + \cdots + a_s\overline{x}_s \leq n^2. \tag{2.5}$$

All this leads us to construct the path $\mathcal{P}'$ by "padding" $\mathcal{P}_0$ with $\overline{x}_i$ consecutive repetitions of an elementary cycle of length a_i for $i = 1, \ldots, s$. Let $\mathcal{P}_2$ be an elementary cycle of length a_1, and let $\mathcal{P}_1$ ($\mathcal{P}_3$) be the part of $\mathcal{P}'$ which precedes (follows) the $\overline{x}_1$ consecutive repetitions of $\mathcal{P}_2$. Then, we have

- $1 \leq |\mathcal{P}_2| = a_1 \leq n$,

- $\lambda = \overline{x}_1$,
- $|\mathcal{P}_1| + |\mathcal{P}_3| = x_0 + a_2\overline{x}_2 + \cdots + a_s\overline{x}_s \leq 2n^2$, since $x_0 \leq n^2$, from Lemma 2.7(c), and by considering inequality (2.5). □

Let us now turn to *weighted digraphs*. In particular, we are interested in digraphs whose arcs have weights from the set $\{-1, 0, +1\}$. As usual, the *weight of a path* is the sum of weights of the arcs in the path. The following theorem states a periodicity property on cycle weights in strongly connected weighted digraphs.

THEOREM 2.9. *Let $G = (V, E)$ be a strongly connected weighted digraph with n vertices, and let $\{a_1, \ldots, a_s\}$ be the set of weights of all the elementary cycles in G, with $\alpha = \gcd(a_1, \ldots, a_s)$. For each $v \in V$, let $\mathcal{X}_v$ be the set of weights of all the cycles containing v. If there exists at least one $a_i > 0$ ($a_i < 0$), then the set $\mathcal{X}_v$ and the set of positive (negative) integral multiples of α coincide on the elements greater than $2n^2$ (less than $-2n^2$).*

Proof. Since each cycle in G is the sum of elementary cycles, $x \in \mathcal{X}_v$ implies that

$$x = a_1x_1 + \cdots + a_sx_s, \tag{2.6}$$

where, for $i = 1, \ldots, s$, the value $x_i \geq 0$ is the number of times an elementary cycle of weight a_i is passed through to form our cycle of weight x. For what we have recalled in subsection 2.3, this means that x must be an integral multiple of α.

Conversely, if x is an integral multiple of α, with $|x| > n^2$, then, by Lemma 2.5, it can be expressed as in (2.6) for some $x_1, \ldots, x_s \geq 0$. However, such a solution does not necessarily describe a cycle in G. To see this, consider the following example: Suppose we have a digraph with three elementary cycles of weights a_1, a_2, a_3, in which the first and the last cycle are connected *only by the cycle of weight a_2*. Any solution with $x_1, x_2, x_3 > 0$ certainly defines (at least) one cycle in such a digraph, where the ith elementary cycle is passed through x_i times, $i = 1, 2, 3$. On the other hand, a solution with $x_1, x_3 > 0$ *and* $x_2 = 0$ does not define any cycle at all, since we cannot traverse both the first and the third elementary cycle without entering (since, $x_2 = 0$) the second one.

To deal with such kind of connectivity problems, we consider a cycle $\mathcal{C}_0$ of weight x_0 which visits *all* vertices of G. From Lemma 2.7(c), $\mathcal{C}_0$ consists of at most n^2 arcs, and this clearly implies that $|x_0| \leq n^2$. Now, notice that x_0 can be expressed as in (2.6) for some $x_1, \ldots, x_s \geq 0$. Hence, $x_0 = k_0\alpha$ for some $k_0 \in \mathbf{Z}$.

Assume $x = k\alpha$ with $|x| > 2n^2$ for $k \in \mathbf{Z}$, and set $z = x - x_0 = (k - k_0)\alpha$. We have that $|z| = |x - x_0| \geq |x| - |x_0| > n^2$. Since $|a_i| \leq n$ for $i = 1, \ldots, s$, from Lemma 2.5 we get $z = a_1z_1 + \cdots + a_sz_s$, for some $z_1, \ldots, z_s \geq 0$. This enables us to "pad" the cycle $\mathcal{C}_0$ with an amount of z_i elementary cycles of weight a_i for $i = 1, \ldots, s$, in order to obtain a cycle of weight x in G. □

3. Computation paths of 2nfa's and simulation by 1dfa's. In this section, we focus on unary 2nfa's. By using results in subsection 2.4, we show that any unary language $\mathcal{L}$ accepted by a 2nfa with n states *forms an ultimately periodic set of period* $\ell = O(\mathrm{e}^{\sqrt{n \ln n}})$. More precisely, we prove that, for any integer $m > 5n^2$, *the string* 1^m *belongs to* $\mathcal{L}$ *if and only if* $1^{m+\ell}$ *belongs to* $\mathcal{L}$. This immediately leads to a $O(\mathrm{e}^{\sqrt{n \ln n}})$ simulation of unary n-state 2nfa's by 1dfa's. The optimality of such a simulation is a direct consequence of a result in [8].

To state our results, we need some tools to examine computations on unary inputs. Specifically, we refer to three properties that Geffert proved in [11] for two-way nondeterministic Turing machines accepting unary languages in sublogarithmic space. We

are now going to reformulate such properties within the realm of unary 2nfa's. Their validity in this new form comes straightforwardly by observing that finite automata can be clearly seen as Turing machines working in sublogarithmic space. Moreover, we improve the last of these three results. Such an improvement will be particularly useful in the next section where we show that, by just squaring the number of states, it is possible to consider 2nfa's with a very particular structure.

From now on, we will always refer to a *unary 2nfa A with n states*.

The first lemma [11, Lemma 3] says that each computation path of A beginning and ending at the same input square and visiting neither of the endmarkers (*U-Turn*) can be replaced by an equivalent computation path not moving the input head "too far," precisely, no more than n^2 positions to the right (or left).

LEMMA 3.1 (U-Turn). *Suppose there exists a computation path of A on input 1^m where*

(a) *the first state is q_1 with the input head at a position i,*
(b) *the last state is q_2 with the input head at the same position,*
(c) *the input head never moves to the left (right) of the ith position, nor does it visit the right (left) endmarker.*

Then, there also exists a computation path on input 1^m satisfying (a), (b), (c) *and where the input head never moves farther than n^2 positions to the right (left) of the ith position.*

The second lemma [11, Lemma 4] states that we can freely "shift" any computation path that never visits the endmarkers to any position of the input tape, provided that we are sufficiently far from the endmarkers. To our purposes, we give this lemma in a slightly different form. (Recall that the left and right endmarker occupy positions 0 and $m+1$, respectively, on the input tape.)

LEMMA 3.2 (position independence). *Suppose there exists a computation path Π of A on input 1^m beginning in the state q_1 with the input head at a position i, ending in the state q_2 with the input head at a position j, and such that*

(a) $1 \le i < j \le m+1$,
(b) *the left endmarker is never visited along Π,*
(c) *the jth position is reached at the last move only.*

Then, there also exists a computation path beginning in the state q_1 with the input head at a position $i+h$, with $h \in \mathbf{Z}$, ending in the state q_2 with the input head at the position $j+h$ which is reached at the last move only, and satisfying (b), *provided that*

$$n^2 \le i+h < j+h \le m+1.$$

A similar result can be stated for computation paths moving toward left, i.e., with the initial (final) position i (j) satisfying $0 \le j < i \le m$.

Proof. Without loss of generality, from Lemma 3.1, we can assume that Π never visits more than n^2 positions to the left of the ith position. Since the input is unary, we can shift Π within the position bounds stated in the theorem. Notice that the right endmarker can possibly be reached only at the end of the path. □

The third lemma [11, Theorem 1] defines the structure of the computation paths that traverse the entire input, from one endmarker to the other. We provide a new proof of this lemma for unary 2nfa's which is different from the one given in [11]. Our proof makes use of the results on the structure of paths in digraphs in subsection 2.4. As a consequence, we obtain a quadratic estimation of the parameters s_1 and s_2 involved in the lemma; this is to be compared with the $O(n^4)$ upper bound in

[11].[2] Though not essential in the unary 2nfa's by 1dfa's simulation, this improvement will be crucial in the quadratic simulation of unary 2nfa's by 2nfa's having both head reversals and nondeterminism at the endmarkers only (Theorem 4.1).

In what follows, by *loop of length* ℓ, we mean a computation path of the automaton A beginning in a state p with the input head at a position i, and ending in the same state with the input head at the position $i+\ell$.

LEMMA 3.3 (dominant loop). *Suppose the input* 1^m *is traversed from left to right by a computation path* Π *of* A *beginning at the left endmarker in the state* q_1*, ending at the right endmarker in the state* q_2*, and such that the endmarkers are visited only in states* q_1 *and* q_2*. Then,* 1^m *can also be traversed by a computation path beginning in the state* q_1*, ending in the state* q_2*, and in which* A

(a) *having traversed the left endmarker and* s_1 *positions,*
(b) *gets into a loop (called* dominant loop*) of length* ℓ*, which starts from a state* p *and is repeated* λ *times,*
(c) *then traverses the remaining* s_2 *input squares, and finally gets the right endmarker,*

for some s_1*,* s_2*,* ℓ *satisfying*

$$1 \le \ell \le n,$$
$$s_1 + s_2 \le 3n^2.$$

Notice that $m = s_1 + \lambda\ell + s_2$*. The same result holds for computation paths traversing inputs from right to left.*

Proof. If $m \le 3n^2$, the result follows trivially. Hence, suppose $m > 3n^2$. Define the digraph $G = (Q, E)$, where Q is the set of states of A, and $(p, q) \in E$ if and only if there exists a computation path of A which starts from p with the input head at a position $n^2 \le i \le m$, reaches q with the input head at the position $i+1$, and never visits either the endmarkers or the $(i+1)$th input square in the intermediate steps.

Since we are sufficiently (namely, more than n^2 positions) far from the left endmarker, and since computations take place on unary inputs, Lemma 3.2 implies that the digraph G does not depend on the input length. For the same reasons, it is also easy to see that for any path in G of length d starting from the state q' and ending in the state q'', there exists a computation path of A on input 1^m beginning in q' with the input head at a position i, ending in q'' with the input head at the position $i+d$, provided that $n^2 \le i < i+d \le m+1$.

Conversely, for any computation path of A on input 1^m beginning in the state q' with the input head at a position i, ending in q'' with the input head at the position j, and satisfying conditions (a), (b), and (c) in Lemma 3.2, one can easily find a path in G of length $j-i$ joining q' to q''.

Let us subdivide the computation path Π into

- Π_{in} which is the part of Π ending when the (n^2+1)th input square is reached for the first time,
- Π_{fin} which is the remaining part of Π.

For the correspondence between paths in G and computations in A above described, we can associate with Π_{fin} a path $\mathcal{P}$ in G. As shown in Theorem 2.8, $\mathcal{P}$ can be rearranged into a path $\mathcal{P}'$ of the same length as $\mathcal{P}$ made of

[2] At the time the conference version of this paper [20] was completed, the authors learned from Viliam Geffert [13] that he too had obtained, by different arguments, a quadratic upper bound on s_1 and s_2.

- an initial path $\mathcal{P}_1$,
- an elementary cycle $\mathcal{P}_2$, with $1 \leq |\mathcal{P}_2| \leq n$, which is repeated λ times,
- a final path $\mathcal{P}_3$,

satisfying $|\mathcal{P}_1| + |\mathcal{P}_3| \leq 2n^2$. Now, again for the correspondence paths-computations, we can associate with $\mathcal{P}_1$, $\mathcal{P}_2$, and $\mathcal{P}_3$, three computation paths Π_1, Π_2, and Π_3, respectively; in particular, Π_2 is a loop—the *dominant loop*—of length $1 \leq \ell \leq n$.

In conclusion, the input 1^m can be traversed by a computation path of A which starts with Π_{in}, continues with Π_1 which is followed by λ consecutive repetitions of the loop Π_2, and ends with Π_3. Let s be the total amount of cells visited outside the dominant loop (without counting the endmarkers), i.e., during Π_{in}, Π_1, and Π_3. It is easy to see that

$$s \leq n^2 + |\mathcal{P}_1| + |\mathcal{P}_3| \leq 3n^2. \qquad \square$$

From the previous lemma, we learn that loops play an important role in the structure of computations of unary 2nfa's. By using Theorem 2.9, we can show that possible loop lengths follow a very regular pattern which is directly predictable from the structure of unary 2nfa's themselves.

To this purpose, let us consider the weighted digraph $\mathcal{A}$ consisting of the digraph representing the transition diagram of our automaton A after removing transitions on the endmarkers, and in which we set weights $+1$, -1, or 0 to arcs depending on whether they represent transitions where the input head is moved right, left, or kept stationary, respectively. It is straightforward that *any cycle of weight ℓ in $\mathcal{A}$ represents a computation loop of length ℓ in the automaton A.* Hence, by taking into account Theorem 2.9, we get the following lemma.

LEMMA 3.4. *Let p be any given state of A, and let α be the greatest common divisor of weights of the elementary cycles in the strongly connected component of $\mathcal{A}$ containing p. Let $\mathcal{X}_p^+$ ($\mathcal{X}_p^-$) be the set of integers $x > 2n^2$ ($x < -2n^2$) such that the automaton A, starting from the state p with the input head at the ith input square, reaches the $(i+x)$th input square in the same state p, without visiting the endmarkers. If $\mathcal{X}_p^+ \neq \emptyset$ ($\mathcal{X}_p^- \neq \emptyset$), then it is exactly the set of all integral multiples of α greater than $2n^2$ (less than $-2n^2$).*

The following result is crucial in order to obtain our simulations.

THEOREM 3.5. *There exists a set of positive integers $\{\ell_1, \ldots, \ell_r\} \subseteq \{1, \ldots, n\}$ satisfying $\ell_1 + \cdots + \ell_r \leq n$, such that, for any $m \geq n$, if the input 1^m can be traversed from left to right by a computation path of A beginning in the state q_1, ending in the state q_2, and where the endmarkers are visited on the first and last move only, then there exists an index $i \in \{1, \ldots, r\}$ such that, for any $\mu > \frac{5n^2 - m}{\ell_i}$, there is also a computation path from q_1 to q_2 which traverses from left to right the input*

$$1^{m+\mu\ell_i}.$$

The same result holds for computation paths traversing inputs from right to left.

Proof. Again, consider the weighted digraph $\mathcal{A}$ associated with the automaton A. Suppose $\mathcal{A}$ has r strongly connected components and, for $i = 1, \ldots, r$, denote by ℓ_i the greatest common divisor of weights of the elementary cycles in the ith strongly connected component. Clearly, ℓ_i cannot exceed n, and since strongly connected components partition the set of vertices of $\mathcal{A}$, we get $\ell_1 + \cdots + \ell_r \leq n$.

Given the left-to-right traversing of input 1^m, let s_1, s_2, ℓ, λ, p, be as in Lemma 3.3, so to have the "decomposition" $m = s_1 + \lambda\ell + s_2$, and p being the state the dominant

loop starts from. Let us suppose that p is contained in the ith strongly connected component of $\mathcal{A}$. By Lemma 3.4, the set of integers $x > 2n^2$, for which there exists a path from p to itself visiting neither of the endmarkers and moving the input head x positions right, is exactly the set of the multiples of ℓ_i greater than $2n^2$. This leads us to conclude that, for integers $\eta > \frac{2n^2}{\ell_i}$, there exists a computation from the state q_1 to the state q_2 that completely traverses inputs $1^{s_1+\eta\ell_i+s_2}$, and visits the endmarkers on the first and last move only. It is enough, in fact, to use the original traverse of 1^m, and substitute the λ repetitions of the dominant loop with a single loop of length $\eta\ell_i$.

Now, by choosing $\mu > \frac{5n^2-m}{\ell_i}$ and suitably setting $\eta = \mu + \frac{\lambda\ell}{\ell_i}$, we get

$$\eta > \frac{5n^2-m}{\ell_i} + \frac{\lambda\ell}{\ell_i} = \frac{5n^2-(s_1+\lambda\ell+s_2)+\lambda\ell}{\ell_i} = \frac{5n^2-(s_1+s_2)}{\ell_i} \geq \frac{2n^2}{\ell_i},$$

where the last inequality is obtained by recalling that $s_1 + s_2 \leq 3n^2$, as stated in Lemma 3.3. Thus, for such η's, we obtain a complete traversing of inputs

$$1^{s_1+\eta\ell_i+s_2} = 1^{s_1+\left(\mu+\frac{\lambda\ell}{\ell_i}\right)\ell_i+s_2} = 1^{m+\mu\ell_i},$$

provided that $\mu > \frac{5n^2-m}{\ell_i}$. □

The technique used in the proof of the following theorem can be regarded as a refinement of the well-known $n \to n + n!$ pumping technique [18]. The result in Theorem 3.5 enables us to suitably pump and compress unary strings in order to show that unary languages accepted by n-state 2nfa's form ultimately periodic sets of period $O(\mathrm{e}^{\sqrt{n \ln n}})$.

THEOREM 3.6. *Let $\mathcal{L}$ be a unary language accepted by a n-state 2nfa. Then, there exists a constant $\ell = O(\mathrm{e}^{\sqrt{n \ln n}})$ such that, for any integer $m > 5n^2$,*

$$1^m \in \mathcal{L} \text{ if and only if } 1^{m+\ell} \in \mathcal{L}.$$

Proof. Given our 2nfa A, we let $\{\ell_1, \ldots, \ell_r\}$ be the set defined in Theorem 3.5 and let $\ell = \mathrm{lcm}(\ell_1, \ldots, \ell_r)$. Since we have observed that $\ell_1 + \cdots + \ell_r \leq n$, as a consequence of Lemma 2.1 we have that $\ell = O(\mathrm{e}^{\sqrt{n \ln n}})$.

Compression. Suppose $1^{m+\ell} \in \mathcal{L}$, and let $\mathcal{C}$ be an accepting computation path of A on such an input. Along $\mathcal{C}$, consider $r_0, r_1, \ldots, r_p$, the sequence of all states in which the input head scans either of the endmarkers. For $j = 1, \ldots, p$, the following two possibilities arise:

(i) In both r_{j-1} and r_j, the input head scans the same endmarker. By Lemma 3.1 (U-Turn), we can suppose that this part of the computation can be accomplished by moving no more than n^2 positions away from the endmarker, and hence it can take place on the input 1^m as well, m being greater than $5n^2$.

(i) In r_{j-1} the input head scans one of the two endmarkers, while in r_j it scans the other. By Theorem 3.5, for some $\ell_i \in \{\ell_1, \ldots, \ell_r\}$, we can replace the computation path from r_{j-1} to r_j on input $1^{m+\ell}$ with a computation path from r_{j-1} to r_j on input $1^{m+\ell+\mu\ell_i}$, provided that $\mu > \frac{5n^2-(m+\ell)}{\ell_i}$. Choose $\mu = \mu_i = -\frac{\ell}{\ell_i}$. Since $m > 5n^2$, it is easy to verify that $\mu_i > \frac{5n^2-(m+\ell)}{\ell_i}$. Thus, we get a computation path on the input $1^{m+\ell+\mu_i\ell_i} = 1^m$ which begins and ends in the states r_{j-1} and r_j, respectively, and visits the endmarkers on the first and last move only.

From (i) and (ii), it is easy to conclude that A accepts the input 1^m as well.

Pumping. The proof of the converse is similar: choose $\mu = \mu_i = \frac{\ell}{\ell_i}$ at point (ii). □

As an immediate consequence of Theorem 3.6 we are able to state our main result of this section.

COROLLARY 3.7. *Each unary n-state 2nfa can be simulated by a $O(e^{\sqrt{n \ln n}})$-state 1dfa.*

Proof. Given a unary n-state 2nfa accepting the language $\mathcal{L}$, the state diagram of an equivalent 1dfa consists of a path of no more than $5n^2$ states joined to a single elementary cycle involving $\ell = O(e^{\sqrt{n \ln n}})$ states. The path takes care of the nonperiodic part of $\mathcal{L}$ (i.e., strings of length not exceeding $5n^2$), while the cycle accounts for the periodic part (i.e., strings of length greater than $5n^2$). Final states are placed onto the path by inspecting membership of all strings of length not exceeding $5n^2$. Moreover, the property "$1^m \in \mathcal{L}$ if and only if $1^{m+\ell} \in \mathcal{L}$" stated in Theorem 3.6 allows us to set final states onto the cycle by just testing membership for strings $1^{5n^2+1}, 1^{5n^2+2}, \ldots, 1^{5n^2+\ell}$. □

It is possible to show that this simulation cost cannot be improved. Actually, a stronger result can be stated, which proves the optimality of all $O(e^{\sqrt{n \ln n}})$ bounds in Figure 1.1.

THEOREM 3.8 (see [8, Theorem 6.1]). *For any integer n, there exists a unary 2dfa with n states such that any equivalent 1nfa requires $\Omega(e^{\sqrt{n \ln n}})$ states.*

The trivial simulations of cost n in Figure 1.1 are optimal. In fact, for fixed $n > 0$, consider the single word language $\mathcal{L}_n = \{1^{n-1}\}$. Such a language is clearly accepted by a (minimum) 1dfa with n states. On the other hand, any 2nfa for $\mathcal{L}_n$ cannot have less than n states [3].

For the optimality of the quadratic simulation of unary 1nfa's by 2dfa's we refer the reader to [8, Theorem 6.2]. Thus, to complete Figure 1.1, we are left to examine unary one-way alternation versus determinism and nondeterminism. This is precisely the subject matter of section 5. Before that, we briefly show how to use the results so far proved to confine *both reversals and nondeterminism* to the endmarkers on unary 2nfa's by just paying a quadratic increase of the number of states.

4. Bringing reversals and nondeterminism at the endmarkers.

THEOREM 4.1. *Each unary n-state 2nfa A can be simulated by a $O(n^2)$-state 2nfa A' which performs both input head reversals and nondeterministic choices only when the input head scans the endmarkers.*

Proof. Without loss of generality, and by adding one more state, we can assume that A accepts with the input head parked on the left endmarker. We informally describe how A' simulates the computation of A on a given input 1^m.

In a first scan, A' deterministically checks whether $m \leq 5n^2$ and, if so, accepts if and only if $1^m \in \mathcal{L}(A)$. We can assume that at the end of this phase, which clearly requires $O(n^2)$ states, the input head is still positioned on the left endmarker.

Otherwise, if $m > 5n^2$, the second part of the simulation starts from the initial state of A. We briefly explain what happens when A' simulates the behavior of A from a given state q_1 and with the input head scanning the left endmarker. (A symmetrical simulation for computations of A from the right endmarker can be trivially stated.) A' nondeterministically chooses one of the following operations:

(a) simulation of a "U-Turn,"
(b) simulation of a left-to-right traversal of the input.

Since $m > 5n^2$, by Lemma 3.1 the simulation (a) can take place in one step, regardless the length of the input. Hence, each U-Turn from the left endmarker can be "embedded" in the transition function of A'. More precisely, it can be replaced by one stationary move. This part of the simulation does not require new states.

Now, we describe how to perform simulation (b). (In the following, a mod b denotes the remainder of the integer division of a by b.) Let $\{\ell_1, \ldots, \ell_r\}$ be the set introduced in Theorem 3.5 with respect to the automaton A. Given a ℓ_i, with $i \in \{1, \ldots, r\}$, two integers m', m'' satisfying

- $5n^2 < m' < m''$, and
- $m' \bmod \ell_i = \varrho = m'' \bmod \ell_i$,

and two states q_1, q_2 of A, it is not hard to prove that

> there exists a computation path of A from q_1 to q_2, scanning the input $1^{m'}$ from left to right, and touching the endmarkers on the first and last moves only if and only if there exists an analogous computation path on $1^{m''}$.

In fact, write $m' = h'\ell_i + \varrho$, $m'' = h''\ell_i + \varrho$ for some $0 < h' \leq h''$ and $0 \leq \varrho < \ell_i$. This yields $m' = m'' + \mu\ell_i$, where $\mu = h' - h'' = \frac{m' - \varrho - m'' + \varrho}{\ell_i} > \frac{5n^2 - m''}{\ell_i}$, by recalling that $m' > 5n^2$. Hence, by Theorem 3.5, from the left-to-right scan of $1^{m''}$, we can obtain a similar computation path for $1^{m'}$. The proof of the converse is similar. As a consequence of this observation, we get that, given the starting state q_1 of A and the input length $m > 5n^2$, the set of states reachable by A after traversing from left to right the input 1^m depends only on q_1 and on the set of pairs

$$\{(\ell_i\,,\ m \bmod \ell_i) \mid i = 1, \ldots, r\}.$$

Thus, in simulating a left-to-right traversal of A on 1^m starting from the state q_1 with the input head on the left endmarker, A' performs the following steps:

(i) nondeterministically selects an ℓ_i, with $i \in \{1, \ldots, r\}$,
(ii) scans the input tape, counting the input length m modulo ℓ_i,
(iii) when the input head reaches the right endmarker, nondeterministically selects one of the states associated with the starting state q_1 and the pair $(\ell_i\,,\ m \bmod \ell_i)$.

A' is easily seen to have both input head reversals and nondeterministic choices at the endmarkers only. Furthermore, during such a simulation from q_1, A' must remember the chosen ℓ_i, and the value m mod ℓ_i; this information can be clearly maintained in $\ell_1 + \cdots + \ell_s$ many states. Globally, $\ell_1 + \cdots + \ell_s$ more states are needed for each possible (starting) state, and since $\ell_1 + \cdots + \ell_s \leq n$, as observed in Theorem 3.5, we conclude that this part of the simulation requires $O(n^2)$ states. This completes the proof. □

5. Some remarks on 1afa's versus other models. We end the paper by briefly discussing the cost of simulating unary 1afa's with deterministic and nondeterministic automata, and the cost of the converse simulations. We begin with the simulations between 1afa's and 1dfa's:

- In [7, section 5.1], it is shown that for any n-state 1afa accepting a language $\mathcal{L}$ on an arbitrary alphabet there exists a 2^n-state 1dfa accepting $\mathcal{L}^R = \{x^R \mid x \in \mathcal{L}\}$, where x^R denotes the reversal of the string x. Such a result is easily seen to directly define the cost of simulating unary 1afa's with 1dfa's, since $x = x^R$ for unary strings.
- Conversely, in [17, Theorem 1] it is shown that for any n-state 1dfa accepting $\mathcal{L}$ there exists a $\lceil \log n \rceil$-state 1afa for $\mathcal{L}^R$.

This leads to the following equivalence which is basically contained in Corollary 1 and Corollary 2 of [17].

THEOREM 5.1. *The class of unary languages accepted by n-state 1afa's is exactly the class of unary languages accepted by 2^n-state 1dfa's.*

As an immediate consequence, one also gets the following corollary.

COROLLARY 5.2. *Let $\mathcal{L}$ be a unary language for which the minimum 1dfa has n states. Then, every 1afa accepting $\mathcal{L}$ requires at least $\lceil \log n \rceil$ states.*

The simulation costs between unary 1afa's and 1dfa's above stated are optimal, as pointed out in the following theorem.

THEOREM 5.3. *For any integer n, there exists a unary n-state 1afa (1dfa) such that each equivalent 1dfa (1afa) requires not less than 2^n ($\lceil \log n \rceil$) states.*

Proof.

1afa by 1dfa. We again refer to the single word language $\mathcal{L}_{2^n} = \{1^{2^n-1}\}$ considered at the end of section 3. Such a language can be accepted by a 1dfa with 2^n states and hence, according to Corollary 5.2, by a 1afa with n states. On the other hand, any 2nfa for $\mathcal{L}_{2^n}$ requires not less than 2^n states [3].

1dfa by 1afa. The minimum 1dfa for $\mathcal{L}_n = \{1^{n-1}\}$ has n state. Thus, the results follows from Corollary 5.2. □

Yet, we have the following theorem.

THEOREM 5.4. *Each unary n-state 1afa can be simulated by a 2^n-state 1nfa, 2dfa, or 2nfa. Such a simulation cost is tight in all the three cases.*

Proof. It is enough to recall that simulating unary 1afa's with 1dfa's costs 2^n states, and that 1afa's are exponentially more succinct than 2nfa's (Theorem 5.3 [*1afa by 1dfa*]). □

Thus, as already pointed out in [3], Theorem 5.4 says that 1afa's and 2dfa's are not polynomially equivalent regarding their state complexity, as conjectured in [8].

Our results in section 3 allow us to show that simulating 2nfa's (and hence 1nfa's and 2dfa's) by 1afa's takes a sublinear amount of states.

THEOREM 5.5. *Each unary 2nfa, 1nfa, and 2dfa with n states can be simulated by a $O(\sqrt{n \ln n})$-state 1afa.*

Proof. Clearly, it suffices to show the bound for 2nfa's. By Corollary 3.7, a unary n-state 2nfa can be simulated by a $O(e^{\sqrt{n \ln n}})$-state 1dfa which in turn, by Theorem 5.1, can be simulated by a 1afa with $O(\sqrt{n \ln n})$ states. □

The simulations by 1afa's in Theorem 5.5 are optimal.

THEOREM 5.6. *For any integer n, there exists a unary 1nfa and a unary 2dfa with n states such that each equivalent 1afa requires $\Omega(\sqrt{n \ln n})$ states.*

Proof. By [8, Theorem 4.5], for any n there exists a unary n-state 1nfa A such that each 1dfa recognizing $\mathcal{L}(A)$ requires $\Omega(e^{\sqrt{n \ln n}})$ states. By Corollary 5.2, this implies that each 1afa for $\mathcal{L}(A)$ must have $\Omega(\sqrt{n \ln n})$ states. Furthermore, the language $\mathcal{L}(A)$ is accepted even by a unary n-state 2dfa [8, Theorem 5.2]. □

With these results, we complete proving the results summarized in Figure 1.1, section 1.

Acknowledgments. The authors wish to thank anonymous referees for helpful comments and remarks. Moreover, the authors kindly acknowledge Viliam Geffert for stimulating discussions.

REFERENCES

[1] C. BERGE, *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1985.

[2] P. Berman and A. Lingas, *On the Complexity of Regular Languages in Terms of Finite Automata*, Tech. Report 304, Polish Academy of Sciences, 1977.
[3] J.-C. Birget, *State-complexity of finite-state devices, state compressibility and incompressibility*, Math. Systems Theory, 26 (1993), pp. 237–269.
[4] A. Brauer, *On a problem of partitions*, Amer. J. Math., 64 (1942), pp. 299–312.
[5] A. Brauer and J.E. Shockley, *On a problem of Frobenius*, J. Reine Angew. Math., 211 (1962), pp. 215–220.
[6] J.A. Brzozowski and E. Leiss, *On equations for regular languages, finite automata, and sequential networks*, Theoret. Comput. Sci., 10 (1980), pp. 19–35.
[7] A. Chandra, D. Kozen, and L. Stockmeyer, *Alternation*, J. ACM, 28 (1981), pp. 114–133.
[8] M. Chrobak, *Finite automata and unary languages*, Theoret. Comput. Sci., 47 (1986), pp. 149–158.
[9] P. Erdős and R.L. Graham, *On a linear diophantine problem of Frobenius*, Acta Arith., 37 (1980), pp. 321–331.
[10] A. Fellah, H. Jürgensen, and S. Yu, *Constructions for alternating finite automata*, Internat. J. Comput. Math., 35 (1990), pp. 117–132.
[11] V. Geffert, *Nondeterministic computations in sublogarithmic space and space constructibility*, SIAM J. Comput., 20 (1991), pp. 484–498.
[12] V. Geffert, *Tally version of the Savitch and Immerman-Szelepcsényi theorems for sublogarithmic space*, SIAM J. Comput., 22 (1993), pp. 102–113.
[13] V. Geffert, *private communication*, 1997.
[14] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages, and Computation*, Addison–Wesley, Reading, MA, 1979.
[15] E. Landau, *Uber die maximalordung der permutationen gegebenen grades*, Archiv. der Math. und Phys., 3 (1903), pp. 92–103.
[16] E. Landau, *Handbuch der lehre von der verteilung der primzahlen.* I, Teubner, Leipzig, Berlin, 1909.
[17] E. Leiss, *Succinct representation of regular languages by boolean automata*, Theoret. Comput. Sci., 13 (1981), pp. 323–330.
[18] P. Lewis II, R. Stearns, and J. Hartmanis, *Memory bounds for recognition of context free and context sensitive languages*, in IEEE Conference Record on Switching Circuit Theory and Logical Design, 1965, pp. 191–202.
[19] U. Liubicz, *Bounds for the optimal determinization of nondeterministic autonomic automata*, Sibirskii Matemat. Journal, 2 (1964), pp. 337–355 (in Russian).
[20] C. Mereghetti and G. Pighizzini, *Optimal simulations between unary automata*, in 15th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1373, Springer-Verlag, Berlin, 1998, pp. 139–149.
[21] I. Niven and H.S. Zuckerman, *An Introduction to the Theory of Numbers*, John Wiley & Sons, New York, 1980.
[22] M. Rabin and D. Scott, *Finite automata and their decision problems*, IBM J. Res. Develop, 3 (1959), pp. 114–125. Also in E.F. Moore, *Sequential Machines*, Addison–Wesley, Reading, MA, 1964, pp. 63–91.
[23] W. Sakoda and M. Sipser, *Nondeterminism and the size of two–way finite automata*, in Proceedings 10th ACM Symposium on Theory of Computing, 1978, pp. 275–286.
[24] J.C. Shepherdson, *The reduction of two-way automata to one-way automata*, IBM J. Res. Develop, 3 (1959), pp. 198–200. Also in E.F. Moore, *Sequential Machines*, Addison–Wesley, Reading, MA, 1964, pp. 92–97.
[25] M. Sipser, *Lower bounds on the size of sweeping automata*, J. Comput. System Sci., 21 (1980), pp. 195–202.
[26] M. Szalay, *On the maximal order in S_n and S_n^**, Acta Arith., 37 (1980), pp. 321–331.
[27] A. Szepietowski, *Turing Machines with Sublogarithmic Space*, Lecture Notes in Comput. Sci. 843, Springer-Verlag, New York, 1994.
[28] P. Turan, *Combinatorics, partitions, group theory*, in Colloquio Internazionale sulle Teorie Combinatorie, B. Serge, ed., Accademia Nazionale dei Lincei, Rome, 1976, pp. 181–200.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 1993–2007

AN APPROXIMATION ALGORITHM FOR FEEDBACK VERTEX SETS IN TOURNAMENTS*

MAO-CHENG CAI†, XIAOTIE DENG‡, AND WENAN ZANG§

Abstract. We obtain a necessary and sufficient condition in terms of forbidden structures for tournaments to possess the min-max relation on packing and covering directed cycles, together with strongly polynomial time algorithms for the feedback vertex set problem and the cycle packing problem in this class of tournaments. Applying the local ratio technique of Bar-Yehuda and Even to the forbidden structures, we find a 2.5-approximation polynomial time algorithm for the feedback vertex set problem in any tournament.

Key words. feedback vertex set, tournament, min-max relation, approximation algorithm

AMS subject classifications. 68Q25, 68R10

PII. S0097539798338163

1. Introduction.

Given a digraph with weights on the vertices, a subset of vertices is called a *feedback vertex set* if it intersects every directed cycle in the digraph. The problem of finding a feedback vertex set with the minimum total weight is called the *feedback vertex set problem*, which arises in a variety of applications. In the area of operating systems, the problem of breaking deadlocks can be formulated as a feedback vertex set problem. Other applications can be found in VLSI, manufacturing systems, and so on. As is well known, the feedback vertex set problem is NP-hard. Furthermore, this problem admits no fully polynomial approximation scheme unless $P = NP$ [11]. For general digraphs, this problem is approximable within $O(\log n \log\log n)$ [7, 17], where n is the number of vertices in the input; for planar digraphs, it is approximable within 9/4 [8] by a primal-dual method.

The feedback vertex set problem remains NP-hard even in tournaments [18], where a tournament is an orientation of a complete graph; Speckenmeyer established this NP-hardness using the vertex cover problem as the source problem. It can be shown that Speckenmeyer's reduction is an L-reduction (a concept introduced by Papadimitriou and Yannakakis [14]). Moreover, with this reduction, an instance of the vertex cover problem has a solution of size $\leq k$ if and only if the instance of the corresponding feedback vertex set problem has a solution of size $\leq k$. Thus the feedback vertex set problem in tournaments is a generalization of the vertex cover problem, and any inapproximability result of the vertex cover problem [9] is also valid for the feedback vertex set problem in tournaments.

It is well known that the vertex cover problem is approximable within a factor of 2, which can be achieved by several methods [10], such as the local ratio technique [3],

*Received by the editors May 4, 1998; accepted for publication (in revised form) October 6, 2000; published electronically March 20, 2001.

http://www.siam.org/journals/sicomp/30-6/33816.html

†Institute of Systems Science, Academia Sinica, Beijing 100080, People's Republic of China (caimc@staff.iss.ac.cn). This author was supported in part by the National Natural Science Foundation of China.

‡Department of Computer Science, City University of Hong Kong, Hong Kong, People's Republic of China (deng@cs.cityu.edu.hk). This author was supported partially by a grant from the Research Grants Council of the Hong Kong Special Administrative Region, China (Project CityU 1049/98E) and a research grant from City University of Hong Kong (7001040).

§Department of Mathematics, University of Hong Kong, Hong Kong, People's Republic of China (wzang@maths.hku.hk). This author was supported in part by RGC grant 338/024/0009.

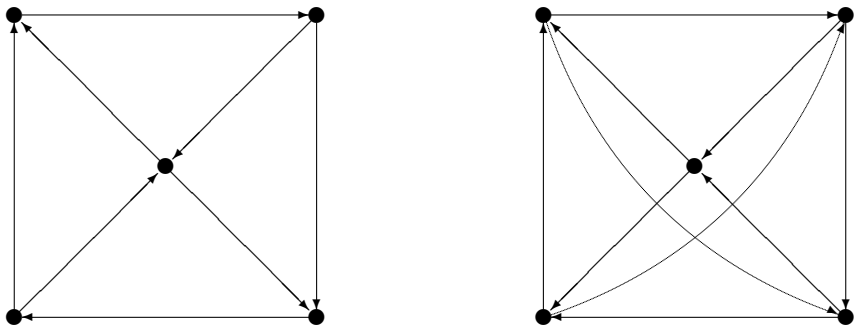
the LP-relaxation method, and the primal-dual method. Despite much recent research effort and progress that have been made in the area of approximation algorithms, the best known approximation ratio of the vertex cover problem is $2-\frac{\log\log n}{2\log n}$ [3, 13], and no approximation algorithm with performance guarantee of $2-\epsilon$ has been discovered so far, no matter how small the positive constant ϵ is. It is thus conjectured in [10] that 2 is the best possible constant. Let us point out that each of those methods mentioned above leads to a 3-approximation algorithm for the feedback vertex set problem in tournaments, which is in the same spirit as the corresponding 2-approximation algorithm for the vertex cover problem. One such algorithm for the unweighted case was first given in [18]. In this work, we improve the approximation ratio to 2.5 for the feedback vertex set problem in tournaments; our approach relies on the local ratio technique of Bar-Yehuda and Even [3] and a characterization of tournaments that possess a min-max relation on packing and covering cycles.

Clearly, a set of vertices in a tournament is a feedback vertex set if and only if it intersects every triangle (a directed cycle of length three, denoted by Δ); thus the feedback vertex set problem is actually the triangle covering problem, which is closely related to the triangle packing problem. Let us now introduce some notions for convenience of presentation.

Given a digraph $T=(V,A)$ such that each vertex $v\in V$ is associated with a nonnegative integer $w(v)$, a *$\triangle$-packing* in T is a family of triangles (repetition is allowed) in T such that each vertex is contained in at most $w(v)$ triangles in this family. A *maximum* $\triangle$-packing in T is a $\triangle$-packing in T with largest size. The *$\triangle$-packing number* of T is the size of a maximum $\triangle$-packing in T. A *$\triangle$-covering* in T is a vertex set $S\subseteq V$ that intersects each triangle in T. The *size* of S, denoted by $w(S)$, is $\sum_{v\in S} w(v)$. A *minimum* $\triangle$-covering in T is a $\triangle$-covering with smallest size; the *$\triangle$-covering number* of T is the size of a minimum $\triangle$-covering in T. The case in which $w(v)=1$ for each $v\in V$ is called *unweighted*; clearly in this case any $\triangle$-packing in T is a family of vertex disjoint triangles of T and the size of any $\triangle$-covering S in T is the number of vertices in S.

Let $\triangle_1,\triangle_2,\ldots,\triangle_m$ be all the triangles in T, let $v_1,v_2,\ldots,v_n$ be all the vertices in V, and let $H_{m\times n}$ be the triangle-vertex incidence matrix, that is, $h_{i,j}=1$ if $\triangle_i$ contains v_j and $h_{i,j}=0$ otherwise. Then the *$\triangle$-covering number* of T equals $\min\{w^Tx \mid Hx\geq e_m,\ x\geq 0,\ \text{integer}\}$, and the *$\triangle$-packing number* of T is $\max\{y^Te_m \mid y^TH\leq w^T,\ y\geq 0,\ \text{integer}\}$, where e_m is the all-one column of size m. It follows from the duality theory of linear programming [16] that the $\triangle$-covering number of T is always greater than or equal to the $\triangle$-packing number of T. The situation in which the packing and covering numbers are equal is particularly interesting. We point out that equality does not necessarily hold in general tournaments: in the unweighted case, both F_1 and F_2 have $\triangle$-packing number of 1 and $\triangle$-covering number of 2. We shall demonstrate that actually F_1 and F_2 are the only obstructions in our problem: if a tournament $T=(V,A)$ contains no F_1 nor F_2, then the $\triangle$-packing number of T always equals the $\triangle$-covering number of T.

The remainder of this paper is organized as follows. In section 2, we give a structural description of tournaments with no F_1 nor F_2. We start with a vertex w with the maximum out-degree, and partition the vertices of T according to their distance from w, that is, $V=\cup_{i=1}^{k}V_i$, where a vertex $v\in V_i$ if and only if the distance from w to v is $i-1$. The subtournament induced by V_i is shown to be acyclic if T contains no F_1 nor F_2. This property leads to a natural order for vertices in V_i. For each $v\in V_{i+1}$, let $V_-(v)$ be the set of vertices in V_i that point to v and let $V_+(v)$ be



$\mathbf{F}_1$ $\mathbf{F}_2$

FIG. 1. *Two Forbidden Subtournaments, where the two arcs not shown in F_1 may take any directions.*

the set of vertices in V_i that are pointed from v. Then every vertex in $V_-(v)$ points to every vertex in $V_+(v)$. Moreover, if $u, v \in V_{i+1}$ with u pointing to v, then every vertex in V_i that points to u also points to v. These properties turn out to be very useful in establishing the fact that, for every triangle in the tournament, its three vertices are in three consecutive subsets of the partition, i.e., V_i, V_{i+1}, V_{i+2} for some i. Then the triangle packing problem becomes the P_3 (a directed path of length 2) packing problem in the digraph D obtained from T by only keeping all the arcs between two consecutive subsets (i.e., V_i and V_{i+1}) of the partition.

In section 3, using the combinatorial structure obtained in section 2, we show that in the unweighted case the P_3-packing number and the P_3-covering number of D are equal if the tournament T is free of subdigraphs F_1 and F_2. To establish the min-max relation, we first show that a particular greedy algorithm for packing P_3 in D results in an optimal solution to the P_3 packing problem. Informally, we prove that the P_3 with the smallest lexicographical (according to the order in section 2) index from V_3 to V_2 to V_1 is in an optimal solution. Then, we show that there is a vertex in this P_3 whose removal reduces the P_3-packing number by one. This implies that both the linear program relaxation $\min\{e_n^t x \mid Hx \geq e_m,\ x \geq 0\}$ and the dual program $\max\{e_n^t y \mid y^t H \leq e_n^t,\ y \geq 0\}$ have integral optimal solutions for every tournament with no F_1 nor F_2 as subdigraph. We further generalize the min-max result to the weighted case.[1]

In section 4, we present a 2.5-approximation polynomial time algorithm for the feedback vertex set problem in any tournament. Applying the local ratio technique to F_1 and F_2, we obtain a 2.5-approximation algorithm for the minimum feedback set problem in any tournament by the local ratio theorem of Bar-Yehuda and Even [3]. We conclude this paper with discussion and remarks in section 5.

2. A structural description. The purpose of this section is to present a structural description of the tournaments with no F_1 nor F_2, which will be used repeatedly in the remaining sections. In our proof, we shall say that u points to v in a digraph, write $u \to v$ if (u, v) is an arc, and we let $N_-(u)$ (resp., $N_+(u)$) stand for the set of all the vertices v with $v \to u$ (resp., $u \to v$).

[1] In the preliminary version [4], this was done by applying a sophisticated TDI technique due to Edmonds and Giles [5, 6, 15]. The present simple proof is suggested by one of the referees.

LEMMA 2.1. *Let $T = (V, A)$ be a strongly connected tournament with no subtournament isomorphic to F_1 nor F_2. Then V can be partitioned into $V_1, V_2, \ldots, V_k$ for some $3 \le k \le |V|$, which have the following properties:*

(a) *For each $i = 1, 2, \ldots, k$, V_i is acyclic and thus V_i admits a linear order $\prec$ such that $x \prec y$ whenever (x, y) is an arc in V_i.*

(b) *For each $i = 1, 2, \ldots, k-1$, there is a map $f : V_{i+1} \to V_i$ such that*
- *for any $v \in V_{i+1}$,*
  (x, v) is an arc for each x in V_i with $x \prec f(v)$ and
  (v, x) is an arc for each x in V_i with $f(v) \preceq x$ and that
- *for any $u, v \in V_{i+1}$ with $u \prec v$, there holds $f(u) \preceq f(v)$.*

(c) *For any i, j with $1 \le i \le j-2 \le k-2$, each arc between V_i and V_j is directed from V_i to V_j.*

Proof. We reserve the symbol w for a vertex in T with *maximum outdegree* throughout the proof. Now let us apply the *breadth-first search* to T and partition the vertices of T as follows.

$V_1 = \{w\}$;
$k = 1$;
while $V - (\cup_{i=1}^{k} V_i) \neq \emptyset$
do $V_{k+1} = \{v \in V - (\cup_{i=1}^{k} V_i) :$ there exists $x \in V_k$ such that $v \to x\}$;
k=k+1;
end

As soon as this algorithm constructs $V_1, V_2, \ldots, V_i$, it proceeds to construct a V_{i+1} if $V - (\cup_{p=1}^{i} V_p) \neq \emptyset$. Since T is strongly connected, $V_{i+1} \neq \emptyset$ for otherwise all the arcs between $\cup_{p=1}^{i} V_p$ and $V - (\cup_{p=1}^{i} V_p)$ are directed to $V - (\cup_{p=1}^{i} V_p)$, a contradiction. Since $V_1, V_2, \ldots$ are pairwise disjoint subsets of V, the algorithm terminates in finite number of steps. It follows that

(2.1) $V_1, V_2, \ldots, V_k$ form a partition of V.

We aim to show that $V_1, V_2, \ldots, V_k$ are as desired. For this purpose, note that (it follows immediately from the algorithm)

(2.2) for each $i = 1, 2, \ldots, k-1$ and each $x \in V_i$, $N_-(x) \cap (V - \cup_{p=1}^{i+1} V_p) = \emptyset$.

Thus property (c) follows. Since T is strongly connected, $N_-(w) \neq \emptyset$ and $N_+(w) \neq \emptyset$, so $V_2 \neq \emptyset$ and $V - (V_1 \cup V_2) \neq \emptyset$. In view of (2.1), we have $k \ge 3$. To prove that $V_1, V_2, \ldots, V_k$ enjoy properties (a) and (b), we apply induction on the subscripts of V_i's.

(2.3) For each $x \in V_2$, we have $N_-(x) \cap V_3 \neq \emptyset$.

To justify it, note that otherwise $d_+(x) > d_+(w)$, contradicting the definition of w.

(2.4) For any $x, y \in V_2$, either $(N_-(x) \cap V_3) \subseteq (N_-(y) \cap V_3)$ or $(N_-(y) \cap V_3) \subseteq (N_-(x) \cap V_3)$.

Assume the contrary: (2.3) guarantees the existence of a vertex u in $(N_-(x) - N_-(y)) \cap V_3$ and a vertex v in $(N_-(y) - N_-(x)) \cap V_3$. Since $uxvyu$ is a cycle of length 4, w points to both u and v (recall (2.2)), and both x and y point to w, $\{u, v, w, x, y\}$ induces an F_1 in T, a contradiction.

Similarly, we can prove that

(2.5) for any $u, v \in V_3$, either $(N_+(u) \cap V_2) \subseteq (N_+(v) \cap V_2)$ or $(N_+(v) \cap V_2) \subseteq (N_+(u) \cap V_2)$.

It follows from (2.3), (2.4), (2.5), and the definition of V_3 that

(2.6) there exists $x \in V_2$ such that $V_3 \subseteq N_-(x)$; there exists $u \in V_3$ such that $V_2 \subseteq N_+(u)$.

The following statement will be used repeatedly in our proof.

(2.7) Let U_1, U_2, and U_3 be three disjoint vertex-subsets of T such that all the arcs between U_i and U_{i+1} are directed to U_{i+1} for each $i = 1, 2, 3$, where the subscript is taken modulo 3. Then each of U_1, U_2, and U_3 is acyclic.

To justify it, assume the contrary: some U_i, say, U_1, contains a triangle $x_1x_2x_3x_1$. Let x_4 be a vertex in U_2 and let x_5 be a vertex in U_3. Then, by hypothesis, x_4 points to x_5; each of x_1, x_2, and x_3 points to x_4; and x_5 points to each of x_1, x_2, and x_3. Thus $x_1x_2x_4x_5x_1$ is a cycle of length 4, x_3 points to both x_1 and x_4, and both x_2 and x_5 point to x_3. Hence $\{x_1, x_2, x_3, x_4, x_5\}$ induces an F_1 in T, a contradiction.

(2.8) Each of V_2 and V_3 is acyclic.

Let $x \in V_2$ and $u \in V_3$ be two vertices as specified in (2.6). By (2.7) with $U_1 = \{w\}$, $U_2 = \{u\}$, and $U_3 = V_2$, we conclude that V_2 is acyclic; by (2.7) with $U_1 = \{w\}$, $U_2 = V_3$, and $U_3 = \{x\}$, we conclude that V_3 is acyclic.

(2.9) Let x, y be two vertices in V_2 and let u, v be two vertices in V_3. Suppose that u points to both x and y, x points to v, and v points to y. Then x points to y and u points to v.

Suppose the contrary, let us distinguish among three cases.

If $y \to x$ and $v \to u$, then $uyxvu$ is a cycle of length 4, both $wvyw$ and $wuxw$ are triangles. Hence $\{u, v, w, x, y\}$ induces an F_2 in T, a contradiction.

If $y \to x$ and $u \to v$, then $uvywu$ is a cycle of length 4, x points to both v and w, and both u and y point to x. Hence $\{u, v, w, x, y\}$ induces an F_1 in T, a contradiction.

If $x \to y$ and $v \to u$, then $uxywu$ is a cycle of length 4, v points to both u and y, and both w and x point to v. Hence $\{u, v, w, x, y\}$ induces an F_1 in T, a contradiction.

(2.10) Let x, y be two vertices in V_2 with $|N_-(x) \cap V_3| < |N_-(y) \cap V_3|$. Then x points to y.

By hypothesis, we have $v \in V_3$ such that x points to v and v points to y. Moreover, (2.6) guarantees the existence of $u \in V_3$ such that u points to both x and y. It follows from (2.9) that x points to y.

(2.11) Let u, v be two vertices in V_3 with $|N_+(v) \cap V_2| < |N_+(u) \cap V_2|$. Then u points to v.

By hypothesis, we have $x \in V_2$ such that u points to x and x points to v. Moreover, (2.6) guarantees the existence of $y \in V_2$ such that both u and v point to y. It follows from (2.9) that u points to v.

It can be seen from (2.8) that V_i admits a linear order $\prec$ such that $x \prec y$ whenever (x, y) is an arc in V_i for each $i = 2, 3$.

(2.12) Let (u, x) be an arbitrary arc from V_3 to V_2. Then $u \to y$ for any y in V_2 with $x \prec y$.

Assume the contrary: $y \to u$ for some y in V_2 with $x \prec y$. By virtue of (2.6), we have $v \in V_3$ such that $v \to x$ and $v \to y$. It follows from (2.9) that $y \to x$, contradicting the hypothesis $x \prec y$.

Since V_1 consists of a single vertex w, property (b) holds trivially for V_1 and V_2.

(2.13) Let $f : V_3 \to V_2$ be the map defined as follows: for any $v \in V_3$, $f(v)$ is the smallest vertex in V_2 such that $v \to f(v)$. Then

- for any $v \in V_3$,
  - (x, v) is an arc for each x in V_2 with $x \prec f(v)$ and
  - (v, y) is an arc for each y in V_2 with $f(v) \preceq y$ and
- for any $u, v \in V_3$ with $u \prec v$, there holds $f(u) \preceq f(v)$.

The first statement follows instantly from (2.12). To justify the second statement, assume the contrary: $f(v) \prec f(u)$ for some u, v in V_3 with $u \prec v$. It follows from

(2.12) that $|N_+(u) \cap V_2| < |N_+(v) \cap V_2|$. By (2.11), we have $v \to u$, contradicting the hypothesis $u \prec v$.

Suppose we have proved that $V_1, V_2, \ldots, V_i$ enjoy properties (a) and (b) for $3 \leq i \leq k-1$; let us proceed to the induction step and consider V_{i+1}. Let $\prec$ be a linear order on V_p as specified in property (a) for $p = 1, 2, \ldots, i$. For convenience, we reserve the symbol s for the largest vertex in V_{i-1} and the symbol t for the largest vertex in V_{i-2}.

(2.14) The following statements hold.

- s points to each vertex in $\{t\} \cup V_{i+1}$;
- t points to each vertex in $V_i \cup V_{i+1}$.

Indeed, by the induction hypothesis of property (b), s points to t and each vertex in V_i points to s; the remaining statements follow from property (c).

(2.15) Let (u, x) be an arbitrary arc from V_{i+1} to V_i. Then $u \to y$ for any y in V_i with $x \prec y$.

Assume the contrary: $y \to u$ for some y in V_i with $x \prec y$. Then $uxstu$ is a cycle of length 4 (recall (2.14)), both t and x point to y, and y points to both s and u. Thus $\{s, t, u, x, y\}$ induces an F_1 in T, a contradiction.

(2.16) V_{i+1} is acyclic.

Let r be the largest vertex in V_i. Then, in view of (2.15) and the definition of V_{i+1}, each vertex in V_{i+1} points to r. By (2.7) with $U_1 = \{r\}$, $U_2 = \{s\}$, and $U_3 = V_{i+1}$, we conclude that V_{i+1} is acyclic.

(2.17) Let u, v be two vertices in V_{i+1} with $|N_+(v) \cap V_i| < |N_+(u) \cap V_i|$. Then u points to v.

Assume the contrary: v points to u. The hypothesis guarantees the existence of $x \in V_i$ such that $u \to x$ and $x \to v$. Let y be the largest vertex in V_i. Then, by (2.15) and the definition of V_{i+1}, we have $u \to y$ and $v \to y$, which implies that $x \neq y$ and hence $x \to y$. Note that $xysux$ is a cycle of length 4, v points to both u and y, and both s and x point to v. Hence $\{s, u, v, x, y\}$ induces an F_1 in T, a contradiction.

According to (2.16), V_{i+1} admits a linear order $\prec$ such that $x \prec y$ whenever (x, y) is an arc in V_{i+1}.

(2.18) Let $f : V_{i+1} \to V_i$ be the map defined as follows: for any $v \in V_{i+1}$, $f(v)$ is the smallest vertex in V_i such that $v \to f(v)$. Then

- for any $v \in V_{i+1}$,
  - (x, v) is an arc for each x in V_i with $x \prec f(v)$ and
  - (v, x) is an arc for each x in V_i with $f(v) \preceq x$ and
- for any $u, v \in V_{i+1}$ with $u \prec v$, there holds $f(u) \preceq f(v)$.

The first statement follows instantly from (2.15). To justify the second statement, assume the contrary: $f(v) \prec f(u)$ for some u, v in V_{i+1} with $u \prec v$. It follows from (2.15) that $|N_+(u) \cap V_2| < |N_+(v) \cap V_2|$. By (2.17), we have $v \to u$, contradicting the hypothesis $u \prec v$. This completes the proof. □

COROLLARY 2.1. *For each $i = 1, 2, \ldots, k-1$, if (v, x) is an arc from V_{i+1} to V_i in T, then (u, x) is an arc for any u in V_{i+1} with $u \prec v$.*

Proof. Since $u \prec v$, by property (b) of the lemma we have $f(u) \preceq f(v) \preceq x$, thus (u, x) is an arc. □

COROLLARY 2.2. *Let $xyzx$ be a triangle in T and let $\{V_1, V_2, \ldots, V_k\}$ be a partition of V as specified in the lemma. Then there exists an i with $1 \leq i \leq k-2$ such that $z \in V_i$, $y \in V_{i+1}$, and $x \in V_{i+2}$ (renaming x, y, and z if necessary).*

Proof. By property (a) of Lemma 2.1, V_i is acyclic for each $i = 1, 2, \ldots, k$. Hence each V_i contains at most two of x, y, and z; let us now verify that V_i cannot contain

two of them. Assume the contrary: both x and y are in V_i. (Rename the vertices if necessary.) By property (c) of the lemma, $z \in V_{i-1}$ or $z \in V_{i+1}$, for otherwise z would have two outgoing or two incoming arcs in the triangle $xyzx$, a contradiction. If $z \in V_{i-1}$, then (since $y \to z$ and $z \to x$) by Corollary 1 we have $y \prec x$, contradicting the fact that $x \to y$; if $z \in V_{i+1}$, then (since $y \to z$ and $z \to x$) by property (b) of the lemma, $y \prec f(z) \preceq x$, contradicting the fact that $x \to y$.

It follows that there exist three subscripts r, s, t with $1 \le r < s < t \le k$ such that $|\{x, y, z\} \cap V_p| = 1$ for each $p = r, s, t$. We claim that $t - r = 2$, for otherwise $t \ge r + 3$, thus either $t \ge s + 2$ (by property (c) of the lemma, the vertex in $\{x, y, z\} \cap V_t$ has two incoming arcs in the triangle $xyzx$) or $s \ge r + 2$ (the vertex in $\{x, y, z\} \cap V_r$ has two outgoing arcs in the triangle), we reach a contradiction in either case, completing the proof. □

LEMMA 2.2. *Let $T = (V, A)$ be a strongly connected tournament. Then* either *one of F_1 and F_2 in T* or *a partition $\{V_1, V_2, \ldots, V_k\}$ of V as described in Lemma* 2.1 *can be found in time $O(|V|^2)$.*

Proof. Let us apply the same algorithm as described in the proof of Lemma 2.1 to T first. This algorithm is essentially a breadth-first search, so it can be implemented in time $O(|V|^2)$. We then need to check if each of (2.4), (2.5), (2.8), (2.10)–(2.12), (2.15)–(2.17) holds. (Recall the proof of Lemma 2.1. The other statements need not be checked; for example, (2.6) follows from (2.4) and (2.5) and (2.9) is proved for (2.10) and (2.11).) If yes, $\{V_1, V_2, \ldots, V_k\}$ is a partition as desired; else, we can exhibit an F_1 or F_2.

Note that (2.4) can be checked in time $O((|V_2| + |V_3|)^2)$. To see it, we first find $N_-(x) \cap V_3$ for each $x \in V_2$; this step takes $O(|V_2||V_3|)$ time. Then sort the vertices in V_2 in nondecreasing order according to $|N_-(x) \cap V_3|$; this step takes $O(|V_2| \log |V_2|)$ time. Suppose $x_1, x_2, \ldots, x_t$ is the resulting order, where $t = |V_2|$. Then we check if $(N_-(x_i) \cap V_3) \subseteq (N_-(x_{i+1}) \cap V_3)$ for $i = 1, 2, \ldots, t-1$. If not, let i be the smallest subscript that violates this condition; then we can exhibit an F_1. (Recall the proof of (2.4).) Otherwise, (2.4) is satisfied; this step takes $O(|V_2||V_3|)$ time. So our statement follows.

Similarly, (2.5) can be checked in time $O((|V_2| + |V_3|)^2)$.

As for (2.8), we can find a triangle in V_2 or declare V_2 is acyclic in time $O(|V_2|^2)$. To see it, let us apply the depth-first search to output the strongly connected components of V_2. If there is no component that contains at least three vertices, then V_2 is acyclic; otherwise, apply the depth-first search on such a component to output a directed cycle C. If C has three vertices, then C is as desired; else, take an arbitrary chord e of C, $\{e\} \cup C$ contains a directed cycle C_1 shorter than C; replace C by C_1; repeat the process.

Similarly, it can be shown that the time complexity for checking each of (2.10)–(2.12), (2.15)–(2.17) is no more than $O((|V_i| + |V_{i+1}|)^2)$ when we proceed to the structure between V_i and V_{i+1} for $i = 1, 2, \ldots, k-1$. Hence, the total complexity is $\sum_{i=1}^{k-1} O((|V_i| + |V_{i+1}|)^2) + O(|V|^2) = O(|V|^2)$.

3. Min-max theorems. The present section is devoted to min-max theorems on packing and covering directed cycles in tournaments. Recall that in the unweighted case every $\triangle$-packing in T is a family of vertex disjoint triangles and the size of a $\triangle$-covering S in T is the number of vertices in S.

THEOREM 3.1. *Let $T = (V, A)$ be a tournament with no subtournament isomorphic to F_1 nor F_2. Then the $\triangle$-packing number of T equals the $\triangle$-covering number of T.*

Proof. Without loss of generality, we may assume that T is strongly connected.

We shall let P_3 stand for an induced directed path with three vertices. Recall the definitions of a $\triangle$-packing and a $\triangle$-covering in section 1; with P_3 in place of $\triangle$ over there, we can define a P_3-packing and a P_3-covering.

Since T contains no F_1 nor F_2, V admits a partition $\{V_1, V_2, \ldots, V_k\}$ as described in Lemma 2.1. Let D be the digraph obtained from T by deleting all arcs in V_i for each i and deleting all the arcs from V_i to V_j for any $i < j$. Then we have

(3.1.1) the $\triangle$-packing number of T equals the P_3-packing number of D; the $\triangle$-covering number of T equals the P_3-covering number of D.

To see it, let $xyzx$ be an arbitrary triangle in T. Then Corollary 2.2 guarantees the existence of some i such that $z \in V_i$, $y \in V_{i+1}$, and $x \in V_{i+2}$. Hence xyz is a P_3 in D. Conversely, if xyz is a P_3 in D, then we have some i (recall the construction of D) such that $z \in V_i$, $y \in V_{i+1}$, and $x \in V_{i+2}$. By property (c) of Lemma 2.1, z points to x in T. Hence $xyzx$ is a triangle in T. So there is a one to one correspondence between triangles in T and $P_3's$ in D; (3.1.1) follows.

In view of (3.1.1), the present theorem is equivalent to the following statement.

(3.1.2) The P_3-packing number of D equals the P_3-covering number of D.

We shall turn to prove (3.1.2). For this purpose, note the following:

(3.1.3) Let $\prec$ be the linear order as defined in Lemma 2.1. Then the following statements hold:

(i) For each $i = 1, 2, \ldots, k-1$, if (v, x) is an arc from V_{i+1} to V_i in D, then (u, x) is an arc in D for any u in V_{i+1} with $u \prec v$.

(ii) For each $i = 1, 2, \ldots, k-1$, there is a map $f : V_{i+1} \to V_i$ such that for each $v \in V_{i+1}$, (v, x) is an arc for each x in V_i with $f(v) \preceq x$ and that there is no arc between v and any x in V_i with $x \prec f(v)$.

From the construction of D, it can be seen that (i) follows instantly from Corollary 2.1 and (ii) follows from property (b) of Lemma 2.1.

Let i^* be the smallest vertex in V_i with respect to the linear order $\prec$ as defined in (3.1.3) for $i = 1, 2, \ldots, k$.

(3.1.4) Without loss of generality, we may assume that $f((i+1)^*) = i^*$ for $i = 1$ and 2.

Suppose the contrary: $f((i+1)^*) \neq i^*$ for $i = 1$ or 2. Then there is no P_3 in D passing through i^*, for otherwise $i \leq 2$ and the construction of D imply that such a P_3 would contain an arc (v, i^*) for some $v \in V_{i+1}$. From (i) of (3.1.3), we conclude that $((i+1)^*, i^*)$ would be an arc in D, so $f((i+1)^*) = i^*$, a contradiction. Hence we may consider $D - \{i^*\}$ instead of D. □

(3.1.5) There exists a maximum P_3-packing in D which contains $3^*2^*1^*$.

To justify (3.1.5), note that, by (3.1.4), $3^*2^*1^*$ is a P_3 in D. Now let $\mathcal{P}$ be a maximum P_3-packing in D such that $|\theta(\mathcal{P}) \cap F^*|$ is as large as possible, where $\theta(\mathcal{P})$ is the set of all the vertices and all the arcs appeared in P_3's in $\mathcal{P}$ and $F^* = \{1^*, 2^*, 3^*, (3^*, 2^*), (2^*, 1^*)\}$. We aim to show that this $\mathcal{P}$ contains $3^*2^*1^*$. To this end, observe that

(i) $\{1^*, 2^*, 3^*\} \cap \theta(\mathcal{P}) \neq \emptyset$. For otherwise, we may add $3^*2^*1^*$ to $\mathcal{P}$ to obtain a larger P_3-packing in D, a contradiction.

(ii) $1^* \in \theta(\mathcal{P})$. For otherwise, in case $2^* \in \theta(\mathcal{P})$, let Q be the P_3 containing 2^* in $\mathcal{P}$ and let Q' be the P_3 obtained from Q by replacing one arc with $(2^*, 1^*)$; in case $2^* \notin \theta(\mathcal{P})$, let Q be the P_3 containing 3^* in $\mathcal{P}$ (recall (i)) and let $Q' = 3^*2^*1^*$. Next, let $\mathcal{P}'$ be the P_3-packing obtained from $\mathcal{P}$ by replacing Q with Q'. Then we have $|\theta(\mathcal{P}') \cap F^*| > |\theta(\mathcal{P}) \cap F^*|$ in each case, a contradiction.

(iii) $2^* \in \theta(\mathcal{P})$. Assume the contrary, let Q be the P_3 containing 1^* in $\mathcal{P}$; we distinguish between two cases: in case 3^* is contained in no $R \in \mathcal{P}$ with $R \neq Q$, let $\mathcal{P}'$ be the P_3-packing obtained from $\mathcal{P}$ by replacing Q with $3^*2^*1^*$. In case 3^* is contained in an $R \in \mathcal{P}$ with $R \neq Q$, we consider two subcases: if 3^* is not a source of R, then let R' be the P_3 obtained from R by replacing one arc with $(3^*, 2^*)$; if 3^* is a source of R, say, $R = 3^*yx$, then set $R' = 3^*2^*x$ (note that $(2^*, x)$ is an arc in D by (3.1.3)); next, let $\mathcal{P}'$ be the P_3-packing obtained from $\mathcal{P}$ by replacing R with R'. It can be seen that $|\theta(\mathcal{P}') \cap F^*| > |\theta(\mathcal{P}) \cap F^*|$ in either case, a contradiction.

(iv) $3^* \in \theta(\mathcal{P})$. For otherwise, let $Q = xy1^*$ be the P_3 containing 1 in $\mathcal{P}$ (recall (ii)). Then $(3^*, y)$ is an arc in D by (i) of (3.1.3) as $3^* \prec x$. Now let $\mathcal{P}'$ be the P_3-packing obtained from $\mathcal{P}$ by replacing Q with 3^*y1^*. Then $|\theta(\mathcal{P}') \cap F^*| > |\theta(\mathcal{P}) \cap F^*|$, a contradiction.

(v) $(3^*, 2^*) \in \theta(\mathcal{P})$. Suppose the contrary: let Q (resp., R) be the P_3 containing 2^* (resp., 3^*) in $\mathcal{P}$ (recall (iii) and (iv)). Then $Q \neq R$. We distinguish between two cases according to the position of 2^* in Q.

Case 1. $Q = x2^*y$. In case $R = 3^*uv$, (x, u) is an arc in D by (ii) of (3.1.3) as $2^* \prec u$, let $Q' = 3^*2^*y$ and $R' = xuv$; in case $R = u3^*v$, both (u, x) and (x, v) are arcs in D according to (3.1.3), let $Q' = 3^*2^*y$ and $R' = uxv$; in case $R = uv3^*$, (v, x) is an arc in D by (3.1.3), let $Q' = 3^*2^*y$ and $R' = uvx$.

Case 2. $Q = xy2^*$. In case $R = 3^*uv$, both $(2^*, v)$ and (y, u) are arcs in D by (3.1.3), let $Q' = 3^*2^*v$ and $R' = xyu$; in case $R = u3^*v$, (y, v) is an arc in D according to (3.1.3), let $Q' = u3^*2^*$ and $R' = xyv$; in case $R = uv3^*$, we consider two subcases: If $v \prec x$, then (u, x) is an arc in D by (3.1.3), let $Q' = v3^*2^*$ and $R' = uxy$; if $x \prec v$, then both $(x, 3^*)$ and (v, y) are arcs in D by (3.1.3), let $Q' = x3^*2^*$ and $R' = uvy$.

Next, in each case let $\mathcal{P}'$ be the P_3-packing obtained from $\mathcal{P}$ by replacing Q with Q' and by replacing R with R'. Then $|\theta(\mathcal{P}') \cap F^*| > |\theta(\mathcal{P}) \cap F^*|$, a contradiction.

(vi) $(2^*, 1^*) \in \theta(\mathcal{P})$. Suppose the contrary: Let Q (resp., $R = yz1^*$) be the P_3 containing $(3^*, 2^*)$ (resp., 1^*) in $\mathcal{P}$ (recall (v) and (ii)). Then $Q \neq R$. In case $Q = 3^*2^*x$, (z, x) is an arc in D by (3.1.3), let $Q' = 3^*2^*1^*$ and $R' = yzx$; in case $Q = x3^*2^*$, (x, y) is an arc in D by (3.1.3), let $Q' = 3^*2^*1^*$ and $R' = xyz$. Next, in each case let $\mathcal{P}'$ be the P_3-packing obtained from $\mathcal{P}$ by replacing Q with Q' and by replacing R with R'. Then $|\theta(\mathcal{P}') \cap F^*| > |\theta(\mathcal{P}) \cap F^*|$, a contradiction.

Since P_3's in $\mathcal{P}$ are vertex disjoint, it follows from (v) and (vi) that $3^*2^*1^*$ is a P_3 in $\mathcal{P}$, completing the proof of (3.1.5).

In view of (3.1.4) and (3.1.5), we have the following *greedy algorithm* for a maximum P_3-packing in D.

(3.1.6) DESCRIPTION. *If $f((i+1)^*) \neq i^*$ for $i = 1$ or 2, then any maximum P_3-packing in $D - \{i^*\}$ is a maximum P_3-packing in D, replace D by $D - \{i^*\}$; else, the union of any maximum P_3-packing in $D - \{1^*, 2^*, 3^*\}$ and $3^*2^*1^*$ gives a maximum P_3-packing in D, replace D by $D - \{1^*, 2^*, 3^*\}$; repeat the process.*

Since (3.1.3) is closed under taking connected induced subdigraphs of D, both (3.1.4) and (3.1.5) are valid with respect to each connected component of new D's. Hence the algorithm will eventually return a maximum P_3-packing in the original D.

Recall (3.1.3): $\prec$ is a linear order defined on each V_i; however, there is no order between any two vertices in two distinct V_i's. Now let us fix this gap and extend $\prec$ to the whole vertex-set V of D.

(3.1.7) Define $u \prec v$ whenever $u \in V_i$ and $v \in V_j$ for any $i < j$.

We point out that if xyz is a P_3 in D, then, according to (3.1.7), there holds $z \prec y \prec x$. Now let us proceed to the order of P_3's in D.

(3.1.8) Let $Q_1 = x_1y_1z_1$ and $Q_2 = x_2y_2z_2$ be two P_3's in D. Define $Q_1 \prec Q_2$ if one of the following three conditions is satisfied: (i) $x_1 \prec x_2$; (ii) $x_1 = x_2$ and $y_1 \prec y_2$; (iii) $x_1 = x_2$, $y_1 = y_2$, and $z_1 \prec z_2$.

Based on (3.1.8), we can further define the order of two P_3-packings with the same size.

(3.1.9) Let $\mathcal{Q} = \{Q_i : i = 1, 2, \ldots, m\}$ and $\mathcal{Q}' = \{Q'_i : j = 1, 2, \ldots, m\}$ be two P_3-packings in D sorted in *increasing* order: $Q_i \prec Q_{i+1}$ and $Q'_i \prec Q'_{i+1}$ for $i = 1, 2, \ldots, m-1$. Define $\mathcal{Q} \prec \mathcal{Q}'$ if there is some i with $1 \leq i \leq m$ such that $Q_i \prec Q'_i$ and $Q_j = Q'_j$ for each $j > i$.

(3.1.10) Let $Q_1 = x_1y_1z_1$ and $Q_2 = x_2y_2z_2$ be two P_3's in D. Define $Q_1 \prec_{strict} Q_2$ if the following two conditions are satisfies simultaneously: (i) $x_1 \prec x_2$, $y_1 \prec y_2$, and $z_1 \prec z_2$; (ii) for each $1 \leq j \leq k$, if $u_1 \in V_j \cap \{x_1, y_1, z_1\}$ and $u_2 \in V_j \cap \{x_2, y_2, z_2\}$, then $u_1 \prec u_2$.

(3.1.11) A maximum P_3-packing $\mathcal{Q} = \{Q_i : i = 1, 2, \ldots, m\}$ in D is called *good* if $Q_1 \prec_{strict} Q_2 \prec_{strict} \ldots \prec_{strict} Q_m$ (renaming Q_i's if necessary).

The algorithm described in (3.1.6) asserts that

(3.1.12) there exists a *good* maximum P_3-packing in D.

Recall that our target is to prove (3.1.2). To achieve it, we still need some preparations.

(3.1.13) Let $\mathcal{Q} = \{Q_i : i = 1, 2, \ldots, m\}$ be a good maximum P_3-packing in D (recall (3.1.11)) with the largest possible order with respect to (3.1.9) and let w be an arbitrary vertex in $\{1^*, 2^*, 3^*\}$. Assume that D and $D - \{w\}$ have the same P_3-packing number. Then there exists a good maximum P_3-packing $\mathcal{Q}' = \{Q'_i : i = 1, 2, \ldots, m\}$ in $D - \{w\}$ such that $Q'_1 \prec Q_1$ and $Q'_i = Q_i$ for $i = 2, 3, \ldots, m$.

To justify (3.1.13), let $\mathcal{Q}' = \{Q'_i : i = 1, 2, \ldots, m\}$ be a good maximum P_3-packing in $D - \{w\}$ with the largest possible order with respect to (3.1.9), the existence of $\mathcal{Q}'$ is guaranteed by (3.1.12) (with $D - \{w\}$ in place of D over there). Let us show that $\mathcal{Q}'$ is as desired. Assume the contrary: let i be the largest index with $Q'_i \prec Q_i$. Then $i \geq 2$. Suppose $Q_i = x_iy_iz_i$ and $Q'_i = x'_iy'_iz'_i$, we distinguish among three cases.

Case 1. $x'_i = x_i$ and $y'_i = y_i$ and $z'_i \prec z_i$. In this case $Q'_i \prec_{strict} Q_i$. Let $\tilde{\mathcal{Q}} = (\mathcal{Q}' - \{Q'_i\}) \cup \{Q_i\}$. Then $\tilde{\mathcal{Q}}$ is a good maximum P_3-packing in $D - \{w\}$. To see it, note that $Q'_j = Q_j$ for each $j > i$. By definition (3.1.11), $Q'_j \prec_{strict} Q'_i$ for each $j < i$, thus no Q'_j with $j < i$ in $\mathcal{Q}'$ passes through any of x_i, y_i, z_i. The statement follows. Since $\mathcal{Q}' \prec \tilde{\mathcal{Q}}$, the existence of $\tilde{\mathcal{Q}}$ contradicts the selection of $\mathcal{Q}'$.

Case 2. $x'_i = x_i$ and $y'_i \prec y_i$. In case $z'_i \preceq z_i$, our proof is exactly the same as that in Case 1; in case $z_i \prec z'_i$, (y_i, z'_i) is an arc in D by (3.1.3). Let $\tilde{\mathcal{Q}} = (\mathcal{Q} - \{Q_i\}) \cup \{x_iy_iz'_i\}$. Then $\tilde{\mathcal{Q}}$ is a good maximum P_3-packing in D with $\mathcal{Q}' \prec \tilde{\mathcal{Q}}$, a contradiction.

Case 3. $x'_i \prec x_i$. Let us consider three subcases.

Subcase 3.1. x'_i and x_i belong to the same V_j for some $1 \leq j \leq k$. In case $y'_i \preceq y_i$, our proof is exactly the same as that in Case 2. So we suppose $y_i \prec y'_i$. Thus (x_i, y'_i) is an arc in D by (3.1.3). Let $\tilde{\mathcal{Q}}$ be the P_3-packing obtained from $\mathcal{Q}'$ by replacing Q'_i with $x_iy'_iz'_i$. Then $\tilde{\mathcal{Q}}$ is a good maximum P_3-packing in $D - \{w\}$ with $\mathcal{Q}' \prec \tilde{\mathcal{Q}}$, contradicting the definition of $\mathcal{Q}'$.

Subcase 3.2. x'_i and y_i belong to the same V_j for some $1 \leq j \leq k$.

Consider the case $x'_i \preceq y_i$. If $y'_i \preceq z_i$ (resp., $z_i \prec y'_i$), let $\tilde{\mathcal{Q}}$ be the P_3-packing obtained from $\mathcal{Q}'$ by replacing Q'_i with Q_i (resp., with $x_iy_iy'_i$, recall (3.1.3)), then $\tilde{\mathcal{Q}}$ is a good maximum P_3-packing in $D - \{w\}$ with $\mathcal{Q}' \prec \tilde{\mathcal{Q}}$, a contradiction.

Next, consider the case $y_i \prec x'_i$. Note that (x_i, x'_i) is an arc in D by (3.1.3). Let

$\tilde{\mathcal{Q}}$ be the P_3-packing obtained from $\mathcal{Q}'$ by replacing Q'_i with $x_i x'_i y'_i$, then $\tilde{\mathcal{Q}}$ is a good maximum P_3-packing in $D - \{w\}$ with $\mathcal{Q}' \prec \tilde{\mathcal{Q}}$, a contradiction.

Subcase 3.3. x'_i and z_i belong to the same V_j for some $1 \leq j \leq k$. Let u be the smaller vertex in $\{x'_i, z_i\}$. Then (y_i, u) is an arc in $D - \{w\}$ by (3.1.3). Let $\tilde{\mathcal{Q}}$ be the P_3-packing obtained from $\mathcal{Q}'$ by replacing Q'_i with $x_i y_i u$, then $\tilde{\mathcal{Q}}$ is a good maximum P_3-packing in $D - \{w\}$ with $\mathcal{Q}' \prec \tilde{\mathcal{Q}}$, a contradiction.

This completes the proof of (3.1.13).

(3.1.14) There exists a $w \in \{1^*, 2^*, 3^*\}$ such that $D - \{w\}$ has less P_3-packing number than D.

To verify (3.1.14), let $\mathcal{Q} = \{Q_i : i = 1, 2, \ldots, m\}$ be a good maximum P_3-packing in D with the largest possible order with respect to (3.1.9). It follows from (3.1.11) that at least one of $1^*, 2^*$, and 3^* is on Q_1, for otherwise we may add $3^*2^*1^*$ to $\mathcal{Q}$ to get a larger good P_3-packing of D, a contradiction. Now let us exhibit w in each of the following cases.

Case 1. 1^* is on Q_1. In this case we may set 1^* as w. To see it, suppose the contrary: (3.1.13) guarantees the existence of a good maximum P_3-packing $\mathcal{Q}' = \{Q'_i : i = 1, 2, \ldots, m\}$ in $D - \{1^*\}$ such that $Q'_1 \prec Q_1$ and $Q'_i = Q_i$ for $i = 2, 3, \ldots, m$. Let $Q_1 = xy1^*$ and let z be the vertex of Q'_1 in V_1. Then (y, z) is an arc in D by (3.1.3). Let $\mathcal{T}$ be the P_3-packing obtained from $\mathcal{Q}$ by replacing Q_1 with xyz, then $\mathcal{T}$ is a good maximum P_3-packing in D with $\mathcal{Q} \prec \mathcal{T}$, a contradiction.

Case 2. 2^* is on Q_1 but neither of 1^* and 3^* is. In this case we may set 2^* as w. To see it, suppose the contrary: (3.1.13) guarantees the existence of a good maximum P_3-packing $\mathcal{Q}' = \{Q'_i : i = 1, 2, \ldots, m\}$ in $D - \{2^*\}$ such that $Q'_1 \prec Q_1$ and $Q'_i = Q_i$ for $i = 2, 3, \ldots, m$. If $Q_1 = x2^*y$, then $Q'_1 \prec Q_1$ implies that $Q'_1 = abc$ for some $a \prec x$ and $2^* \prec b$ with $a \in V_3$. By virtue of (3.1.3), (x, b) is an arc in D; set $R = xbc$. If $Q_1 = xy2^*$, then $Q'_1 \prec Q_1$ implies that Q'_1 contains a vertex z in V_2 with $2^* \prec z$. By (3.1.3), (y, z) is an arc in D; set $R = xyz$. In each case, let $\mathcal{T}$ be the P_3-packing obtained from $\mathcal{Q}$ by replacing Q_1 with R, then $\mathcal{T}$ is a good maximum P_3-packing in D with $\mathcal{Q} \prec \mathcal{T}$, a contradiction.

Case 3. 3^* is on Q_1. In this case we may set 3^* as w. To see it, suppose the contrary: (3.1.13) guarantees the existence of a good maximum P_3-packing $\mathcal{Q}' = \{Q'_i : i = 1, 2, \ldots, m\}$ in $D - \{3^*\}$ such that $Q'_1 \prec Q_1$ and $Q'_i = Q_i$ for $i = 2, 3, \ldots, m$. It follows that 3^* is not the source of Q_1 for otherwise $Q_1 \prec Q'_1$, a contradiction. In case $Q_1 = y3^*z$, let $Q'_1 = abc$. Then $a \in V_3$ or $a \in V_4$ in order for $Q'_1 \prec Q_1$. Set $R = yab$ in the former case (note that (y, a) is an arc in D by (3.1.3)), and set $R = ybc$ in the latter case (note that (y, b) is an arc in D by (3.1.3)). In case $Q_1 = yz3^*$, Q'_1 must contain a vertex a in V_3 in order for $Q'_1 \prec Q_1$, set $R = yza$ (note that (z, a) is an arc in D by (3.1.3)). Now let $\mathcal{T}$ be the P_3-packing obtained from $\mathcal{Q}$ by replacing Q_1 with R. Then $\mathcal{T}$ is a good maximum P_3-packing in D with $\mathcal{Q} \prec \mathcal{T}$ in each case, a contradiction.

This completes the proof of (3.1.14).

Now we are ready to prove (3.1.2).

We apply induction on the number of vertices in D. If D has at most three vertices, the statement is trivial. Let us proceed to the induction step. If $f((i+1)^*) \neq i^*$ for $i = 1$ or 2, then any P_3 in $D - \{i^*\}$ is a P_3 in D (recall (3.1.4)). Thus the desired statement follows from the induction hypothesis on $D - \{i^*\}$. So we suppose $f((i+1)^*) = i^*$ for $i = 1$ and 2. By (3.1.5), $3^*2^*1^*$ is in a maximum P_3-packing in D, so the P_3-packing number of D = the P_3-packing number of $D - \{1^*, 2^*, 3^*\}+1$; by (3.1.14) there exists a vertex w on $3^*2^*1^*$ such that the P_3-covering number of D

= the P_3-covering number of $D-\{w\}+1$, which implies that the P_3-covering number of $D \geq$ the P_3-covering number of $D-\{1^*,2^*,3^*\}+1$. Since by induction hypothesis $D-\{1^*,2^*,3^*\}$ has the same P_3-packing and P_3-covering numbers, the P_3-packing number of $D \geq$ the P_3-covering number of D, so equality must hold and (3.1.2) follows.

This completes the proof of Theorem 3.1.

We further generalize Theorem 3.1 to the weighted case. In the next section we show that it allows us to obtain a 2.5-approximation polynomial time algorithm for the feedback vertex set problem in any tournament.

THEOREM 3.2. *Let $T=(V,A)$ be a tournament with a weight $w(v)$ on each vertex $v \in V$. Then the $\triangle$-packing number of T equals the $\triangle$-covering number for any nonnegative integral w if and only if T contains no F_1 nor F_2.*

Proof. To see the necessity, suppose the contrary: T contains some F_i, $i=1$ or 2. Let w be such that $w(v)=1$ if v is a vertex of F_i and 0 otherwise. Then the $\triangle$-packing (resp., $\triangle$-covering) number of T with respect to w equals the packing (resp., covering) number of F_i in the unweighted case, which is 1 (resp., 2). Hence the min-max relation is violated.

Now let us justify the sufficiency. Suppose T contains no F_1 nor F_2; we aim to establish the min-max result. Without loss of generality, we assume that T is strongly connected and that $w(v)>0$ for each $v \in V$ (for otherwise we may delete it from T).

Let us now construct a new tournament $\tilde{T}$ from T by replacing each vertex v in T with an acyclic subtournament on vertex set $S(v)$ such that $|S(v)|=w(v)$ and that for each $i \in S(u)$ and each j in $S(v)$, (i,j) is an arc in $\tilde{T}$ if and only if (u,v) is an arc in T. It is easy to see that $|S(v) \cap \{i,j,k\}| \leq 1$ for each $S(v)$ and each triangle $ijki$ in $\tilde{T}$. Observe that

(3.2.1) $\tilde{T}$ contains no F_1 nor F_2.

Assume the contrary: $\tilde{T}$ contains some F_k, where $k=1$ or 2. Suppose the vertex set of F_k is $\{i_1,i_2,\ldots,i_5\}$. Since F_k is not a subgraph of T, we may assume the existence of a vertex v in T with $\{i_1,i_2,\ldots,i_5\} \cap S(v) = \{i_1,\ldots,i_j\}$ and $j \geq 2$. From the construction of $\tilde{T}$, it follows that $j=2$ for otherwise $F_k-\{i_{j+1}\}$ contains no triangle, a contradiction; next, there exist two vertices in $\{i_3,i_4,i_5\}$, say, i_3 and i_4, such that the arcs between $\{i_1,i_2\}$ and $\{i_3,i_4\}$ are all directed to $\{i_1,i_2\}$ or all directed to $\{i_3,i_4\}$. Thus $F_k-\{i_5\}$ is acyclic, a contradiction.

By virtue of (3.2.1), we deduce the following statement from Theorem 3.1.

(3.2.2) The $\triangle$-packing number of $\tilde{T}$ equals the $\triangle$-covering number of $\tilde{T}$.

Now let $\tilde{\mathcal{Q}}$ be a maximum $\triangle$-packing in $\tilde{T}$ and let $\tilde{C}$ be a minimum $\triangle$-covering in $\tilde{T}$. We construct a $\triangle$-packing $\mathcal{Q}$ of T from $\tilde{\mathcal{Q}}$ as follows: for each triangle $ijki$ in $\tilde{\mathcal{Q}}$ with $i \in S(a)$, $j \in S(b)$ and $k \in S(c)$, create a triangle $abca$ in $\mathcal{Q}$ (note that a triangle in T may appear multiple times); that is, $\mathcal{Q}$ consist of all the created triangles in T. Then $\mathcal{Q}$ is a $\triangle$-packing of T for each vertex v of T is contained in at most $w(v)$ triangles of $\mathcal{Q}$.

In order to construct a $\triangle$-covering C of T from $\tilde{C}$, observe that $S(v) \subseteq \tilde{C}$ whenever $S(v) \cap \tilde{C} \neq \emptyset$. To see it, assume the contrary: $i \in \tilde{C}$ but $i' \notin \tilde{C}$ for some i and i' in $S(v)$. By the minimality of $\tilde{C}$, there exists a triangle $ijki$ in $\tilde{Q}$ which is covered by a unique vertex, i, in $\tilde{C}$. Thus the triangle $i'jki'$ is not covered by $\tilde{C}$, a contradiction. Now define $C=\{v \in V \mid S(v) \subseteq \tilde{C}\}$. Then it follows from the above observation that C is a $\triangle$-covering of T.

According to (3.2.2), $\sum_{v \in C} w(v) = \sum_{v \in C} |S(v)| = |\tilde{C}| = |\tilde{Q}| = |Q|$. Thus Q is a maximum $\triangle$-packing of T and C is a minimum $\triangle$-covering of T; we therefore get the desired min-max relation.

4. Algorithms. Now we are ready to present an $O(|V|^2)$ algorithm for the maximum $\triangle$-packing problem and an $O(|V|^3)$ algorithm for the minimum $\triangle$-covering problem in any tournament with no F_1 nor F_2.

To find a maximum $\triangle$-packing in T, let us apply the following algorithm, where $\triangle(P(T))$ stands for a maximum $\triangle$-packing in T and $P_3(P(D))$ stands for a maximum P_3-packing in D.

MAXIMUM $\triangle$-PACKING ALGORITHM.

DESCRIPTION. Find all the strongly connected components $T_1, T_2, \ldots, T_s$ of T. If $s \geq 2$, then apply the algorithm on each of $T_1, T_2, \ldots, T_s$, $\triangle(P(T)) = \cup_{i=1}^{s} \triangle(P(T_i))$. Otherwise, let $\{V_1, V_2, \ldots, V_k\}$ be a partition of T as described in Lemma 2.1 and let D be the digraph as constructed in the proof of Theorem 3.1. Find $P_3(P(D))$ as follows: If $f((i+1)^*) \neq i^*$ for $i = 1$ or 2, then $P_3(P(D)) = P_3(P(D - \{i^*\}))$. Replace D by $D - \{i^*\}$; else, set $w(x) = w(x) - \delta$ for each $x \in \{1^*, 2^*, 3^*\}$, where $\delta = \min\{w(1^*), w(2^*), w(3^*)\}$, and set $W = \{v \in V : w(v) = 0\}$. Then $P_3(P(D)) = P_3(P(D - W)) \cup \{3^*2^*1^*, \ldots, 3^*2^*1^*\}$, where the multiplicity of $3^*2^*1^*$ is δ. Replace D by $D - W$; repeat the process. Return $\triangle(P(T)) = P_3(P(D))$.

To show the validity of the algorithm, we need only consider the case $f((i+1)^*) = i^*$ for $i = 1$ and 2. Let $\tilde{D}$ be the digraph obtained from D as follows: each vertex v in D is replaced by a set $S(v)$ of $w(v)$ vertices; for each $i \in S(u)$ and each j in $S(v)$, (i, j) is an arc in $\tilde{D}$ if and only if (u, v) is an arc in D. Clearly, there is a one to one correspondence between maximum *weighted* P_3-packings in D and maximum *unweighted* P_3-packings in $\tilde{D}$. With $\tilde{D}$ in place of D, repeated applications of (3.1.5) guarantee the existence of a maximum P_3-packing in $\tilde{D}$ which contains δ copies of $3^*2^*1^*$. (After getting the first copy, we remove the three vertices on this copy from $\tilde{D}$; then applying (3.1.5) in the resulting digraph, we get the second copy, etc.) Thus the validity of the algorithm follows instantly from the above-mentioned correspondence.

The strongly connected components can be found in time $O(|V|^2)$ by the depth-first search; in case T is strongly connected, the partition $\{V_1, V_2, \ldots, V_k\}$ can be constructed in time $O(|V|^2)$ by the breadth-first search; D can be obtained from the partition in time $O(|V|^2)$; $P_3(P(D))$ can be obtained in time $O(|V|)$. Hence, the total time complexity of the algorithm is $O(|V|^2)$.

To find a minimum covering set, let us apply the following algorithm, where $\triangle(C(T))$ stands for a minimum $\triangle$-covering in T.

MINIMUM $\triangle$-COVERING ALGORITHM.

DESCRIPTION. Find all the strongly connected components $T_1, T_2, \ldots, T_s$ of T. If $s \geq 2$, then apply the algorithm on each of $T_1, T_2, \ldots, T_s$, $\triangle(C(T)) = \cup_{i=1}^{s} \triangle(C(T_i))$. Otherwise, let $\{V_1, V_2, \ldots, V_k\}$ be the partition as described in Lemma 2.1 and let D be the digraph as constructed in the proof of Theorem 3.1. If $f((i+1)^*) \neq i^*$ for $i = 1$ or 2, then $\triangle(C(T)) = \triangle(C(T - \{i^*\}))$. Replace T by $T - \{i^*\}$ and replace D by $D - \{i^*\}$; else, find an x in $\{1^*, 2^*, 3^*\}$ by the maximum $\triangle$-packing algorithm such that $|\triangle(P(T))| = |\triangle(P(T - \{x\}))| + w(x)$. Set $\triangle(C(T)) = \triangle(C(T - \{x\})) \cup \{x\}$. Replace T by $T - \{x\}$ and replace D by $D - \{x\}$; repeat the process.

To justify the validity of the algorithm, we need to show the existence of an x in $\{1^*, 2^*, 3^*\}$ such that $|\triangle(P(T))| = |\triangle(P(T - \{x\}))| + w(x)$. Let S be a minimum $\triangle$-covering of T. Then S must contain at least one $x \in \{1^*, 2^*, 3^*\}$ as $3^*2^*1^*$ is a P_3 in D, which corresponds to a triangle in T. For this x, $S - \{x\}$ is clearly a minimum $\triangle$-covering of $T - \{x\}$. Thus $w(\triangle(C(T))) = w(\triangle(C(T - \{x\}))) + w(x)$. It follows from the min-max result that $|\triangle(P(T))| = |\triangle(P(T - \{x\}))| + w(x)$ since both T and $T - \{x\}$ have the same $\triangle$-packing and $\triangle$-covering numbers. In addition, we need to

show that if $|\triangle(P(T))| = |\triangle(P(T-\{x\}))|+w(x)$, then x is in a minimum $\triangle$-covering of $T-\{x\}$. This implication is trivial as $w(\triangle(C(T))) = w(\triangle(C(T-\{x\}))) + w(x)$.

Since the time complexity of the maximum $\triangle$-packing algorithm is $O(|V|^2)$, it takes $O(|V|^2)$ to find the desired x. Note that T has $|V|$ vertices, the total complexity of the algorithm is $O(|V|^3)$. The proof is complete. □

Given an arbitrary tournament $T = (V, A)$ with a positive integer $w(v)$ on each vertex $v \in V$, let us now present a 2.5-approximation algorithm for the minimum $\triangle$-covering problem in T, which relies on "eliminating" the problematic subdigraphs, F_1 and F_2, from T.

Approximation $\triangle$-Covering Algorithm.

Step 0. Set $\tilde{w} = w$.

Step 1. While T contains a subtournament H isomorphic to F_1 or F_2 such that $\tilde{w}(v) > 0$ for each vertex v in H, do: set $\tilde{w}(v) = \tilde{w}(v) - \delta$ for each vertex v in H, where $\delta = \min\{\tilde{w}(v) : v \in V(H)\}$.

Step 2. Set $\triangle(C_0) = \{v \in V : \tilde{w}(v) = 0\}$ and $V_1 = V - \triangle(C_0)$.

Step 3. Let $\triangle(C_1)$ be returned by applying the minimum $\triangle$-covering algorithm on $T(V_1)$ with respect to the weight $\tilde{w}$. Return $\triangle(C) = \triangle(C_0) \cup \triangle(C_1)$.

Since it takes $O(|V|^2)$ time to output an H in Step 1 according to Lemma 2.2, the total complexity for Step 1 is $O(|V|^3)$; as justified in section 4, Step 3 takes $O(|V|^3)$. So the total complexity of our algorithm is $O(|V|^3)$.

Based on the local ratio theorem of Bar-Yehuda and Even [3], we get the following statement.

Theorem 4.1. *The performance guarantee of the above algorithm is* 2.5*; that is, if* $\triangle(C^*)$ *is a minimum* $\triangle$*-covering in* T*, then* $w(\triangle(C)) \leq 2.5\ w(\triangle(C^*))$.

5. Concluding remarks. The feedback vertex problem in tournaments is a generalization of the vertex cover problem. In this work, we have pointed out that each existing method that leads to a 2-approximation algorithm for the latter problem yields a 3-approximation algorithm for the former problem and that the corresponding algorithms are in the same spirit. Although it is hard to improve the approximation ratio of 2 for the vertex cover problem, by characterizing the class of tournaments with the min-max relation on packing and covering directed cycles, we have succeeded in improving the approximation ratio for the feedback vertex set problem from 3 to 2.5, using the local ratio technique.

Recent applications of the local ratio technique are made by Bar-Yehuda to some other problems [1, 2]. It would be interesting to see if the local ratio technique can be applied in a more sophisticated way to improve the approximation ratio for the feedback set problem in tournaments, for example, by combining the methods developed for the triangle packing and covering problems in graphs by Krivelevich [12].

Acknowledgments. The authors are grateful to Dr. Xiaoyun Lu for his stimulating suggestions and helpful discussions. They also wish to thank anonymous referees for their critical comments that have greatly improved the presentation of this paper. One referee was very kind to suggest that the authors adopt the current simple proof of Theorem 4.1 of his/hers.

REFERENCES

[1] R. Bar-Yehuda, *One for the price of two: A unified approach for approximating covering problems*, Algorithmica, 27 (2000), pp. 131–144.

[2] R. Bar-Yehuda, *Using homogeneous weights for approximating the partial covering problem*, in Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, Baltimore, MD, 1999, pp. 71–75.

[3] R. Bar-Yehuda and S. Even, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.

[4] M. Cai, X. Deng, and W. Zang, *A TDI system and its application to approximation algorithms*, in Proceedings of the 39th IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998, pp. 227–233.

[5] J. Edmonds and R. Giles, *A min-max relation for submodular functions on graphs*, Ann. Discrete Math., 1 (1977), pp. 185–204.

[6] J. Edmonds and R. Giles, *Total dual integrality of linear systems*, in Progress in Combinatorial Optimization, W. R. Pulleyblank, ed., Academic Press, New York, 1994, pp. 117–131.

[7] G. Even, J. Naor, B. Schieber, and M. Sudan, *Approximating minimum feedback sets and multi-cuts in directed graphs*, Algorithmica, 20 (1998), pp. 151–174.

[8] M. X. Goemans and D. P. Williamson, *Primal-dual approximation algorithms for feedback problems in planar graphs*, Combinatorica, 18 (1998), pp. 37–59.

[9] J. Håstad, *Some optimal inapproximability results*, in Proceedings of the 29th Annual ACM Symposium on Theory of Computing, 1997, pp. 1–10.

[10] D. S. Hochbaum, *Approximating covering and packing problems: Set cover, vertex cover, independent set, and related problems*, in Approximation Algorithms for NP-Hard Problems, D.S. Hochbaum, ed., PWS Publishing Company, Boston, 1997, pp. 94–143.

[11] V. Kann, *On the Approximability of NP-Complete Optimization Problems*, Ph.D. thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992.

[12] M. Krivelevich, *On a conjecture of tuza about packing and covering of triangles*, Discrete Math., 142 (1995), pp. 281–286.

[13] B. Monien and E. Speckenmeyer, *Ramsey numbers and an approximation algorithm for the vertex cover problem*, Acta Inform., 22 (1985), pp. 115–123.

[14] C.H. Papadimitriou, and M. Yannakakis, *Optimization, approximation and complexity classes*, J. Comput. System Sci., 43 (1991), pp. 425–440.

[15] A. Schrijver, *Total dual integrality from directed graphs, crossing families and sub- and supermodular functions*, in Progress in Combinatorial Optimization, W. R. Pulleyblank, ed., Academic Press, New York, 1994, pp. 315–362.

[16] A. Schrijver, *Theory of Linear and Integer Programming*, John Wiley, New York, 1986.

[17] P. Seymour, *Packing directed circuits fractionally*, Combinatorica, 15 (1995), pp. 281–288.

[18] E. Speckenmeyer, *On feedback problems in digraphs*, in Graph-Theoretic Concepts in Computer Science, Lecture Notes in Comput. Sci. 411, Springer-Verlag, Berlin, 1989, pp. 218–231.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 2008–2035

THE SHORTEST VECTOR IN A LATTICE IS HARD TO APPROXIMATE TO WITHIN SOME CONSTANT*

DANIELE MICCIANCIO†

Abstract. We show that approximating the *shortest vector problem* (in any ℓ_p norm) to within any constant factor less than $\sqrt[p]{2}$ is hard for NP under *reverse unfaithful random* reductions with inverse polynomial error probability. In particular, approximating the *shortest vector problem* is not in RP (random polynomial time), unless NP equals RP. We also prove a proper NP-hardness result (i.e., hardness under deterministic many-one reductions) under a reasonable number theoretic conjecture on the distribution of square-free smooth numbers. As part of our proof, we give an alternative construction of Ajtai's constructive variant of Sauer's lemma that greatly simplifies Ajtai's original proof.

Key words. NP-hardness, shortest vector problem, point lattices, geometry of numbers, sphere packing

AMS subject classifications. 68Q25, 68W25, 11H06, 11P21, 11N25

PII. S0097539700373039

1. Introduction.

Lattices are geometric objects that can be pictorially described as the set of intersection points of an infinite, regular (but not necessarily orthogonal) n-dimensional grid. The rich combinatorial structure of lattices makes them very powerful tools to attack many important problems in mathematics and computer science. In particular, lattices have been used to solve integer programming with finitely many variables [25, 24, 20], factorization of polynomials over the integers [24, 31], low density subset-sum problems [23, 12, 8], and many cryptanalytic problems [32, 17, 13, 7, 6].

Despite the many successful applications of lattice techniques, the most fundamental problems on lattices resisted any attempt to devise polynomial time algorithm to solve them. These are the *shortest vector problem* (SVP) and the *closest vector problem* (CVP). In SVP, given a lattice, one must find the shortest nonzero vector in the lattice (i.e., the intersection point in the grid closest to the origin). CVP is the inhomogeneous counterpart of SVP: given a lattice and a target point (not necessarily in the lattice), find the lattice point closest to the target. Both problems can be defined with respect to any norm, but the Euclidean norm ℓ_2 is the most commonly used.

The first intractability results for lattice problems date back to 1981 when van Emde Boas [34] proved that CVP[1] is NP-hard and conjectured the same for SVP. Since then, the hardness result for CVP was considerably strengthened, proving that even finding approximate solutions to CVP is hard (see section 2 for more information). Despite the similarities between the two problems, progress in proving the hardness of

*Received by the editors May 16, 2000; accepted for publication (in revised form) September 22, 2000; published electronically March 20, 2001. A preliminary version of this paper appeared in the Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science (FOCS '98). This work was done when the author was at MIT, supported in part by DARPA contract DABT 63-96-C-0018.

http://www.siam.org/journals/sicomp/30-6/37303.html

†Department of Computer Science and Engineering, University of California, San Diego, 9500 Gilman Drive, Mail Code 0114, La Jolla, California 92093-0114 (daniele@cs.ucsd.edu).

[1]In this van Emde Boas paper, this problem is called "Nearest Vector," and the name "Closest Vector" is used for SVP in the ℓ_∞ norm.

SVP was much slower. Even for the exact version of this problem, proving the conjectured NP-hardness remained an open problem for almost two decades. Recently, Ajtai [2] proved that the SVP is hard for NP under *reverse unfaithful random* reductions (RUR-reductions for short, see [18]). These are probabilistic reductions that map NO instances to NO instances with probability 1 and YES instances to YES instances with non-negligible probability.[2] Although not a proper NP-hardness result (i.e., hardness for NP under many-one reductions, which would imply that SVP is not in P unless NP = P), hardness under RUR-reductions also gives evidence of the intractability of a problem. In particular, it implies that SVP is not in RP unless NP = RP. (Here RP is the class of decision problems with random polynomial decision algorithms that are always correct on NO instances and "usually" correct on YES instances.) So, Ajtai's result gives the first theoretical evidence that SVP is indeed intractable, resolving (in a probabilistic sense) van Emde Boas' conjecture. In the same paper, Ajtai also remarks that his NP-hardness proof can be adapted to show the hardness of approximating the length of the shortest vector within some very small factor $1 + o(1)$ that rapidly approaches 1 as the dimension of the lattice grows.

In this paper we prove the first nonapproximability result for the shortest vector problem to within some factor bounded away from 1. Namely, we show that (for any ℓ_p norm) approximating SVP within any constant factor less than $\sqrt[p]{2}$ is hard for NP under RUR-reductions. In particular, approximating SVP in the Euclidean norm within any factor less then $\sqrt{2}$ is hard for NP. The error probability of the reduction is polynomially small, i.e., the reduction correctly maps YES instances to YES instances with probability $1 - 1/\operatorname{poly}(n)$ for some polynomial function $\operatorname{poly}(n)$. Moreover, randomness itself is used in a very restricted way and it can be removed under standard computational or number theoretic assumptions. In particular we show that

(i) SVP has no polynomial time approximation algorithm unless the polynomial hierarchy [26, 33] collapses to the second level.

(ii) Approximating SVP is NP-hard (under deterministic many-one reductions) if the following conjecture on the distribution of square-free smooth numbers holds true: for any $\epsilon > 0$ and for all sufficiently large n there exists a square-free polylog-smooth integer in the interval $[n, n + n^{\epsilon}]$, i.e., an integer whose prime factors are all less than $(\lg n)^c$ (for some constant c independent of n) and have exponent one.

The rest of the paper is organized as follows. In section 2 we give an overview of related work. In section 3 we formally define the approximation problems associated to SVP, CVP, and a variant of the latter. In section 4 we prove that SVP is NP-hard to approximate by reduction from the modified CVP using a geometric lemma which is proved in section 5. In section 6 we present deterministic reductions under various computational or number theoretic assumptions. Section 7 concludes with remarks and open problems.

In section 5 we use a combinatorial theorem (Theorem 5.9) similar to a result originally proved by Ajtai in [2]. In the appendix we present a proof of this combinatorial theorem that greatly simplifies Ajtai's original construction.

2. Related work. The complexity of lattice problems has been widely investigated since the early 1980s because of the many connections between these problems and other areas of computer science. Results are usually presented for the Euclidean

[2] In Ajtai's proof, as well in our result, the success probability is in fact $1 - 1/p(n)$ for some polynomial function $p(n)$.

(ℓ_2) norm but can be easily adapted to any ℓ_p norm, or in some cases any norm. Unless otherwise stated, the following results refer to the ℓ_2 norm.

The first polynomial time approximation algorithms for SVP was the celebrated Lenstra–Lenstra–Lovász (LLL) basis reduction algorithm [24] that achieves an approximation factor $2^{O(n)}$ exponential in the dimension n. In [4] Babai showed how to achieve a similar approximation factor for CVP combining LLL with a lattice rounding technique. To date, the best approximation factor for SVP achievable in polynomial time is $2^{O(n(\log\log n)^2/\log n)}$ using Schnorr's block reduction algorithm [29]. In fact, Schnorr gives a hierarchy of basis reduction algorithms that go from polynomial time to exponential time achieving better and better approximation factors. Unfortunately, Schnorr's result is almost ubiquitously cited as a polynomial time algorithm for approximating SVP within a factor $2^{\epsilon n}$ for any fixed $\epsilon > 0$. It turns out, as recently observed by Goldreich and Håstad [14], that one can set ϵ to a slowly decreasing function of n while maintaining the running time polynomial and achieving a slightly subexponential approximation factor $2^{O(n(\log\log n)^2/\log n)}$. A similar approximation factor can be achieved for CVP combining Schnorr's block reduction algorithm with Kannan's reduction [19] from approximate CVP to approximate SVP.

On the complexity side, CVP was proved NP-hard to solve exactly by van Emde Boas in [34]. The first inapproximability results for CVP are due to Arora et al. [3] who proved that CVP is NP-hard to approximate within any constant factor, and quasi NP-hard to approximate within factor $2^{\log^{1-\epsilon} n}$. The latter result is improved to a proper NP-hardness result by Dinur, Kindler, and Sufra in [10], but the proof is much more complicated. Interestingly, CVP remains hard to solve exactly even if the lattice is known in advance and can be arbitrarily preprocessed before the target point is revealed [28].

The NP-hardness of SVP (in the ℓ_2 norm) was conjectured in [34] but remained an open problem for a long time. The first result is due to Ajtai [2] who proved that solving the problem exactly is NP-hard for randomized reductions. Ajtai's result can also be adapted to show the inapproximability of SVP within certain factors $1 + o(1)$ that rapidly approach 1 as the dimension of the lattice grows. In this paper we prove the first NP-hardness result for approximating SVP within factors bounded away from one.

Interestingly, SVP in the ℓ_∞ norm seems to bear much more similarities to CVP than SVP in the ℓ_2 norm. In fact, the NP-hardness of SVP in the ℓ_∞ norm was already proved in [34]. The quasi NP-hardness of approximating SVP within $2^{\log^{0.5-\epsilon} n}$ appeared in [3], and a proper NP-hardness result is proved in [9] using techniques similar to [10].

The unlikelihood of the NP-hardness of approximating SVP and CVP within polynomial factors has also been investigated. In [22], Lagarias, Lenstra, and Schnorr showed that approximating SVP and CVP within factors $O(n)$ and $O(n^{1.5})$ is in coNP. This result is improved by Banaszczyk [5] where both problems are shown in coNP for $O(n)$ approximation factors. These results imply that approximating SVP and CVP within $\Omega(n)$ polynomial factors cannot be NP-hard, unless NP=coNP. Under the stronger assumption that NP is not contained in coAM, Goldreich and Goldwasser [15] show that approximating SVP and CVP within $\Omega(\sqrt{n}/\log n)$ cannot be NP-hard.

Our results are achieved by reducing the approximate SVP from a variant of CVP which was shown NP-hard to approximate in [3]. The techniques we use to reduce CVP to SVP are related to those used in [30, 1] and [2]. In particular all these works use variants of the "prime number lattice" originally defined by Schnorr in [30] to

(empirically) reduce factoring to lattice reduction. The intent in [30] was to find new factoring algorithms and the method is not formally analyzed. In [1] Adleman attempts to give a formal reduction from factoring to the shortest vector problem. Although based on relatively complicated number theoretic conjectures, Adleman's work marks an important step in the study of the hardness of the shortest vector problem because the reduction from factoring to SVP is presented for the first time as theoretical evidence that SVP is hard. Trying to remove the number theoretic conjectures from Adleman's proof, Ajtai had the fundamental intuition that variants of the prime number lattice could be used to reduce any NP problem (not necessarily related to factoring) to SVP. In the breakthrough paper [2] Ajtai uses an enhanced version of the prime number lattice to reduce an NP-complete problem (a variant of subset sum) to SVP.

In this paper we use a variant of the prime number lattice which is much closer to the original lattice introduced by Schnorr. However, it should be noted that the enhanced lattice defined by Ajtai was the starting point of our investigation, and we rediscovered Schnorr's prime number lattice while trying to understand and simplify Ajtai's construction.

In our proof the prime number lattice is for the first time explicitly connected to sphere packing, giving a simpler and more geometric interpretation of the combinatorics underlying the lattice. The simpler and more geometric approach used in this paper allows us to translate some of the techniques to other settings. In fact, similar techniques have been recently used by Dumer, Micciancio, and Sudan [11] to prove similar results for the minimum distance problem for linear codes.

3. Definitions. Let $\mathbb{R}$ and $\mathbb{Z}$ be the sets of the reals and the integers, respectively. The m-dimensional Euclidean space is denoted $\mathbb{R}^m$. A *lattice* in $\mathbb{R}^m$ is the set of all integer combinations $\mathcal{L} = \{\sum_{i=1}^{n} x_i \mathbf{b}_i : x_i \in \mathbb{Z}\}$ of n linearly independent vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ in $\mathbb{R}^m$ $(m \geq n)$. The set of vectors $\mathbf{b}_1, \ldots, \mathbf{b}_n$ is said to form a *basis* of the lattice, and the integer n is called the *rank* of the lattice. A basis can be compactly represented by the matrix $\mathbf{B} = [\mathbf{b}_1 | \ldots | \mathbf{b}_n] \in \mathbb{R}^{m \times n}$ having the basis vectors as columns. The lattice generated by $\mathbf{B}$ is denoted $\mathcal{L}(\mathbf{B})$. Notice that $\mathcal{L}(\mathbf{B}) = \{\mathbf{Bx} : \mathbf{x} \in \mathbb{Z}^n\}$, where $\mathbf{Bx}$ is the usual matrix-vector multiplication.

For any $p \geq 1$, the ℓ_p norm of a vector $\mathbf{x} \in \mathbb{R}^n$ is defined as $\|\mathbf{x}\|_p = \sqrt[p]{\sum x_i^p}$. The following definitions can be given with respect to any norm. Since in the rest of this paper the norm being used will always be clear from the context, we omit explicit references to a norm in order to keep notation simple, but it should be noted that the definitions are norm dependent. The *minimum distance* of a lattice, $\lambda(\mathcal{L})$, is the minimum distance between any two distinct lattice points and equals the length of the shortest nonzero lattice vector:

$$\lambda(\mathcal{L}) = \min\{\|\mathbf{x} - \mathbf{y}\| : \mathbf{x} \neq \mathbf{y} \in \mathcal{L}\} = \min\{\|\mathbf{x}\| : \mathbf{x} \in \mathcal{L}, \mathbf{x} \neq \mathbf{0}\}.$$

For vector $\mathbf{x} \in \mathbb{R}^n$ and set $S \subseteq \mathbb{R}^n$, let $\operatorname{dist}(\mathbf{v}, S) = \min_{\mathbf{w} \in S} \|\mathbf{v} - \mathbf{w}\|$ be the distance between $\mathbf{v}$ and S. For vector $\mathbf{x} \in \mathbb{R}^n$ and real r, let $\mathcal{B}(\mathbf{v}, r) = \{\mathbf{w} \in \mathbb{R}^n : \|\mathbf{v} - \mathbf{w}\| \leq r\}$ be the ball of radius r centered in $\mathbf{v}$.

When discussing computational issues related to lattices, it is customary to assume that the lattices are represented by a basis matrix $\mathbf{B}$ and that $\mathbf{B}$ has integer entries. In order to study the computational complexity of lattice problems, we formulate them in terms of promise problems. A *promise* problem is a generalization of the familiar notion of decision problem. The difference is that in a promise problem not every string is required to be either a YES or a NO instance. Given a string with

the promise that it is either a YES or NO instance, one has to decide which of the two sets it belongs to.

Following [15], we formulate the approximation problems associated with the shortest vector problem and the closest vector problem in terms of the following promise problems.

DEFINITION 3.1 (approximate SVP). *The promise problem* GAPSVP_γ *(where* $\gamma \geq 1$ *is a function of the dimension) is defined as follows. Instances are pairs* $(\mathbf{B}, d)$*, where* $\mathbf{B} \in \mathbb{Z}^{n \times k}$ *is a lattice basis and* d *a positive number such that*

(i) $(\mathbf{B}, d)$ *is a* YES *instance if* $\lambda(\mathbf{B}) \leq d$*, i.e.,* $\|\mathbf{Bz}\| \leq d$ *for some* $\mathbf{z} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$*;*

(ii) $(\mathbf{B}, d)$ *is a* NO *instance if* $\lambda(\mathbf{B}) > \gamma \cdot d$*, i.e.,* $\|\mathbf{Bz}\| > \gamma \cdot d$ *for all* $\mathbf{z} \in \mathbb{Z}^n \setminus \{\mathbf{0}\}$*.*

DEFINITION 3.2 (approximate CVP). *The promise problem* GAPCVP_γ *(where* $\gamma \geq 1$ *is a function of the dimension) is defined as follows. Instances are triples* $(\mathbf{B}, \mathbf{y}, d)$*, where* $\mathbf{B} \in \mathbb{Z}^{n \times k}$ *is a lattice basis,* $\mathbf{y} \in \mathbb{Z}^n$ *a vector, and* d *a positive number such that*

(i) $(\mathbf{B}, \mathbf{y}, d)$ *is a* YES *instance if dist*$(\mathbf{y}, \mathcal{L}(\mathbf{B})) \leq d$*, i.e.,* $\|\mathbf{Bz} - \mathbf{y}\| \leq d$ *for some* $\mathbf{z} \in \mathbb{Z}^n$*;*

(ii) $(\mathbf{B}, \mathbf{y}, d)$ *is a* NO *instance if dist*$(\mathbf{y}, \mathcal{L}(\mathbf{B})) > \gamma \cdot d$*, i.e.,* $\|\mathbf{Bz} - \mathbf{y}\| > \gamma \cdot d$ *for all* $\mathbf{z} \in \mathbb{Z}^n$*.*

The relation between the promise problems above and the corresponding lattice optimization problems is easily explained. On one hand, if one can compute a γ-approximation $d' \in [\lambda(\mathbf{B}), \gamma \cdot \lambda(\mathbf{B})]$ to the length of the shortest nonzero lattice vector, then one can solve GAPSVP_γ by checking whether $d' \leq \gamma \cdot d$ or $d' > \gamma \cdot d$. On the other hand, assume one has a decision oracle O that solves GAPSVP_γ. (By definition, when the input does not satisfy the promise, the oracle can return any answer.) Let $u \in \mathbb{Z}$ be an upper bound to $\lambda(\mathbf{B})$ (for example, let u be the length of any of the basis vectors). Notice that $O(\mathbf{B}, u)$ always returns YES, while $O(\mathbf{B}, 0)$ always returns NO. Using binary search find an integer $d \in \{0, \ldots, u^2\}$ such that $O(\mathbf{B}, \sqrt{d}) = \text{YES}$ and $O(\mathbf{B}, \sqrt{d-1}) = \text{NO}$. Then, $\lambda(\mathbf{B})$ must lie in the interval $[\sqrt{d}, \gamma \cdot \sqrt{d}]$. A similar argument holds for the closest vector problem.

Reductions between promise problems are defined in the obvious way. A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is a reduction from $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ to $(\Sigma_{\text{YES}}, \Sigma_{\text{NO}})$ if it maps YES instances to YES instances and NO instances to NO instances, i.e., $f(\Pi_{\text{YES}}) \subseteq \Sigma_{\text{YES}}$ and $f(\Pi_{\text{NO}}) \subseteq \Sigma_{\text{NO}}$. Clearly any algorithm A to solve $(\Sigma_{\text{YES}}, \Sigma_{\text{NO}})$ can be used to solve $(\Pi_{\text{YES}}, \Pi_{\text{NO}})$ as follows: on input $I \in \Pi_{\text{YES}} \cup \Pi_{\text{NO}}$, run A on $f(I)$ and output the result. Notice that $f(I)$ always satisfies the promise $f(I) \in \Sigma_{\text{YES}} \cup \Sigma_{\text{NO}}$, and $f(I)$ is a YES instance iff I is a YES instance.

We define one last promise problem that will be useful in the sequel. The problem is a modification of GAPCVP in which YES instances are required to have a boolean solution, and in the NO instances the target vector can be multiplied by any nonzero integer.

DEFINITION 3.3 (modified CVP). *The promise problem* GAPCVP'_γ *(where* $\gamma \geq 1$ *is a function of the dimension) is defined as follows. Instances are triples* $(\mathbf{B}, \mathbf{y}, d)$ *where* $\mathbf{B} \in \mathbb{Z}^{n \times k}$ *is a full rank matrix,* $\mathbf{y} \in \mathbb{Z}^n$ *a vector, and* d *a positive number such that*

(i) $(\mathbf{B}, \mathbf{y}, d)$ *is a* YES *instance if* $\|\mathbf{Bz} - \mathbf{y}\| \leq d$ *for some* $\mathbf{z} \in \{0,1\}^n$*;*

(ii) $(\mathbf{B}, \mathbf{y}, d)$ *is a* NO *instance if* $\|\mathbf{Bz} - w\mathbf{y}\| > \gamma \cdot d$ *for all* $\mathbf{z} \in \mathbb{Z}^n$ *and all* $w \in \mathbb{Z} \setminus \{\mathbf{0}\}$*.*

In [3] it is proved that GAPCVP_γ and its variant GAPCVP'_γ are NP-hard for any constant factor $\gamma \geq 1$.

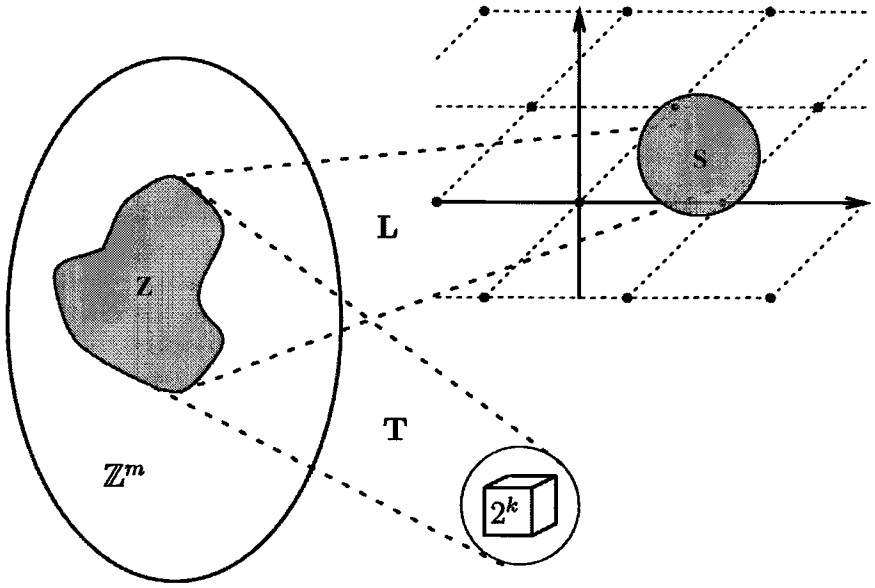


FIG. 4.1. *Lattice $\mathcal{L}(\mathbf{L})$ has minimum distance $\widetilde{\gamma} \approx \sqrt[p]{2}$ times the radius of sphere $\mathcal{B}(\mathbf{s}, r)$ and all boolean vectors of length k can be expressed as $\mathbf{Tz}$ for some lattice vector $\mathbf{Lz}$ inside the sphere.*

4. Hardness of approximating SVP. In this section we present the main result of this paper: for any ℓ_p norm ($p \geq 1$), and for any constant $\gamma \in [1, \sqrt[p]{2})$, the promise problem GAPSVP$_\gamma$ is hard for NP (under RUR-reductions). The proof is by reduction from a variant of the closest vector problem (GAPCVP$'$) and is based on the following simple idea: Assume one wants to find the point in a lattice $\mathcal{L}(\mathbf{B})$ (approximately) closest to some vector $\mathbf{y}$. One may look for the shortest nonzero vector in the lattice generated by the matrix $[\mathbf{B}|\mathbf{y}]$, i.e., the Minkowski sum of the original lattice $\mathcal{L}(\mathbf{B})$ and the lattice $\mathcal{L}(\mathbf{y}) = \mathbb{Z} \cdot \mathbf{y}$ of all integer multiples of the target vector. If the shortest vector in $\mathcal{L}([\mathbf{B}|\mathbf{y}])$ is of the form $\mathbf{Bx} - \mathbf{y}$ then $\mathbf{Bx}$ necessarily is the lattice vector in $\mathcal{L}(\mathbf{B})$ closest to $\mathbf{y}$. However, if the original lattice $\mathcal{L}(\mathbf{B})$ contains vectors as short as the distance of $\mathbf{y}$ from $\mathcal{L}(\mathbf{B})$, then solving the shortest vector problem in the lattice $\mathcal{L}([\mathbf{B}|\mathbf{y}])$ might find a vector of the form $\mathbf{Bx}$, unrelated to the target $\mathbf{y}$. (Notice that the shortest vector in $\mathcal{L}([\mathbf{B}|\mathbf{y}])$ might also correspond to a vector in $\mathcal{L}(\mathbf{B})$ close to a multiple of $\mathbf{y}$, but this is not a problem if we are reducing from GAPCVP$'$.)

We solve this problem by embedding the lattice $\mathcal{L}([\mathbf{B}|\mathbf{y}])$ in a higher dimensional space; i.e., we introduce new coordinates and extend the basis vectors in $[\mathbf{B}|\mathbf{y}]$ with appropriate values. The embedding is based on the construction of a lattice $\mathcal{L}(\mathbf{L})$ and a sphere $\mathcal{B}(\mathbf{s}, r)$ with the property that the minimum distance between lattice points in $\mathcal{L}(\mathbf{L})$ is bigger than the radius r of the sphere (by a constant factor $\widetilde{\gamma} > \gamma$) and at the same time the sphere contains exponentially many lattice vectors from $\mathcal{L}(\mathbf{L})$. We use the lattice points in the sphere to represent all potential solutions to the GAPCVP$'$ problem. In particular, we also build a linear integer transformation $\mathbf{T}$ such that any boolean vector $\mathbf{x} \in \{0,1\}^k$ (i.e., any potential solution to the GAPCVP$'$ problem) can be expressed as $\mathbf{Tz}$ ($\mathbf{z}$ an integer vector) for some lattice point $\mathbf{Lz}$ in the ball $\mathcal{B}(\mathbf{s}, r)$. These requirements are summarized in the following lemma (see Figure 4.1).

LEMMA 4.1. *For any ℓ_p norm ($p \geq 1$) and any constant $\widetilde{\gamma} \in [1, \sqrt[p]{2})$ there exists a (probabilistic) algorithm that on input $k \in \mathbb{Z}^+$ outputs, in* $\mathrm{poly}(k)$ *time, two positive integers $m, r \in \mathbb{Z}^+$, a lattice basis $\mathbf{L} \in \mathbb{Z}^{(m+1)\times m}$, a vector $\mathbf{s} \in \mathbb{Z}^{m+1}$, and a linear integer transformation $\mathbf{T} \in \mathbb{Z}^{k\times m}$ such that*

(i) $\lambda(\mathbf{L}) > \widetilde{\gamma} \cdot r$,

(ii) *with probability at least* $1 - 1/\mathrm{poly}(k)$ *for all $\mathbf{x} \in \{0,1\}^k$ there exists a $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{Tz} = \mathbf{x}$ and $\mathbf{Lz} \in \mathcal{B}(\mathbf{s}, r)$.*

Remark. From the proof of Lemma 4.1 in section 5 it appears that the lemma can be stated in a stronger form asserting the existence of a single algorithm that takes p and $\widetilde{\gamma}$ as additional parameters. However, it should be noted that for every ℓ_p norm, the algorithm can be used only for factors $\widetilde{\gamma} < \sqrt[p]{2}$, and the complexity of the algorithm (e.g., running time or output size) grows to infinity as $\widetilde{\gamma}$ gets closer to $\sqrt[p]{2}$. Stating the result as a single algorithm would require to determine the dependency of the running time on how close $\widetilde{\gamma}$ is to $\sqrt[p]{2}$. In order to keep the notation simple, we will state all the results in this paper for fixed norms ℓ_p and factors γ, but a generalization of the results to variable ℓ_p and γ (subject to the constraint $\gamma < \sqrt[p]{2}$) is indeed possible and the dependency of the running time on p and γ can (in principle) be extracted from the proofs presented in this paper. It should also be noted that as p gets larger and larger, the maximum constant for which we can prove hardness of SVP in the ℓ_p norm approaches 1. This is quite counterintuitive, as we know that SVP in the ℓ_∞ norm is hard to approximate within any constant [3, 9]. So, it is natural to expect that SVP in the ℓ_p norm is harder when p is large.

We defer the proof of the above lemma to section 5 and move straight to the main theorem.

THEOREM 4.2. *For any ℓ_p norm ($p \geq 1$) and for any constant $\gamma \in [1, \sqrt[p]{2})$, the promise problem* GAPSVP_γ *is hard for* NP *under RUR-reductions with inverse polynomial error probability.*

Proof. Fix an ℓ_p norm and a constant $\gamma \in [1, \sqrt[p]{2})$. Let $\widetilde{\gamma}$ be a real between γ and $\sqrt[p]{2}$ and let γ' be a real greater than $(\gamma^{-p} - \widetilde{\gamma}^{-p})^{-1/p}$, and assume without loss of generality that γ'/γ and $\widetilde{\gamma}/\gamma$ are rational numbers. We prove that GAPSVP_γ is hard for NP by reduction from the promise problem $\mathrm{GAPCVP}'_{\gamma'}$ which is known to be NP-hard (see [3]).

Let $(\mathbf{B}, \mathbf{y}, d)$ be an instance of $\mathrm{GAPCVP}'_{\gamma'}$ with $\mathbf{B} \in \mathbb{Z}^{n\times k}$, $\mathbf{y} \in \mathbb{Z}^n$, and $d \in \mathbb{Z}$. We define an instance $(\mathbf{V}, t)$ of GAPSVP_γ such that if $(\mathbf{B}, \mathbf{y}, d)$ is a NO instance of $\mathrm{GAPCVP}'_{\gamma'}$ then $(\mathbf{V}, t)$ is a NO instance of GAPSVP_γ, and if $(\mathbf{B}, \mathbf{y}, d)$ is a YES instance of $\mathrm{GAPCVP}'_{\gamma'}$ then $(\mathbf{V}, t)$ is a YES instance of GAPSVP_γ with high probability.

Run the (randomized) algorithm from Lemma 4.1 on input k to obtain a lattice basis $\mathbf{L} \in \mathbb{Z}^{(m+1)\times m}$, a vector $\mathbf{s} \in \mathbb{Z}^{m+1}$, a linear integer transformation $\mathbf{T} \in \mathbb{Z}^{k\times m}$ and a integer $r \in \mathbb{Z}$ such that

(i) $\|\mathbf{Lz}\|_p > \widetilde{\gamma} \cdot r$ for all $\mathbf{z} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$,

(ii) with probability at least $1 - 1/\mathrm{poly}(k)$, for all vectors $\mathbf{x} \in \{0,1\}^k$ there exists a $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{Tz} = \mathbf{x}$ and $\|\mathbf{Lz} - \mathbf{s}\|_p \leq r$.

Define the lattice

$$\mathbf{V} = \left[\begin{array}{c|c} a \cdot \mathbf{BT} & a \cdot \mathbf{y} \\ b \cdot \mathbf{L} & b \cdot \mathbf{s} \end{array}\right],$$

where a and b are two integer scaling factors such that $\frac{a}{b} = \frac{r\widetilde{\gamma}}{d\gamma'}$ and $t = a \cdot d\gamma'/\gamma = b \cdot r\widetilde{\gamma}/\gamma$ is integer. We want to prove that if $(\mathbf{B}, \mathbf{y}, d)$ is a NO instance of $\mathrm{GAPCVP}'_{\gamma'}$

then $(\mathbf{V}, t)$ is a NO instance of GAPSVP$_\gamma$, and (provided the construction in the lemma succeeds) if $(\mathbf{B}, \mathbf{y}, d)$ is a YES instance of GAPCVP$'_{\gamma'}$ then $(\mathbf{V}, t)$ is a YES instance of GAPSVP$_\gamma$.

First assume $(\mathbf{B}, \mathbf{y}, d)$ is a NO instance and consider a generic nonzero integer vector

$$\mathbf{w} = \begin{bmatrix} \mathbf{z} \\ w \end{bmatrix}.$$

We want to prove that $\|\mathbf{V}\mathbf{w}\|_p^p > (\gamma t)^p$. Notice that

$$\|\mathbf{V}\mathbf{w}\|_p^p = (a \cdot \|\mathbf{B}\mathbf{x} + w\mathbf{y}\|_p)^p + (b \cdot \|\mathbf{L}\mathbf{z} + w\mathbf{s}\|_p)^p,$$

where $\mathbf{x} = C\mathbf{z}$. We prove that

$$a \cdot \|\mathbf{B}\mathbf{x} + w\mathbf{y}\|_p > \gamma t, \qquad \text{or} \qquad b \cdot \|\mathbf{L}\mathbf{z} + w\mathbf{s}\|_p > \gamma t.$$

We distinguish two cases:

1. If $w \neq 0$, then by definition of GAPCVP$'_{\gamma'}$,
$$a \cdot \|\mathbf{B}\mathbf{x} + w\mathbf{y}\|_p > a \cdot \gamma' d = \gamma t;$$
2. if $w = 0$, then $\mathbf{z} \neq 0$ and by construction
$$b \cdot \|\mathbf{L}\mathbf{z} + w\mathbf{s}\|_p = b \cdot \|\mathbf{L}\mathbf{z}\|_p > b \cdot \widetilde{\gamma} r = \gamma t.$$

Now assume that $(\mathbf{B}, \mathbf{y}, d)$ is a YES instance, i.e., there exists a boolean vector $\mathbf{x} \in \{0,1\}^k$ such that $\|\mathbf{B}\mathbf{x} - \mathbf{y}\|_p \leq d$. By construction, there exists a vector $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{T}\mathbf{z} = \mathbf{x}$ and $\|\mathbf{L}\mathbf{z} - \mathbf{s}\|_p \leq r$. Define

$$\mathbf{w} = \begin{bmatrix} \mathbf{z} \\ -1 \end{bmatrix}$$

and compute the norm of the corresponding lattice vector:

$$\begin{aligned} \|\mathbf{V}\mathbf{w}\|_p^p &= (a \cdot \|\mathbf{B}\mathbf{x} - \mathbf{y}\|_p)^p + (b \cdot \|\mathbf{L}\mathbf{z} - \mathbf{s}\|_p)^p \\ &\leq (ad)^p + (br)^p \\ &= \left(\frac{\gamma t}{\gamma'}\right)^p + \left(\frac{\gamma t}{\widetilde{\gamma}}\right)^p \\ &\leq t^p \gamma^p \left(\left(\frac{1}{\gamma^p} - \frac{1}{\widetilde{\gamma}^p}\right) + \frac{1}{\widetilde{\gamma}^p}\right) \\ &= t^p, \end{aligned}$$

proving that $(\mathbf{V}, t)$ is a YES instance of GAPSVP$_\gamma$. ∎

5. Proof of the geometric lemma. In this section we prove Lemma 4.1. As explained in section 4, this lemma asserts the existence of an integer lattice $\mathcal{L}(\mathbf{L})$ with large minimum distance, a sphere $\mathcal{B}(\mathbf{s}, r)$ of radius less than $\lambda(\mathbf{L})$ (by a constant factor $\widetilde{\gamma}$) containing a large number of lattice points, and a linear integer transformation that maps the coordinates (with respect to $\mathbf{L}$) of the lattice points in the sphere onto the set of all binary strings of some shorter length. Moreover, $\mathbf{L}, \mathbf{s}$, and $\mathbf{T}$ can be computed in (random) polynomial time.

The lattice and the sphere are more easily defined using arbitrary real numbers. So, we first drop the requirement that $\mathbf{L}$ and $\mathbf{s}$ have integer entries and define a real matrix $\widetilde{\mathbf{L}}$ and a real vector $\widetilde{\mathbf{s}}$ with the desired properties. Then, we show how to approximate $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{s}}$ with integer matrices. Finally, we prove Lemma 4.1 combining the integer lattice construction with a combinatorial theorem on low-degree hypergraphs.

5.1. The real lattice. In the next lemma we define a real lattice $\mathcal{L}(\widetilde{\mathbf{L}})$ and prove a lower bound on the length of its nonzero vectors. The definition of $\widetilde{\mathbf{L}}$ is parametric with respect to an ℓ_p norm ($p \geq 1$), a real number $\alpha > 0$, and a set of positive integers $A = \{a_1, \ldots, a_m\}$. The idea is to map the multiplicative structure of the integers $a_1, \ldots, a_m$ to the additive structure of the lattice $\mathcal{L}(\widetilde{\mathbf{L}})$, defining a basis vector for each a_i and expressing its entries in terms of the logarithm of a_i. This way the problem of finding a sphere containing many lattice points is reduced to the problem of finding a small interval containing many products of the a_i's. At the end we will set α to some large number (exponential in m), and A to a set of small primes. The existence of a sphere containing many lattice points will follow from the density of the primes and a simple averaging argument.

LEMMA 5.1. *Let $A = \{a_1, \ldots, a_m\}$ be a set of relatively prime odd positive integers. Then for any ℓ_p norm ($p \geq 1$), and any real $\alpha > 0$, all nonzero vectors in the lattice generated by the (columns of the) matrix*

$$\widetilde{\mathbf{L}} = \begin{bmatrix} \sqrt[p]{\ln a_1} & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sqrt[p]{\ln a_m} \\ \alpha \ln a_1 & \cdots & \alpha \ln a_m \end{bmatrix} \in \mathbb{R}^{(m+1)\times m} \tag{5.1}$$

have ℓ_p norm bigger than $\sqrt[p]{2 \ln \alpha}$.

Proof. We want to prove that for all nonzero integer vectors $\mathbf{z} \in \mathbb{Z}^m$,

$$\|\widetilde{\mathbf{L}}\mathbf{z}\|_p^p \geq 2 \ln \alpha.$$

We first introduce some notation. Let $\mathbf{R} \in \mathbb{R}^m$ be the row vector

$$\mathbf{R} = [\ln a_1, \ln a_2, \ldots, \ln a_m] \tag{5.2}$$

and $\mathbf{D} \in \mathbb{R}^{m\times m}$ be the diagonal matrix

$$\mathbf{D} = \begin{bmatrix} \sqrt[p]{\ln a_1} & 0 & \cdots & 0 \\ 0 & \sqrt[p]{\ln a_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \sqrt[p]{\ln a_m} \end{bmatrix}. \tag{5.3}$$

Notice that

$$\widetilde{\mathbf{L}} = \begin{bmatrix} \mathbf{D} \\ \alpha \mathbf{R} \end{bmatrix}$$

and $\|\widetilde{\mathbf{L}}\mathbf{z}\|_p^p = \|\mathbf{D}\mathbf{z}\|_p^p + \alpha^p |\mathbf{R}\mathbf{z}|^p$. We bound the two terms separately. Define the integers

$$\hat{g} = \prod\{a_i^{z_i} : z_i > 0\}, \qquad \check{g} = \prod\{a_i^{-z_i} : z_i < 0\}, \qquad g = \hat{g}\check{g} = \prod_{i=1}^{m} a_i^{|z_i|}.$$

Using this notation, the first term satisfies

$$\begin{aligned}\|\mathbf{Dz}\|_p^p &= \sum_i |z_i|^p \ln a_i \\ &\geq \sum_i |z_i| \ln a_i \\ &= \ln g\end{aligned}$$

because $p \geq 1$ and the z_i's are integers. Bounding the second term is slightly more complex:

$$\begin{aligned}|\mathbf{Rz}| &= \left| \sum_i z_i \ln a_i \right| \\ &= |\ln \hat{g} - \ln \check{g}| \\ &= \ln \left(1 + \frac{|\hat{g} - \check{g}|}{\min\{\hat{g}, \check{g}\}} \right).\end{aligned}$$

Now notice that since $\mathbf{z}$ is nonzero, $\hat{g}$ and $\check{g}$ are distinct odd integers and therefore $|\hat{g} - \check{g}| \geq 2$. Moreover, $\min\{\hat{g}, \check{g}\} < \sqrt{\hat{g}\check{g}} = \sqrt{g}$. By monotonicity and concavity of function $\ln(1 + x)$ over the interval $[0, 2]$, one gets

$$\ln \left(1 + \frac{|\hat{g} - \check{g}|}{\min\{\hat{g}, \check{g}\}} \right) > \ln \left(1 + \frac{2}{\sqrt{g}} \right) > \frac{2}{\sqrt{g}} \cdot \frac{\ln 3}{2} > \frac{1}{\sqrt{g}}.$$

Combining the two bounds one gets

$$\|\widetilde{\mathbf{L}}\mathbf{z}\|_p^p = \|\mathbf{Dz}\|_p^p + \alpha^p (\mathbf{Rz})^p > \ln g + \frac{\alpha^p}{g^{p/2}}$$

which is a continuous function of g with derivative

$$\frac{1}{g} \left(1 - \frac{p}{2} \cdot \frac{\alpha^p}{g^{p/2}} \right).$$

The function is minimized (over the reals) when $g = \alpha^2 \left(\frac{p}{2}\right)^{2/p}$ with minimum

$$2 \ln \alpha + \left(\frac{2}{p} \right) \ln \left(\frac{p}{2} \right) + \left(\frac{2}{p} \right) > 2 \ln \alpha + \left(\frac{2}{p} \right) \ln p > 2 \ln \alpha.$$

Therefore, for all nonzero integer vectors $\mathbf{z}$, $\|\widetilde{\mathbf{L}}\mathbf{z}\|_p^p > 2 \ln \alpha$. □

Consider now a sphere centered in

$$\widetilde{\mathbf{s}} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \alpha \ln \beta \end{bmatrix}, \tag{5.4}$$

where β is a positive real to be specified. We now show that there is a close relationship between finding lattice vectors close to $\widetilde{\mathbf{s}}$ and approximating β as a product of the a_i's.

LEMMA 5.2. *Let* $\widetilde{\mathbf{L}}$ *and* $\widetilde{\mathbf{s}}$ *be defined as in* (5.1) *and* (5.4). *For any* ℓ_p *norm* ($p \geq 1$), *reals* $\alpha, \beta \geq 1$, *positive integers* $a_1, \ldots, a_m$, *and boolean vector* $\mathbf{z} \in \{0,1\}^m$, *if the integer* $g = \prod_i a_i^{z_i}$ *belongs to the interval* $[\beta, \beta(1+1/\alpha)]$, *then*

$$\|\widetilde{\mathbf{L}}\mathbf{z} - \widetilde{\mathbf{s}}\|_p \leq \sqrt[p]{\ln \beta + 2}.$$

Proof. Let $\mathbf{D}$ and $\mathbf{R}$ be as defined in (5.3) and (5.2). Notice that since $\mathbf{z}$ is a 0-1 vector,

$$\|\mathbf{D}\mathbf{z}\|_p^p = \mathbf{R}\mathbf{z} = \ln g,$$

and therefore

$$\begin{aligned}\|\widetilde{\mathbf{L}}\mathbf{z} - \widetilde{\mathbf{s}}\|_p^p &= \|\mathbf{D}\mathbf{z}\|_p^p + \alpha^p |\mathbf{R}\mathbf{z} - \ln \beta|^p \\ &= \ln g + \alpha^p |\ln g - \ln \beta|^p \\ &= \ln \beta + \ln \frac{g}{\beta} + \left|\alpha \ln \frac{g}{\beta}\right|^p.\end{aligned}$$

From the assumption $g \in [\beta, \beta(1+1/\alpha)]$ and using the inequality $\ln(1+x) < x$ (true for all $x \neq 0$) one gets

$$0 \leq \ln \frac{g}{\beta} \leq \ln\left(1 + \frac{1}{\alpha}\right) < \frac{1}{\alpha}$$

which, substituted in the above expression, gives

$$\|\widetilde{\mathbf{L}}\mathbf{z} - \widetilde{\mathbf{s}}\|_p^p < \ln \beta + \frac{1}{\alpha} + 1 \leq \ln \beta + 2 \qquad \square$$

Now let ϵ be a small positive real constant and set $\alpha = \beta^{(1-\epsilon)}$. From Lemma 5.1, the minimum distance between lattice points is bigger than $\lambda = \sqrt[p]{2(1-\epsilon)\ln \beta}$, and there are many lattice points within distance $\sqrt[p]{\ln \beta + 2} \approx \lambda / \sqrt[p]{2}$ from $\widetilde{\mathbf{s}}$, provided that the interval $[\beta, \beta + \beta^\epsilon]$ contains many products of the form $\prod_{i \in S} a_i$ ($S \subseteq \{1, \ldots, m\}$). If $a_1, \ldots, a_m$ are the first m odd prime numbers, this is the same as saying that $[\beta, \beta + \beta^\epsilon]$ contains many square-free odd (a_m)-smooth numbers. We now informally estimate for which values of m and β one should expect $[\beta, \beta + \beta^\epsilon]$ to contain a large number of such products. A rigorous probabilistic analysis will follow right after.

Fix some integer $c > 1/\epsilon$, let h be a sufficiently large integer and set $m = h^c$. Let $a_1, \ldots, a_m$ be the first m odd primes, and consider the set of products

$$M = \left\{ \prod_{i \in S} a_i : S \subset \{1, \ldots, m\}, |S| = h \right\}.$$

Notice that

$$|M| = \binom{m}{h} = \prod_{i=0}^{h-1} \frac{m-i}{h-i} \geq \prod_{i=0}^{h-1} \frac{m}{h} = h^{(c-1)h} \tag{5.5}$$

and all elements of M belong to the interval $[1, (a_m)^h]$. If we choose β uniformly at random in this interval, the expected size of $[\beta, \beta + \beta^\epsilon]$ is $\Omega((a_m)^{\epsilon h})$ and we can estimate the number of elements of M contained in $[\beta, \beta + \beta^\epsilon]$ to be

$$\Omega((a_m)^{\epsilon h}) \cdot \frac{|M|}{(a_m)^h} \geq \Omega\left(\frac{h^{c-1}}{(a_m)^{1-\epsilon}}\right)^h.$$

By the prime number theorem, $a_m = O(m \ln m) = O(h^c \ln h)$ and therefore our estimate is $\Omega(h^{\epsilon c - 1}/\ln h)^h > 2^h$ for all sufficiently large h.

Making the above argument more formal, one can prove that there exists an interval $[\beta, \beta + \beta^\epsilon]$ containing exponentially (in h) many products from M. Still, it is not clear how to find the right β. If square-free smooth numbers are distributed uniformly enough, any choice of β is good. Unfortunately, we do not know enough about the distribution of smooth numbers to prove such a statement about small intervals $[\beta, \beta + \beta^\epsilon]$. (It can be proved that for all β the interval $[\beta, 2\beta]$ contains square-free smooth numbers, but not much is known about interval of sublinear size.)

So, we exploit the smooth number distribution (whatever it is) to bias the choice of the interval toward those containing many smooth numbers. The idea is to set β to the product of a random (size h) subset of the a_i's. This way, the interval $[\beta, \beta + \beta^\epsilon]$ is selected with a probability roughly proportional to the number of square-free (a_m)-smooth numbers contained in it. So, for example, intervals containing no smooth numbers are never selected, and intervals containing few smooth numbers are selected with very small probability. The probability of choosing an interval containing few products is bounded in the next lemma. In fact the lemma is quite general and applies to any set M of real numbers bigger than 1.

LEMMA 5.3. *For every positive real numbers $\epsilon \in [0, 1)$, $\mu > 1$, integer $H \geq 1$, and any finite subset $M \subset [1, \mu)$, if β is chosen uniformly at random from M, then the probability that $[\beta, \beta + \beta^\epsilon)$ contains less than H elements from M is at most*

$$\Pr_{\beta \in M}\{|[\beta, \beta + \beta^\epsilon) \cap M| < H\} \leq \frac{\mu^{1-\epsilon} \cdot H}{\kappa(\epsilon) \cdot |M|},$$

where $\kappa(\epsilon) = 1 - 2^{1-\epsilon}$.

Proof. Let B be the set of all $\beta \in M$ such that $|[\beta, \beta + \beta^\epsilon) \cap M| < H$. We show that $|B|$ can be partitioned into at most $K = \mu^{1-\epsilon}/\kappa(\epsilon)$ subsets, each containing less than H elements. It follows that

$$\Pr_{\beta \in M}\{\beta \in B\} = \frac{|B|}{|M|} \leq \frac{K(H-1)}{|M|} = \frac{\mu^{1-\epsilon} \cdot H}{\kappa(\epsilon) \cdot |M|}.$$

Divide $[1, \mu)$ into $\lceil \log_2 \mu \rceil$ intervals $[2^k, 2^{k+1})$ for $k = 0, \ldots, \lceil \log_2 \mu \rceil - 1$. Then divide each interval $[2^k, 2^{k+1})$ into $2^k/2^{\epsilon k} = 2^{(1-\epsilon)k}$ subintervals of size $2^{\epsilon k}$. Notice that each subinterval is of the form $[x, x + y)$ for some $y \leq x^\epsilon$, therefore it contains at most $H - 1$ points from B. It remains to count the total number of subintervals. Adding up the number of subintervals for each interval $[2^k, 2^{k+1})$ we get

$$\begin{aligned} K &= \sum_{k=0}^{\lceil \log_2 \mu \rceil - 1} 2^{(1-\epsilon)k} \\ &= \frac{2^{(1-\epsilon)\lceil \log_2 \mu \rceil} - 1}{2^{1-\epsilon} - 1} \\ &< \frac{(2\mu)^{1-\epsilon}}{2^{1-\epsilon} - 1} = \frac{\mu^{1-\epsilon}}{\kappa(\epsilon)}. \end{aligned}$$

□

Applying this lemma to the set of square free smooth numbers we get the following proposition.

PROPOSITION 5.4. *For all reals $\epsilon, \delta > 0$, there exists an integer c such that for all sufficiently large integer h, the following holds. Let $m = h^c$, $a_1, \ldots, a_m$ be the*

first m odd primes, and M the set of all products $\prod_{i\in S} a_i$, where S is a size h subset of $\{1,\ldots,m\}$. If β is chosen uniformly at random from M then the probability that $[\beta,\beta+\beta^\epsilon)$ contains less than $h^{\delta h}$ elements of M is at most 2^{-h}.

Proof. Fix some $\epsilon,\delta>0$ and let c be an integer bigger than $(1+\delta)/\epsilon$. Let $\mu=a_m^h$. Notice that M is contained in $[1,\mu)$ and $|M|\geq h^{(c-1)h}$ (see (5.5)). Applying Lemma 5.3 to set M with $H=h^{\delta h}$, we get

$$\begin{aligned}\Pr\{|[\beta,\beta+\beta^\epsilon)\cap M|<H\} &< \frac{h^{\delta h}\cdot\mu^{1-\epsilon}}{\kappa(\epsilon)|M|}\\ &< \frac{h^{\delta h}a_m^{(1-\epsilon)h}}{\kappa(\epsilon)h^{(c-1)h}}.\end{aligned}$$

By the prime number theorem, $a_m=O(m\ln m)=O(h^c\ln h)$, which substituted in the above expression gives

$$\begin{aligned}\Pr\{|[\beta,\beta+\beta^\epsilon]\cap M|\leq H\} &< \frac{h^{\delta h}\,O(h^c\ln h)^{(1-\epsilon)h}}{\kappa(\epsilon)h^{(c-1)h}}\\ &= \left(\frac{O(\ln h)^{(1-\epsilon)}}{h^{\epsilon c-(1+\delta)}}\right)^h\\ &< \left(\frac{O(\ln h)}{h^{\epsilon c-(1+\delta)}}\right)^h\\ &< 2^{-h}\end{aligned}$$

for all sufficiently large h because $\epsilon c-(1+\delta)>0$. □

Combining Lemma 5.1, Lemma 5.2, and Proposition 5.4, we immediately get the following theorem.

THEOREM 5.5. *For all reals $\epsilon,\delta>0$, there exists an integer c such that the following holds. Let h be a positive integer, $m=h^c$, and $a_1,\ldots,a_m$ be the first m odd primes. Let β be the product of a random subset of $\{a_1,\ldots,a_m\}$ of size h and set $\alpha=\beta^{1-\epsilon}$. Define $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{s}}$ as in (5.1) and (5.4), and let $\widetilde{r}=\sqrt[p]{(1+\epsilon)\ln\beta}>1$. Then*

(i) *all nonzero vectors in $\mathcal{L}(\widetilde{\mathbf{L}})$ have ℓ_p norm greater than $\sqrt[p]{2\left((1-\epsilon)/(1+\epsilon)\right)}r$.*

(ii) *For all sufficiently large h, with probability at least $1-2^{-h}$, the ball $\mathcal{B}(\widetilde{\mathbf{s}},r)$ contains more than $h^{\delta h}$ lattice points of the form $\mathbf{Lz}$ where $\mathbf{z}$ is a 0-1 vector with exactly h ones.*

5.2. Working over the integers. In the previous subsection we proved that as far as real entries are allowed one can easily define a basis $\widetilde{\mathbf{L}}$ and probabilistically find a vector $\widetilde{\mathbf{s}}$ with the property that a sphere of radius slightly more then $\lambda(\widetilde{\mathbf{L}})/\sqrt[p]{2}$ contains many lattice points. We now prove that the same result can be achieved using a suitable integer approximation of $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{s}}$. The error incurred by approximating a multiple of $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{s}}$ with integers is bounded in the following two lemmas.

LEMMA 5.6. *For all $\eta\geq 1$ and all integer vectors $\mathbf{z}\in\mathbb{Z}^m$,*

$$\|\mathbf{Lz}\|_p\geq(\eta-1)m\|\widetilde{\mathbf{L}}\mathbf{z}\|_p,$$

where $\mathbf{L}=\lfloor(m\eta)\widetilde{\mathbf{L}}\rceil$ is the matrix obtained multiplying $\widetilde{\mathbf{L}}$ by $m\eta$ and rounding each entry to the closest integer.

Proof. By triangular inequality

$$\|\mathbf{Lz}\|_p=\|(m\eta)\widetilde{\mathbf{L}}\mathbf{z}+(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z}\|_p$$

$$\begin{aligned}&\geq \|(m\eta)\widetilde{\mathbf{L}}\mathbf{z}\|_p - \|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z}\|_p\\&= \eta m\|\widetilde{\mathbf{L}}\mathbf{z}\|_p - \|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z}\|_p.\end{aligned}$$

It remains to prove that $\|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z}\|_p \leq m\|\widetilde{\mathbf{L}}\mathbf{z}\|_p$. Notice that all entries in $(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})$ are at most $1/2$ in absolute value. Therefore

$$\begin{aligned}\|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z}\|_p &\leq \frac{1}{2}\sqrt[p]{\|\mathbf{z}\|_p^p + \left(\sum|z_i|\right)^p}\\&\leq \frac{1}{2}\sqrt[p]{\|\mathbf{z}\|_p^p + m^p\|\mathbf{z}\|_p^p}\\&\leq m\|\mathbf{z}\|_p.\end{aligned}$$

Furthermore,

$$\begin{aligned}\|\widetilde{\mathbf{L}}\mathbf{z}\|_p^p &= \|\mathbf{D}\mathbf{z}\|_p^p + \alpha^p|\mathbf{R}\mathbf{z}|^p\\&\geq \|\mathbf{D}\mathbf{z}\|_p^p\\&\geq \|\mathbf{z}\|_p^p\end{aligned}$$

because $\mathbf{D}$ is diagonal with all entries greater than 1. This proves that $\|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z}\|_p \leq m\|\widetilde{\mathbf{L}}\mathbf{z}\|_p$ and therefore $\|\mathbf{L}\mathbf{z}\|_p \geq (\eta-1)m\|\widetilde{\mathbf{L}}\mathbf{z}\|_p$. □

LEMMA 5.7. *For all $\eta > 0$ and all integer vectors $\mathbf{z} \in \mathbb{Z}^m$*

$$\|\mathbf{L}\mathbf{z}-\mathbf{s}\|_p \leq (\eta+1)m\|\widetilde{\mathbf{L}}\mathbf{z}-\widetilde{\mathbf{s}}\|_p,$$

where $\mathbf{L} = \lfloor(m\eta)L\rceil$ and $\mathbf{s} = \lfloor(m\eta)\mathbf{s}\rceil$ are the matrices obtained multiplying $\widetilde{\mathbf{L}}$ and $\widetilde{\mathbf{s}}$ by $m\eta$ and rounding each entry to the closest integer.

Proof. By triangular inequality

$$\begin{aligned}\|\mathbf{L}\mathbf{z}-\mathbf{s}\|_p &= \|((m\eta)\widetilde{\mathbf{L}}\mathbf{z}-(m\eta)\widetilde{\mathbf{s}}) + (\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z} - (\mathbf{s}-(m\eta)\widetilde{\mathbf{s}})\|_p\\&\leq \|((m\eta)\widetilde{\mathbf{L}}\mathbf{z}-(m\eta)\widetilde{\mathbf{s}})\|_p + \|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z} - (\mathbf{s}-(m\eta)\widetilde{\mathbf{s}})\|_p\\&= \eta m\|\widetilde{\mathbf{L}}\mathbf{z}-(m\eta)\widetilde{\mathbf{s}}\|_p + \|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z} - (\mathbf{s}-(m\eta)\widetilde{\mathbf{s}})\|_p.\end{aligned}$$

Notice that all entries in $(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})$ and $(\mathbf{s}-(m\eta)\widetilde{\mathbf{s}})$ are at most $1/2$ in absolute value. Therefore

$$\|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z} - (\mathbf{s}-(m\eta)\widetilde{\mathbf{s}})\|_p^p \leq \left(\frac{1}{2}\right)^p\left(\|\mathbf{z}\|_p^p + \left(\sum|z_i|+1\right)^p\right) < m^p\|\mathbf{z}\|_p^p.$$

Furthermore,

$$\|\widetilde{\mathbf{L}}\mathbf{z}-\widetilde{\mathbf{s}}\|_p \geq \|\mathbf{D}\mathbf{z}\|_p \geq \|\mathbf{z}\|_p$$

because $\mathbf{D}$ is diagonal with all entries greater than 1. This proves that

$$\|(\mathbf{L}-(m\eta)\widetilde{\mathbf{L}})\mathbf{z} - (\mathbf{s}-(m\eta)\widetilde{\mathbf{s}})\|_p \leq m\|\widetilde{\mathbf{L}}\mathbf{z}-\widetilde{\mathbf{s}}\|_p,$$

and therefore

$$\|\mathbf{L}\mathbf{z}-\mathbf{s}\|_p \leq (\eta+1)\|\mathbf{L}\mathbf{z}-\mathbf{s}\|_p \qquad \square$$

We can now prove a variant of Theorem 5.5 where all the numbers are integers.

THEOREM 5.8. *For every $p \geq 1$, $\gamma \in [1, \sqrt[p]{2})$ and $\delta > 0$ there exists a probabilistic algorithm that on input an integer h outputs (in $\mathrm{poly}(h)$ time) integers m, r, a matrix $\mathbf{L} \in \mathbb{Z}^{(m+1)\times m}$, and an integer vector $\mathbf{s} \in \mathbb{Z}^{m+1}$ such that*

(i) *all vectors in* $\mathcal{L}(\mathbf{L})$ *have* ℓ_p *norm bigger than* γr*;*
(ii) *for all sufficiently large* h*, with probability at least* $1-2^{-h}$ *the sphere* $\mathcal{B}(\mathbf{s},r)$ *contains at least* $h^{\delta h}$ *lattice points of the form* $\mathbf{Lz}$ *where* $\mathbf{z}$ *is a* 0-1 *vector with exactly* h *ones.*

Proof. We show that for all $p \geq 1, \delta > 0$ and $\epsilon > 0$ the theorem is satisfied with

$$\gamma = \left(\frac{(1-\epsilon)^{1+1/p}}{(1+\epsilon)^{2+1/p}}\right) \cdot \sqrt[p]{2}.$$

Let c be as in Theorem 5.5. On input h, algorithm A computes $m = h^c$, and the first m odd primes $a_1, a_2, \ldots, a_m$. Let $\widetilde{\mathbf{L}}$, $\widetilde{\mathbf{s}}$, and $\widetilde{r}$ be as defined in Theorem 5.5, and compute the approximations

$$\mathbf{L} = \lfloor (m/\epsilon)\widetilde{\mathbf{L}} \rceil, \qquad \mathbf{s} = \lfloor (m/\epsilon)\widetilde{\mathbf{s}} \rceil, \qquad r = \lceil (1+1/\epsilon) m\widetilde{r} \rceil.$$

Let $\mathbf{z} \in \mathbb{Z}^m$ be a nonzero integer vector. We want to bound $\|\mathbf{Lz}\|_p$. We know from Theorem 5.5 that

$$\|\widetilde{\mathbf{L}}\mathbf{z}\|_p > \sqrt[p]{2\frac{1-\epsilon}{1+\epsilon}}\widetilde{r}. \tag{5.6}$$

Using Lemma 5.6 (with $\eta = 1/\epsilon$) and (5.6) we get

$$\begin{aligned} \|\mathbf{Lz}\|_p &\geq \left(\frac{1}{\epsilon} - 1\right) m \|\widetilde{\mathbf{L}}\mathbf{z}\|_p \\ &> \left(\frac{(1-\epsilon)^{1+1/p}}{\epsilon(1+\epsilon)^{1/p}}\right) m \sqrt[p]{2} \cdot \widetilde{r}. \end{aligned} \tag{5.7}$$

Notice that r satisfies the bounds $r < (1+1/\epsilon)m\widetilde{r} + 1$ and $r > (1+1/\epsilon)$ because $\widetilde{r} > 1$. Thus, we can bound $\widetilde{r}$ as follows:

$$\begin{aligned} \widetilde{r} &> \frac{r-1}{(1+1/\epsilon)m} \\ &= \frac{1-1/r}{(1+1/\epsilon)m} \cdot r \\ &> \frac{1-1/(1+1/\epsilon)}{(1+1/\epsilon)m} \cdot r \\ &= \frac{\epsilon}{(\epsilon+1)^2 m} \cdot r. \end{aligned} \tag{5.8}$$

Combining (5.7) and (5.8) we get

$$\|\mathbf{Lz}\|_p > \left(\frac{(1-\epsilon)^{1+1/p}}{\epsilon(1+\epsilon)^{1/p}}\right) \sqrt[p]{2} \frac{\epsilon}{(\epsilon+1)^2} r = \gamma r.$$

Now consider the sphere $\mathcal{B}(\mathbf{s}, r)$. By Theorem 5.5, for all sufficiently large h, with probability at least $1-2^{-h}$, the ball $\mathcal{B}(\widetilde{\mathbf{s}}, \widetilde{r})$ contains at least $h^{\delta h}$ lattice points of the form $\widetilde{\mathbf{L}}\mathbf{z}$ where $\mathbf{z}$ is a 0-1 vector with exactly h ones. For each such point $\widetilde{\mathbf{L}}\mathbf{z}$, we can use Lemma 5.7 (with $\eta = 1/\epsilon$) to bound the distance of $\mathbf{Lz}$ from $\mathbf{s}$ as follows:

$$\begin{aligned} \|\mathbf{Lz} - \mathbf{s}\|_p &\leq (1+1/\epsilon) m \|\widetilde{\mathbf{L}}\mathbf{z} - \widetilde{\mathbf{s}}\|_p \\ &\leq (1+1/\epsilon) m\widetilde{r} \leq r. \end{aligned}$$

Therefore $\mathbf{Lz}$ belongs to the sphere $\mathcal{B}(\mathbf{s}, r)$. This proves that $\mathcal{B}(\mathbf{s}, r)$ also contains at least $h^{\delta h}$ lattice points of the desired form. ∎

5.3. Projecting lattice points to binary strings. In order to complete the proof of Lemma 4.1 we need the following combinatorial theorem from [27]. (See the proof in the appendix.)

THEOREM 5.9. *Let $\mathcal{Z} \subseteq \{0,1\}^m$ be a set of vectors containing exactly h ones. If $|\mathcal{Z}| \geq h! m^{\frac{4\sqrt{h}k}{\epsilon}}$, and $\mathbf{T} \in \{0,1\}^{k \times m}$ is chosen setting each entry to 1 independently at random with probability $p = \frac{1}{4hk}$, then the probability that all binary vectors $\{0,1\}^k$ are contained in $\mathbf{T}(\mathcal{Z}) = \{\mathbf{Tz} : \mathbf{z} \in \mathcal{Z}\}$ is at least $1 - 6\epsilon$.*

We remark that a similar theorem was already proved in [2], and we could have used that result instead of Theorem 5.9. However, our construction and analysis are much simpler than those in [2] and are probably more efficient.

We can now prove Lemma 4.1. Fix an ℓ_p norm ($p \geq 1$) and a constant $\gamma \in [1, \sqrt[p]{2})$. Let k be a sufficiently large integer. We want to build in $\text{poly}(k)$ time an integer lattice $\mathbf{L}$, an integer vector $\mathbf{s}$, an integer transformation matrix $\mathbf{T}$, and an integer radius r such that

(i) all nonzero vectors in $\mathcal{L}(\mathbf{L})$ have ℓ_p norm greater than γr;

(ii) with probability at least $1 - 1/\text{poly}(k)$, for all $\mathbf{x} \in \{0,1\}^k$ there exists a $\mathbf{z} \in \mathbb{Z}^m$ such that $\mathbf{Tz} = \mathbf{x}$ and $\|\mathbf{Lz} - \mathbf{s}\|_p \leq r$.

Let $\delta = 2$ and run the algorithm from Theorem 5.8 on input $h = k^4$. This algorithm outputs an integer matrix $\mathbf{L} \in \mathbb{Z}^{(m+1) \times m}$ and a vector $\mathbf{s} \in \mathbb{Z}^m$ and $r \in \mathbb{Z}$. Notice that since $\mathbf{s}$ is computed in polynomial time, m must be polynomial in h, i.e., $m < h^c$ for some constant c independent of h. Let $\mathcal{Z}$ be the set of all vectors $\mathbf{z} \in \{0,1\}^m$ with exactly h ones, such that $\mathbf{Lz} \in \mathcal{B}(\mathbf{s}, r)$. We know from Theorem 5.8 that all nonzero vectors in $\mathcal{L}(\mathbf{L})$ have ℓ_p norm greater than γr, and with probability at least $1 - 2^{-h}$ the set $\mathcal{Z}$ contains at least h^{2h} elements.

Now, choose matrix $\mathbf{T} \in \{0,1\}^{k \times m}$ by setting each entry to one independently with probability $1/(4hk)$. Notice that

$$|\mathcal{Z}| \geq h^{2h} > h! m^{h/c} = h! m^{\frac{4\sqrt{h}k}{\epsilon}},$$

where $\epsilon = 4c/k$. So, by Theorem 5.9, the probability that for each $\mathbf{x}$ there exists a vector $\mathbf{z}$ such that $\mathbf{x} = \mathbf{Tz}$ and $\mathbf{Lz} \in \mathcal{B}(\mathbf{s}, r)$ is at least $1 - 1/O(k)$. This concludes the proof of Lemma 4.1.

6. Deterministic reductions. In section 4 we proved that approximating SVP is hard for NP under RUR-reductions. In particular, this proves that approximating SVP is not in RP unless NP = RP. In this section we address the question whether SVP is hard under deterministic reductions.

A quick inspection of the proof of Theorem 4.2 immediately shows that the only place in the reduction where randomness is used is Lemma 4.1. A deterministic polynomial time algorithm satisfying the conditions in Lemma 4.1 would immediately give a proper NP-hardness result (under deterministic many-one reductions) for GAPSVP.

To date, we do not know if such a deterministic polynomial time algorithm exists. However, one can show that such an algorithm exists if one assumes a reasonable number theoretic conjecture, or allows for nonuniform reductions.

6.1. Nonuniform reductions. The reduction presented in the proof of Theorem 4.2 always maps NO instances to NO instances and YES instances to YES instances provided that the probabilistic algorithm in the lemma succeeds. Notice that the construction in the lemma depends only on the dimension k of the GAPCVP$'$ instance we are reducing. Moreover, the success of the algorithm does not depend on the particular instance we are reducing.

Therefore, if we allow for nonuniform reductions in the proof of Theorem 4.2, we can encode the objects $\mathbf{L}, \mathbf{T}, \mathbf{s}, r$ satisfying Lemma 4.1 directly in the reduction as polynomial size nonuniform hints. Notice that the existence of $\mathbf{L}, \mathbf{T}, \mathbf{s}, r$ is guaranteed by the (probabilistic) proof of Lemma 4.1.

This gives the following variant of Theorem 4.2.

THEOREM 6.1. *For any ℓ_p norm ($p \geq 1$) and for any constant $\gamma \in [1, \sqrt[p]{2})$, the promise problem* GAPSVP$_\gamma$ *is hard for* NP *under deterministic nonuniform polynomial reductions. In particular,* GAPSVP$_\gamma$ *is not in* P/poly *unless* NP $\subseteq$ P/poly.

Using standard results on nonuniform complexity [21], this also implies the following corollary.

COROLLARY 6.2. *For any ℓ_p norm ($p \geq 1$) and for any constant $\gamma \in [1, \sqrt[p]{2})$, the promise problem* GAPSVP$_\gamma$ *is not in* P *unless the polynomial hierarchy* [26, 33] *collapses to the second level.*

6.2. NP-hardness under a number theoretic conjecture. In this section we show how the proof of the geometric lemma can be made deterministic using a number theoretic conjecture. This results in a proper NP-hardness result for GAPSVP (i.e. NP-hardness under deterministic many-one reductions) but relies on an unproven assumption on the distribution of square-free smooth numbers. The conjecture is the following.

CONJECTURE 1. *For any $\epsilon > 0$ there exists a d such that for all large enough n, there exists an (odd) integer in $[n, n + n^\epsilon]$ which is square-free and $(\log^d n)$-smooth; i.e., all of its prime factors have exponent 1 and are less than $\log^d n$.*

We remark that although the above conjecture is very plausible, proving it seems to be beyond current mathematical techniques. We now show that if the above conjecture is true, then there exists a deterministic (uniform) polynomial time algorithm satisfying the requirements of Lemma 4.1. For simplicity, we show how to build real matrices $\mathbf{L}, \mathbf{s}, r$ satisfying the condition in the lemma. $\mathbf{L}, \mathbf{s}, r$ can be easily transformed into integer matrices as explained in subsection 5.2 using Lemma 5.6 and Lemma 5.7 to bound the errors incurred in the approximation process.

Let ϵ be a positive real between 0 and 1. Let d be an integer (whose existence is guaranteed by the conjecture) such that for all large enough n there exists a $(\log^d n)$-smooth square-free (odd) integer in the interval $[n, n+n^{\epsilon/2}]$. Let $\mathbf{L}$ and $\mathbf{s}$ be as defined in (5.1) and (5.4) with $m = k^{d+1} + k$, $a_1, \ldots, a_m$ the first m (odd) prime numbers, $\beta = a_m^{\frac{2k}{\epsilon}}$ and $\alpha = \beta^{1-\epsilon}$. Finally, let $\mathbf{T} \in \{0,1\}^{k \times m}$ be the matrix $\mathbf{T} = [\mathbf{0}_{k \times k^{d+1}} | \mathbf{I_k}]$.

From Lemma 5.1 we know that for all nonzero vectors $\mathbf{z} \in \mathbb{Z}^m$,

$$\|L\mathbf{z}\|_p \geq \sqrt[p]{2(1-\epsilon)\ln\beta}.$$

We now show that for all $\mathbf{x} \in \{0,1\}^k$ there exists a $\mathbf{y} \in \mathbb{Z}^{k^{d+1}}$ such that

$$\left\| \mathbf{L} \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} - \mathbf{s} \right\|_p < \sqrt[p]{\ln\beta + 2} = r. \tag{6.1}$$

Since the equality

$$\mathbf{T} \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix} = \mathbf{x}$$

follows directly from the definition to $\mathbf{T}$, (6.1) proves the second condition in Lemma 4.1. By Lemma 5.2 it is sufficient to show that for every integer $g_{\mathbf{x}} = \prod_{i=1}^k a_{k^{d+1}+i}^{x_i}$

(with $x_i \in \{0,1\}$) there exists an integer $g_{\mathbf{y}} = \prod_{i=1}^{k^{d+1}} a_i^{y_i}$ (with $y_i \in \{0,1\}$) such that $g = g_{\mathbf{x}} g_{\mathbf{y}} \in [\beta, \beta + \beta^{\epsilon}]$. Fix some $g_{\mathbf{x}} = \prod_{i=1}^{k} a_{k^{d+1}+i}^{x_i}$. Notice that

$$\frac{\beta}{g_{\mathbf{x}}} > \frac{\beta}{a_m^k} = a_m^{(\frac{2}{\epsilon}-1)k} > 2^k$$

so for all sufficiently large k, there exists a $\log^d(\beta/g_{\mathbf{x}})$-smooth square-free (odd) integer in the interval

$$[\beta/g_{\mathbf{x}}, (\beta/g_{\mathbf{x}}) + (\beta/g_{\mathbf{x}})^{\epsilon/2}].$$

But

$$\log^d(\beta/g_{\mathbf{x}}) \leq \log^d(\beta) = O(k \log k)^d < k^{d+1}.$$

So, this smooth number can be expressed as $g_{\mathbf{y}} = \prod_{i=1}^{k^{d+1}} a_i^{y_i}$ with $y_i \in \{0,1\}$. Therefore,

$$g_{\mathbf{x}} g_{\mathbf{y}} \in [\beta, \beta + g_{\mathbf{x}}(\beta/g_{\mathbf{x}})^{\epsilon/2}].$$

Finally, notice that $g_{\mathbf{x}} \leq a_m^k = \beta^{\epsilon/2}$. So, if we define

$$\mathbf{z} = \begin{bmatrix} \mathbf{y} \\ \mathbf{x} \end{bmatrix}$$

then

$$\prod a_i^{z_i} = g_{\mathbf{x}} g_{\mathbf{y}} \in [\beta, \beta + \beta^{\epsilon}]$$

and by Lemma 5.2 the lattice vector $\mathbf{Lz}$ belongs to the sphere $\mathcal{B}(\mathbf{s}, r)$.

This completes the proof that if Conjecture 1 is true, then $\mathbf{L}, \mathbf{T}, \mathbf{s}, r$ satisfy the conditions of Lemma 4.1. Then, the reduction in the proof of Theorem 4.2 gives the following corollary.

COROLLARY 6.3. *If Conjecture* 1 *holds true, then for any ℓ_p norm and any constant $\gamma < \sqrt[p]{2}$,* GAPSVP$_\gamma$ *is* NP*-hard (under deterministic many-one reductions).*

7. Discussion. We proved that approximating the shortest vector problem in any ℓ_p within factors less than $\sqrt[p]{2}$ is not in polynomial time under any of the following assumptions:

1. NP $\neq$ RP,
2. NP $\not\subseteq$ P/poly,
3. Conjecture 1 is true and NP $\neq$ P.

Although all of these results give theoretical evidence that SVP cannot be approximated in polynomial time, the problem whether solving SVP (even exactly) is NP-hard under deterministic many-one reductions remains open. We notice that the only place where randomness (or nonuniform hints, or the number theoretic conjecture) is used in our reduction is the proof of Lemma 4.1. A deterministic polynomial time solution to Lemma 4.1 would immediately give an NP-hardness result for SVP under deterministic many-one reductions. We leave finding a deterministic algorithm satisfying Lemma 4.1 as an open problem.

Our NP-hardness proof is by reduction from approximate CVP. In particular we reduced instances of CVP of size n to instances of SVP of size $m = n^c$, where $c > 2$

is a constant independent of n. Although this gives a polynomial relation between n and m it should be noted that m can be much bigger than n. Therefore, in order to assert that an instance of SVP is hard to solve in practice, the dimension m must be rather large. Finding a more efficient reduction, where, for example, $m = O(n)$, is left as an open problem. Interestingly, a dimension and approximation preserving reduction is possible in the other direction from SVP to CVP [16].

The geometric lemma used in our reduction is in a certain sense optimal (in the ℓ_2 norm): it can be formally proved that any lattice $\mathbf{L}$ satisfying the lemma must have vectors of length less than $r/\sqrt{2}$ (see [27]). Proving that SVP is NP-hard to approximate within factors larger than $\sqrt{2}$ cannot be done by simply improving the geometric lemma. We leave as an open problem to prove that SVP is NP-hard to approximate within any constant factor.

Appendix. A combinatorial theorem on low-degree hyper-graphs.

In this appendix we prove Theorem 5.9. We want to prove that if $\mathcal{Z} \subset \{0,1\}^m$ is a set of vectors of weight h, and $|\mathcal{Z}| \geq h! m^{\frac{4\sqrt{h}k}{\epsilon}}$, then the probability that $\{0,1\}^k \subseteq \mathbf{T}(\mathcal{Z})$ (where $\mathbf{T} \in \{0,1\}^{k\times m}$ is a linear transformation chosen at random setting each entry to 1 independently with probability $p = \epsilon/(4hk)$) is at least $1 - 6\epsilon$.

The theorem can be reformulated in terms of hyper-graphs as follows. Let $(N, \mathcal{Z})$ be an h-regular hyper-graph, i.e., a hyper-graph all of whose hyper-edges have size h. Let $\mathbf{T} = (T_1, \ldots, T_k)$ be a collection of subsets of N chosen at random including each element of N in T_i independently with probability $p = \epsilon/(4hk)$. For any subset $U \subseteq N$, let

$$\mathbf{T}(U) = (|T_1 \cap U|, |T_2 \cap U|, \ldots, |T_k \cap U|)$$

and define $\mathbf{T}(\mathcal{Z}) = \{\mathbf{T}(U) : U \in \mathcal{Z}\}$. We want to prove that if $|\mathcal{Z}| > h!|N|^{4\sqrt{h}k/\epsilon}$, then $\{0,1\}^k \subseteq \mathbf{T}(\mathcal{Z})$ with probability at least $1 - 6\epsilon$.

The correspondence between the matrix and hyper-graph formulation is immediate: identify the hyper-edges with the corresponding characteristic vectors in $\{0,1\}^{|N|}$ and the collection $\mathbf{T}$ with a matrix whose rows are the characteristic vectors of the sets T_i. Then $\mathbf{T}(U) = \mathbf{Tu}$ where $\mathbf{u}$ is the characteristic vector of set U.

We first prove a weaker result: we show for every vector $\mathbf{x} \in \{0,1\}^k$, $\mathbf{x} \in \mathbf{T}(\mathcal{Z})$ with high probability. Consider the target vector $\mathbf{x}$ as fixed. We want to bound the probability that $\mathbf{T}(U) \neq \mathbf{x}$ for all $U \in \mathcal{Z}$. Since the set $\mathcal{Z}$ is very big, the expected number of $U \in \mathcal{Z}$ such that $\mathbf{T}(U) = \mathbf{x}$ is also very high. Unfortunately, this is not sufficient to conclude that with high probability there exists a $U \in \mathcal{Z}$ such that $\mathbf{T}(U) = \mathbf{x}$, because the events $\mathbf{T}(U) = \mathbf{x}$ (indexed by the hyper-edges $U \in \mathcal{Z}$) might be strongly correlated. Notice that if U and V are disjoint (i.e., $U \cap V = \emptyset$), then the corresponding events are independent. In fact the size of the intersection $|U \cap V|$ is a good measure of the correlation between the events $\mathbf{T}(U) = \mathbf{x}$ and $\mathbf{T}(V) = \mathbf{x}$. Notice that if $|\mathcal{Z}|$ is big, then many hyper-edges in $\mathcal{Z}$ will intersect because there cannot be more than m/h mutually disjoint hyper-edges. However, one can still hope that for most of the pairs $U, V \in \mathcal{Z}$, the intersection $U \cap V$ is very small. This is not necessarily true for any hyper-graph $\mathcal{Z}$, but one can show that if $\mathcal{Z}$ is sufficiently large, then it must contain a large hyper-graph with this small intersection property.

The proof of the theorem is divided in four major steps:

1. We first show that the probability that $\mathbf{x} \notin \mathbf{T}(\mathcal{Z})$ can be bounded by the expectation

$$\operatorname*{Exp}_R[e^{\gamma R} - 1], \tag{A.1}$$

where γ is a small positive real, and $R = |U \cap V|$ is the random variable defined as the size of the intersection of two randomly chosen hyper-edges $U, V \in \mathcal{Z}$.

2. We show that $\mathcal{Z}$ "contains" a hyper-graph such that the intersection of two randomly selected hyper-edges is very small with high probability.

3. Then, we prove the weak version of the theorem applying the bound (A.1) to this hyper-graph contained in $\mathcal{Z}$.

4. Finally, we derive the strong version of our theorem from the weak one.

Each of the above steps is described in the following subsections.

A.1. The exponential bound. We start by computing the probability that $\mathbf{T}(U) = \mathbf{x}$ for some fixed set U. In the next lemma we prove a more general statement concerning the probability that two events $\mathbf{T}(U) = \mathbf{x}$ and $\mathbf{T}(V) = \mathbf{x}$ are simultaneously satisfied and relate it to the size of the intersection $r = |U \cap V|$ of the two sets U, V.

LEMMA A.1. *Let $\mathbf{x} \in \{0,1\}^k$ be any boolean vector, $U, V \subset N$ be two sets of size d and let $\mathbf{T} \in \{0,1\}^{k \times |N|}$ be chosen at random by setting each entry to 1 independently with probability p. Then, the probability (over the choice of $\mathbf{T}$) that both $\mathbf{T}(U)$ and $\mathbf{T}(V)$ equal $\mathbf{x}$ is*

$$\Phi(r) = (1-p)^{(2d-r)k} \left[\frac{pr}{1-p} + \left(\frac{p(d-r)}{1-p} \right)^2 \right]^{\|\mathbf{x}\|_1},$$

where $r = |U \cap V|$.

Proof. Since the rows of matrix $\mathbf{T}$ are chosen independently,

$$\Pr_{\mathbf{T}}\{\mathbf{T}(U) = \mathbf{T}(V) = \mathbf{x}\} = \prod_{i=1}^{k} \Pr_{T_i}\{|T_i \cap U| = |T_i \cap V| = x_i\}.$$

We prove that for all $i = 1, \ldots, k$,

$$\Pr_{T_i}\{|T_i \cap U| = |T_i \cap V| = x_i\} = (1-p)^{(2d-r)} \left[\frac{pr}{1-p} + \left(\frac{p(d-r)}{1-p} \right)^2 \right]^{x_i}.$$

First consider the case $x_i = 0$ and compute the probability (over the choice of T_i) that $|T_i \cap U| = |T_i \cap V| = 0$. This is true iff none of the elements of $U \cup V$ belongs to T_i, so the probability is

$$\Pr_{T_i}\{|T_i \cap U| = |T_i \cap V| = 0\} = (1-p)^{|U \cup V|} = (1-p)^{2d-r}.$$

Now consider the case $x_i = 1$ and compute the probability (over the choice of T_i) that $|T_i \cap U| = |T_i \cap V| = 1$. This is true iff either (1) T_i contains one element of $U \cap V$ and no other element of $U \cup V$, or (2) T_i contains one element of $U \setminus V$, one element of $V \setminus U$, and no other element of $U \cup V$. Event (1) has probability

$$|U \cap V| \cdot p(1-p)^{|U \cup V|-1} = (1-p)^{2d-r} \left(\frac{pr}{1-p} \right)$$

while event (2) has probability

$$|U \setminus V| \cdot |V \setminus U| \cdot p^2 (1-p)^{|U \cup V|-2} = (1-p)^{2d-r} \left(\frac{p(d-r)}{1-p} \right)^2.$$

Adding up the two probabilities, we get

$$\Pr_{T_i}\{|T_i \cap U| = |T_i \cap V| = 1\} = (1-p)^{(2d-r)}\left(\frac{pr}{1-p} + \left(\frac{p(d-r)}{1-p}\right)^2\right). \qquad \square$$

By choosing $U = V$ in the previous lemma one gets the following corollary.

COROLLARY A.2. *Let $\mathbf{x} \in \{0,1\}^k$ be a boolean vector, $U \subseteq N$ a subset of size d, and $\mathbf{T} \in \{0,1\}^{k \times |N|}$ a random matrix chosen by setting each entry to 1 independently with probability p. Then,*

$$\Pr_{\mathbf{T}}\{\mathbf{T}(U) = \mathbf{x}\} = \Phi(d) = (1-p)^{dk}\left(\frac{pd}{1-p}\right)^{\|\mathbf{x}\|_1}.$$

Notice that when $U \cap V = \emptyset$,

$$\Pr\{\mathbf{T}(U) = \mathbf{T}(V) = \mathbf{x}\} = \Phi(0) = \Phi(d)^2 = \Pr\{\mathbf{T}(U) = \mathbf{x}\}\Pr\{\mathbf{T}(V) = \mathbf{x}\},$$

i.e., the events $\mathbf{T}(U) = \mathbf{x}$ and $\mathbf{T}(V) = \mathbf{x}$ are independent. We can now prove the following proposition.

PROPOSITION A.3. *Let $(N, \mathcal{Z})$ be a d-regular hyper-graph and let $\mathbf{T} \in \{0,1\}^{k \times |N|}$ be chosen at random by setting each entry to 1 independently with probability p. Then, for each $\mathbf{x} \in \{0,1\}^k$ the probability (over the choice of $\mathbf{T}$) that $\mathbf{x} \notin \mathbf{T}(\mathcal{Z})$ is at most $\mathrm{Exp}_R[e^{\gamma R}] - 1$, where $\gamma = \frac{kp}{1-p} + \frac{k}{pd^2}$ and $R = |U \cap V|$ is the random variable defined as the size of the intersection of two randomly chosen elements of $\mathcal{Z}$.*

Proof. Fix some vector $\mathbf{x} \in \{0,1\}^k$ and choose $\mathbf{T}$ at random as specified in the proposition. For all $U \in \mathcal{Z}$, let X_U be the indicator random variable

$$X_U = \begin{cases} 1 & \text{if } \mathbf{T}(U) = \mathbf{x}, \\ 0 & \text{otherwise.} \end{cases}$$

Define the random variable $X = \sum_{U \in \mathcal{Z}} X_U$. Notice that $X = 0$ iff $\mathbf{x} \notin \mathbf{T}(\mathcal{Z})$. Moreover, if $X = 0$ then $|X - \mathrm{Exp}[X]| \geq \mathrm{Exp}[X]$. Using Chebyshev's inequality we get the following bound:

$$\begin{aligned} \Pr\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z})\} &= \Pr\{X = 0\} \\ &\leq \Pr\{|X - \mathrm{Exp}[X]| \geq \mathrm{Exp}[X]\} \\ &\leq \frac{\mathrm{Var}[X]}{\mathrm{Exp}[X]^2} = \frac{\mathrm{Exp}[X^2]}{\mathrm{Exp}[X]^2} - 1. \end{aligned}$$

So, let us compute the moments $\mathrm{Exp}[X]$ and $\mathrm{Exp}[X^2]$. For the first moment we have

$$\mathop{\mathrm{Exp}}_{\mathbf{T}}[X] = \sum_{U \in \mathcal{Z}} \Pr_{\mathbf{T}}\{\mathbf{T}(U) = \mathbf{x}\} = |\mathcal{Z}| \cdot \Phi(d),$$

and for the second one

$$\begin{aligned} \mathop{\mathrm{Exp}}_{\mathbf{T}}[X^2] &= \mathop{\mathrm{Exp}}_{\mathbf{T}}\left[\left(\sum_{U \in \mathcal{Z}} X_U\right)^2\right] \\ &= \mathop{\mathrm{Exp}}_{\mathbf{T}}\left[\sum_{U,V \in \mathcal{Z}} X_U \cdot X_V\right] \\ &= \sum_{U,V \in \mathcal{Z}} \Pr_{\mathbf{T}}\{\mathbf{T}(U) = \mathbf{T}(V) = \mathbf{x}\} \\ &= |\mathcal{Z}|^2 \cdot \mathop{\mathrm{Exp}}_{R}[\Phi(R)], \end{aligned}$$

where $R = |U \cap V|$ is the size of two randomly chosen $U, V \in \mathcal{Z}$. Therefore,

$$\begin{aligned}
\Pr_{\mathbf{T}}\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z})\} &= \frac{\operatorname{Exp}_R[\Phi(R)]}{\Phi(d)^2} - 1 \\
&= \operatorname*{Exp}_R\left[(1-p)^{-kR}\left(\frac{(1-p)R}{pd^2} + \left(1 - \frac{R}{d}\right)^2\right)^{\|\mathbf{x}\|_1}\right] - 1 \\
&< \operatorname*{Exp}_R\left[\left(1 + \frac{p}{1-p}\right)^{kR}\left(\frac{R}{pd^2} + 1\right)^k\right] - 1 \\
&< \operatorname*{Exp}_R\left[e^{\frac{pkR}{1-p}} e^{\frac{kR}{pd^2}}\right] - 1 \\
&= \operatorname*{Exp}_R[e^{\gamma R} - 1],
\end{aligned}$$

where $\gamma = \frac{kp}{1-p} + \frac{k}{pd^2}$. ∎

A.2. Well spread hyper-graphs. In the previous section we showed that the probability that $\mathbf{x} \notin \mathbf{T}(\mathcal{Z})$ is at most $\operatorname{Exp}_R[e^{\gamma R}] - 1$. Obviously, the bound is interesting only when $\operatorname{Exp}_R[e^{\gamma R}] < 2$. Notice that this can be true only if

$$\Pr_R\{R = r\} < e^{-\gamma r}$$

for all but a single value of r. Therefore the probability $\Pr_R\{R = r\}$ must decrease exponentially fast in r. This is not necessarily true for any low degree regular hyper-graph $\mathcal{Z}$. In this section we show that if $\mathcal{Z}$ is sufficiently large, then $\mathcal{Z}$ must "contain" a hyper-graph such that

$$\Pr_R\{R = r\} \le 1/r!.$$

More precisely we show that $\mathcal{Z}$ contains a hyper-graph satisfying the following property.

DEFINITION A.4. *Let $(N, \mathcal{Z})$ be a d-regular hyper-graph. $\mathcal{Z}$ is* well spread *if for all $W \subseteq N$ of size at most d, the fraction of hyper-edges containing W is at most*

$$\frac{|\{U \in \mathcal{Z} : W \subseteq U\}|}{|\mathcal{Z}|} \le \frac{1}{d(d-1)\cdots(d-|W|+1)} = \frac{(d-|W|)!}{d!}.$$

Well spread hyper-graphs have the important property that the size of the intersection of two randomly selected hyper-edges is small with very high probability, as shown in the next lemma.

LEMMA A.5. *Let $(N, \mathcal{Z})$ a regular well spread hyper-graph. Choose $U, V \in \mathcal{Z}$ independently and uniformly at random and let $R = |U \cap V|$. For all $r > 0$,*

$$\Pr_R\{R \ge r\} < \frac{1}{r!}.$$

Proof. Let d be the degree of the hyper-graph. We prove that for any fixed set U of size d, the probability that $|U \cap V| \ge r$ when V is chosen at random from $\mathcal{Z}$ is at most $\frac{1}{r!}$. If $|U \cap V| \ge r$ then V contains a subset of U of size r. Therefore, by union bound,

$$\Pr_{V \in \mathcal{Z}}\{|U \cap V| \ge r\} \le \sum_{W \in \binom{U}{r}} \Pr_{V \in \mathcal{Z}}\{W \subseteq V\} = \sum_{W \in \binom{U}{r}} \frac{|\{V \in \mathcal{Z} : W \subseteq V\}|}{|\mathcal{Z}|},$$

where $\binom{U}{r}$ denotes the set of all the size r subsets of U. Since $\mathcal{Z}$ is well spread, the fraction $|\{V \in \mathcal{Z} : W \subseteq V\}|/|\mathcal{Z}|$ is at most $\frac{(d-r)!}{d!}$, which substituted in the previous expression, gives

$$\Pr_{V \in \mathcal{Z}}\{|U \cap V| \geq r\} \leq \binom{d}{r}\frac{(d-r)!}{d!} = \frac{1}{r!}. \qquad \square$$

We now show how to find well spread hyper-graphs "inside" any sufficiently big regular hyper-graph. For any subset $W \subseteq N$, define the induced hyper-graph

$$\mathcal{Z}_W = \{A \subseteq N \setminus W : A \cup W \in \mathcal{Z}\}.$$

In other words, $\mathcal{Z}_W$ is the set of hyper-edges containing W, with the nodes in W removed. Notice the following basic facts:

1. Hyper-graph $\mathcal{Z}$ is well spread if for every set W of size at most d, $|\mathcal{Z}_W| \leq \frac{(d-|W|)!}{d!}|\mathcal{Z}|$.
2. $\mathcal{Z}_W$ is d'-regular with $d' = d - |W|$.
3. If $W = \emptyset$ then $\mathcal{Z}_W = \mathcal{Z}$.
4. $(\mathcal{Z}_W)_V = \mathcal{Z}_{W \cup V}$ if $U \cap V = \emptyset$, and $(\mathcal{Z}_W)_V = \emptyset$ otherwise.
5. If $|W| > d$ then $\mathcal{Z}_W = \emptyset$.

In the following lemma we prove that for any regular hyper-graph $\mathcal{Z}$, there exists a set W such that $\mathcal{Z}_W$ is well spread.

LEMMA A.6. *Let $(N, \mathcal{Z})$ be an h-regular hyper-graph. Then there exists a set $W \subset N$ such that $(N, \mathcal{Z}_W)$ is well spread and $|\mathcal{Z}_W| > |\mathcal{Z}|/h!$.*

Proof. If $(N, \mathcal{Z})$ is well spread, let $W = \emptyset$ and the statement is obviously true. Otherwise, there exists some set W of size at most h such that $|\mathcal{Z}_W| > \frac{(h-|W|)!}{h!} \cdot |\mathcal{Z}|$. Let W be maximal (with respect to the set inclusion ordering relation) among these sets. Obviously, $|\mathcal{Z}_W| > |\mathcal{Z}|/h!$. Notice that $\mathcal{Z}_W$ is d-regular, with $d = h - |W|$. We prove that $(N, \mathcal{Z}_W)$ is well spread. Let V be a subset of N of size at most d. There are three cases:

(i) If $V \cap W \neq \emptyset$ then $|(\mathcal{Z}_W)_V| = 0 \leq \frac{(d-|V|)!}{d!} \cdot |\mathcal{Z}_W|$.
(ii) If $V = \emptyset$, then $|(\mathcal{Z}_W)_V| = |\mathcal{Z}_W| = \frac{d!}{d!} \cdot |\mathcal{Z}_W|$.
(iii) Finally assume $V \neq \emptyset$ and $V \cap W = \emptyset$. By the maximality of W one gets

$$\begin{aligned} |(\mathcal{Z}_W)_V| &= |\mathcal{Z}_{V \cup W}| \\ &\leq \frac{(h-|V \cup W|)!}{h!}|\mathcal{Z}| \\ &= \frac{(d-|V|)!}{d!}\frac{(h-|W|)!}{h!}|\mathcal{Z}| \\ &< \frac{(d-|V|)!}{d!}|\mathcal{Z}_W|. \qquad \square \end{aligned}$$

A.3. Weak probabilistic construction. We now combine the tools developed in the previous sections to prove the following theorem.

THEOREM A.7. *For every sufficiently small constant $\epsilon > 0$, positive integer k and h-regular hyper-graph $(N, \mathcal{Z})$ of size $|\mathcal{Z}| > h!|N|^{\sqrt{h}k/\epsilon}$ the following holds. Define matrix $\mathbf{T} \in \{0,1\}^{k \times |N|}$ at random by setting each entry to 1 independently with probability $p = \frac{\epsilon}{hk}$. Then, for every $\mathbf{x} \in \{0,1\}^k$,*

$$\Pr\{\mathbf{x} \in \mathbf{T}(\mathcal{Z})\} > 1 - 5\epsilon.$$

From Lemma A.6, there exists a subset $W \subset N$ such that $(N, \mathcal{Z}_W)$ is well spread and $|\mathcal{Z}_W| \geq |\mathcal{Z}|/h! > |N|^{\sqrt{h}k/\epsilon}$. Choose $\mathbf{T} \in \{0,1\}^{k \times |N|}$ at random by setting each entry to one independently with probability $p = \frac{\epsilon}{hk}$. Let F be the event that all entries in $\mathbf{T}$ that belongs to the columns corresponding to elements in W are 0. Notice that $\Pr\{\neg F\} \leq |W|kp \leq hkp = \epsilon$. Notice also that

$$\Pr_{\mathbf{T}}\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z}) \mid F\} \leq \Pr_{\mathbf{T}}\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z}_W)\}$$

Let d be the degree of $\mathcal{Z}_W$. Since $|\mathcal{Z}_W| \leq \binom{|N|}{d} < |N|^d$ and $|\mathcal{Z}_W| > |N|^{\sqrt{h}k/\epsilon}$, hyper-graph $\mathcal{Z}_W$ has degree at least $d > \sqrt{h}k/\epsilon$.

Applying Proposition A.3 to d-regular hyper-graph $\mathcal{Z}_W$, the probability (over the choice of $\mathbf{T}$) that $\mathbf{x} \notin \mathbf{T}(\mathcal{Z}_W)$ is at most $\mathrm{Exp}_R[e^{\gamma R}] - 1$, where R is the size of the intersection of two random elements in $\mathcal{Z}_W$ and

$$\begin{aligned}\gamma &= \frac{kp}{1-p} + \frac{k}{pd^2} \\ &= \frac{\epsilon}{h - \epsilon/k} + \frac{hk^2}{\epsilon d^2} \\ &< \frac{\epsilon}{1-\epsilon} + \epsilon.\end{aligned}$$

But $\mathcal{Z}_W$ is well spread, so by lemma A.5, $\Pr_R\{R \geq r\} < 1/r!$ and the expectation $\mathrm{Exp}_R[e^{\gamma R}]$ can be bounded as follows:

$$\begin{aligned}\mathop{\mathrm{Exp}}_R[e^{\gamma R}] &= \sum_{r \geq 0} e^{\gamma r} \Pr_R\{R = r\} \\ &= \sum_{r \geq 0} e^{\gamma r} \left(\Pr_R\{R \geq r\} - \Pr_R\{R \geq r+1\}\right) \\ &= \sum_{r \geq 0} e^{\gamma r} \Pr_R\{R \geq r\} - \sum_{r \geq 1} e^{\gamma(r-1)} \Pr_R\{R \geq r\} \\ &= 1 + (1 - e^{-\gamma}) \sum_{r \geq 1} e^{\gamma r} \Pr_R\{R \geq r\} \\ &< 1 + \gamma \sum_{r \geq 1} \frac{e^{\gamma r}}{r!} \\ &= 1 + \gamma(e^{e^\gamma} - 1).\end{aligned}$$

So, the probability that $\mathbf{x} \notin \mathbf{T}(\mathcal{Z})$ given F is less than $\gamma(e^{e^\gamma} - 1)$ and

$$\begin{aligned}\Pr_{\mathbf{T}}\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z})\} &\leq \Pr\{\neg F\} + \Pr\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z}) \mid F\} \\ &\leq \epsilon + \gamma(e^{e^\gamma} - 1).\end{aligned}$$

Using the bound $\gamma < \epsilon(1 + 1/(1-\epsilon))$, we get that for all sufficiently small ϵ

$$\Pr_{\mathbf{T}}\{\mathbf{x} \notin \mathbf{T}(\mathcal{Z})\} \leq 5\epsilon.$$

A.4. Strong probabilistic construction. We proved that for every boolean vector $\mathbf{x}$, if $\mathbf{T}$ is chosen as described in Theorem A.7, then with high probability there

exists a $U \in \mathcal{Z}$ such that $\mathbf{T}(U) = \mathbf{x}$. It follows by an averaging argument that with high probability the size of $\mathbf{T}(\mathcal{Z}) \cap \{0,1\}^k$ (the set of all boolean vectors that can be represented as $\mathbf{T}(U)$ for some $U \in \mathcal{Z}$) is almost equal to the size of the whole $\{0,1\}^k$. We now show how to project $\mathbf{T}(\mathcal{Z}) \cap \{0,1\}^k$ onto the set of all binary strings of some shorter length.

For any vector $\mathbf{x} \in \{0,1\}^n$ and subset $G \subseteq \{1,\ldots,n\}$, define the projection $\mathbf{x}|_G \in \{0,1\}^{|G|}$ as the vector obtained taking the coordinates of $\mathbf{x}$ with index in G. The projection operation is extended to set of vectors in the obvious way: $\mathcal{W}|_G = \{\mathbf{x}|G : \mathbf{x} \in \mathcal{W}\}$. The next lemma shows that the probability that a random projection $\mathcal{W}|_G$ covers the whole set $\{0,1\}^G$ of binary strings is at least equal to the density of $|\mathcal{W}|$ in $\{0,1\}^n$.

LEMMA A.8. *Let $\mathcal{W}$ be a subset of $\{0,1\}^n$. If G is chosen uniformly at random among all subsets of $\{1,\ldots,n\}$, then*

$$\Pr_G\left\{\mathcal{W}|_G = \{0,1\}^G\right\} \geq \frac{|\mathcal{W}|}{2^n}.$$

Proof. By induction on n. The base case $n = 0$ is trivially true. (Notice that $\{0,1\}^G = \{0,1\}^n = \{\varepsilon\}$ and $\mathcal{W}|_G = \mathcal{W} = \{0,1\}^G$ iff $|\mathcal{W}| = 1$.) So, assume the statement holds for all $\mathcal{W} \subseteq \{0,1\}^n$ and let us prove it for $\mathcal{W} \subseteq \{0,1\}^{n+1}$. Choose G at random and let $G' = G \setminus \{n+1\}$. Notice that G' is a random subset of $\{1,\ldots,n\}$. Define the following sets:

$$\mathcal{W}_0 = \left\{\mathbf{x} : \begin{bmatrix}\mathbf{x}\\0\end{bmatrix} \in \mathcal{W}\right\}, \qquad \mathcal{W}_1 = \left\{\mathbf{x} : \begin{bmatrix}\mathbf{x}\\1\end{bmatrix} \in \mathcal{W}\right\}.$$

Notice that $|\mathcal{W}| = |\mathcal{W}_0| + |\mathcal{W}_1| = |\mathcal{W}_0 \cup \mathcal{W}_1| + |\mathcal{W}_0 \cap \mathcal{W}_1|$. Moreover, if

(i) either $(n+1) \in G$ and $(\mathcal{W}_0 \cap \mathcal{W}_1)|_{G'} = \{0,1\}^{G'}$
(ii) or $(n+1) \notin G$ and $(\mathcal{W}_0 \cup \mathcal{W}_1)|_{G'} = \{0,1\}^{G'}$,

then $\mathcal{W}|_G = \{0,1\}^G$. Therefore, using the inductive hypothesis, we get

$$\begin{aligned}
\Pr\{\mathcal{W}|_G = \{0,1\}^G\} &\geq \Pr\{(n+1) \in G\} \Pr\{(\mathcal{W}_0 \cup \mathcal{W}_1)|_{G'} = 2^{G'}\} \\
&\quad + \Pr\{(n+1) \notin G\} \Pr\{(\mathcal{W}_0 \cap \mathcal{W}_1)|_{G'} = 2^{G'}\} \\
&\geq \frac{1}{2}\left(\frac{|\mathcal{W}_0 \cup \mathcal{W}_1|}{2^n}\right) + \frac{1}{2}\left(\frac{|\mathcal{W}_0 \cap \mathcal{W}_1|}{2^n}\right) \\
&= \frac{|\mathcal{W}_0 \cup \mathcal{W}_1| + |\mathcal{W}_0 \cap \mathcal{W}_1|}{2^{n+1}} \\
&= \frac{|\mathcal{W}|}{2^{n+1}}. \qquad \square
\end{aligned}$$

Now, we can easily derive Theorem 5.9 from Lemma A.8 and Theorem A.7. Instead of choosing the matrix $\mathbf{T} \in \{0,1\}^{k \times m}$ as specified in Theorem 5.9, we do the following mental experiment. First choose a bigger matrix $\mathbf{T}' \in \{0,1\}^{4k \times n}$ at random by setting each entry to 1 independently with probability $p = \frac{4\epsilon}{dk}$. Then choose a random subset $G \subseteq 1,\ldots,4k$ of its rows. If G has size at least k, set $\mathbf{T}$ to the submatrix of $\mathbf{T}'$ with rows corresponding to the first k elements of G. (If G has less than k elements, the experiment fails.)

Let $\mathcal{W} = \mathbf{T}'(\mathcal{Z}) \cap \{0,1\}^{4k}$. Notice that the probability distribution of matrix $\mathbf{T}$ (conditioned on the event $|G| \geq k$) is the same as in Theorem 5.9. Moreover, if $|G| \geq k$ and $\{0,1\}^G \subseteq \mathcal{W}|_G$ then $\{0,1\}^k \subseteq \mathbf{T}(\mathcal{Z})$. So, we can bound the probability

that matrix $\mathbf{T}$ does not satisfy Theorem 5.9 as the sum of the probabilities that $|G| < k$ and $\{0,1\}^k \not\subseteq \mathbf{T}(\mathcal{Z})$.

Notice that $\mathrm{Exp}[|G|] = 2k$ and $\mathrm{Var}[|G|] = k$. So, by Chebychev's inequality

$$\begin{aligned}\Pr\{|G| < k\} &< \Pr\{||G| - \mathrm{Exp}[|G|]| < k\} \\ &< \frac{\mathrm{Var}[|G|]}{k^2} \\ &= \frac{1}{k} < \epsilon\end{aligned}$$

for all sufficiently large k. Now, let us bound the probability that $\{0,1\}^G \subseteq \mathcal{W}|_G$ when G and $\mathbf{T}'$ are chosen at random. Using Lemma A.8 and the independence of G and $\mathbf{T}'$, one gets

$$\begin{aligned}\Pr_{G,\mathbf{T}'}\{\{0,1\}^G \subseteq \mathcal{W}|_G\} &= \mathop{\mathrm{Exp}}_{\mathbf{T}'}[\Pr_G\{\{0,1\}^G \subseteq \mathcal{W}|_G\}] \\ &\geq \mathop{\mathrm{Exp}}_{\mathbf{T}'}\left[\frac{|\mathcal{W}|}{2^{4k}}\right] \\ &= \mathop{\mathrm{Exp}}_{\mathbf{T}'}\left[\Pr_{\mathbf{x}\in\{0,1\}^{4k}}[\mathbf{x} \in \mathcal{W}]\right] \\ &= \mathop{\mathrm{Exp}}_{\mathbf{x}\in\{0,1\}^{4k}}\left[\Pr_{\mathbf{T}'}[\mathbf{x} \in \mathbf{T}'(\mathcal{Z})]\right] \\ &\geq \min_{\mathbf{x}\in\{0,1\}^{4k}} \Pr_{\mathbf{T}'}\{\mathbf{x} \in \mathbf{T}'(\mathcal{Z})\} \\ &\geq 1 - 5\epsilon.\end{aligned}$$

Therefore the probability that $\{0,1\}^k \not\subseteq \mathbf{T}(\mathcal{Z})$ is at most 5ϵ. By union bound, with probability at least $1 - 6\epsilon$ matrix $\mathbf{T}$ satisfies Theorem 5.9.

Acknowledgments. Many people contributed to this work with useful discussions and comments. I wish to thank all of them. I will not attempt to list them all here, but some of them deserve a special mention. Special thanks to Shafi Goldwasser who convinced me that SVP was NP-hard to approximate and encouraged me to work on this problem, Oded Goldreich who patiently listened to my proofs when I had not even started writing this paper and actively contributed to the proof of Theorem 5.9, and Muli Safra who suggested an important simplification in Lemma 5.2. Thanks also to Miklós Ajtai, whose work has largely inspired the results presented in this paper.

REFERENCES

[1] L. M. Adleman, *Factoring and Lattice Reduction*, manuscript, 1995.

[2] M. Ajtai, *The shortest vector problem in L_2 is NP-hard for randomized reductions (extended abstract)*, in Proceedings of the 30th Annual ACM Symposium on Theory of Computing, Dallas, TX, 1998, pp. 10–19.

[3] S. Arora, L. Babai, J. Stern, and E. Z. Sweedyk, *The hardness of approximate optima in lattices, codes, and systems of linear equations*, J. Comput. System Sci., 54 (1997), pp. 317–331. Preliminary version in FOCS'93.

[4] L. Babai, *On Lovasz' lattice reduction and the nearest lattice point problem*, Combinatorica, 6 (1986), pp. 1–13.

[5] W. Banaszczyk, *New bounds in some transference theorems in the geometry of numbers*, Math. Ann., 296 (1993), pp. 625–635.

[6] M. Bellare, S. Goldwasser, and D. Micciancio, *"Pseudo-random" generators within cryptographic applications: the DSS case*, in Advances in Cryptology—CRYPTO '97, B. S.

Kaliski Jr., ed., Lecture Notes in Comput. Sci. 1294, Springer-Verlag, New York, 1997, pp. 277–291.

[7] D. Coppersmith, *Small solutions to polynomial equations, and low-exponent RSA vulnerabilities*, J. Cryptology, 10 (1997), pp. 233–260.

[8] M. J. Coster, A. Joux, B. A. LaMacchia, A. M. Odlyzko, C.-P. Schnorr, and J. Stern, *Improved low-density subset sum algorithms*, Comput. Complexity, 2 (1992), pp. 111–128.

[9] I. Dinur, *Approximating SVP_∞ to within almost-polynomial factors is NP-hard*, in Algorithms and Complexity, 4th Italian Conference, CIAC 2000, Proceedings, G. Bongiovanni, G. Gambosi, and R. Petreschi, eds., Lecture Notes in Comput. Sci. 1767, Rome, Italy, 2000, Springer-Verlag, New York, pp. 263–276.

[10] I. Dinur, G. Kindler, and S. Safra, *Approximating CVP to within almost-polynomial factors is NP-hard*, in 39th Annual IEEE Symposium on Foundations of Computer Science, Palo Alto, CA, 1998.

[11] I. Dumer, D. Micciancio, and M. Sudan, *Hardness of approximating the minimum distance of a linear code*, in 40th Annual IEEE Symposium on Foundations of Computer Science, New York, 1999, pp. 475–484.

[12] A. M. Frieze, *On the Lagarias-Odlyzko algorithm for the subset sum problem*, SIAM J. Comput., 15 (1986), pp. 536–539.

[13] A. M. Frieze, J. Håstad, R. Kannan, J. C. Lagarias, and A. Shamir, *Reconstructing truncated integer variables satisfying linear congruences*, SIAM J. on Comput., 17 (1988), pp. 262–280.

[14] O. Goldreich, *private communication*, 1999.

[15] O. Goldreich and S. Goldwasser, *On the limits of nonapproximability of lattice problems*, J. Comput. System Sci., 60 (2000), pp. 540–563. Preliminary version in STOC'98.

[16] O. Goldreich, D. Micciancio, S. Safra, and J.-P. Seifert, *Approximating shortest lattice vectors is not harder than approximating closest lattice vectors*, Inform. Process. Lett., 71 (1999), pp. 55–61.

[17] J. Håstad, *Solving simultaneous modular equations of low degree*, SIAM J. Comput., 17 (1988), pp. 336–341. Preliminary version in Crypto85.

[18] D. S. Johnson, *A catalog of complexity classes*, in Handbook of Theoretical Computer Science, vol. A (Algorithms and Complexity), Elsevier Science, Amsterdam, 1990, chap. 2, pp. 67–161.

[19] R. Kannan, *Algorithmic geometry of numbers*, in Annual Reviews of Computer Science, Vol. 2, Annual Review Inc., Palo Alto, CA, 1987, pp. 231–267.

[20] R. Kannan, *Minkowski's convex body theorem and integer programming*, Math. Oper. Res., 12 (1987), pp. 415–440.

[21] R. M. Karp and R. J. Lipton, *Some connections between nonuniform and uniform complexity classes*, in Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, Los Angeles, CA, 1980, pp. 28–30. Appeared in journal form as R. M. Karp and R. J. Lipton, *Turing machines that take advice*, Enseign. Math., 28 (1982) pp. 191–209.

[22] J. C. Lagarias, H. W. Lenstra, Jr., and C.-P. Schnorr, *Korkine-Zolotarev bases and successive minima of a lattice and its reciprocal lattice*, Combinatorica, 10 (1990), pp. 333–348.

[23] J. C. Lagarias and A. M. Odlyzko, *Solving low-density subset sum problems*, J. ACM, 32 (1985), pp. 229–246.

[24] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, *Factoring polynomials with rational coefficients*, Math. Ann., 261 (1982), pp. 513–534.

[25] H. W. Lenstra, *Integer programming with a fixed number of variables*, Math. Oper. Res., 8 (1983), pp. 538–548.

[26] A. R. Meyer and L. J. Stockmeyer, *The equivalence problem for regular expression with squaring requires exponential space*, in Proceedings of the 13th IEEE Symposium on Switching and Automata Theory, 1972, pp. 25–29.

[27] D. Micciancio, *On the Hardness of the Shortest Vector Problem*, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, 1998.

[28] D. Micciancio, *The hardness of the closest vector problem with preprocessing*, IEEE Trans. Inform. Theory, 47 (2001).

[29] C.-P. Schnorr, *A hierarchy of polynomial time lattice basis reduction algorithms*, Theoret. Comput. Sci., 53 (1987), pp. 201–224.

[30] C.-P. Schnorr, *Factoring integers and computing discrete logarithms via Diophantine approximation*, in Advances in Computational Complexity 13, DIMACS Ser. Discrete Math. Theoret. Comput. Sci., AMS, Providence, RI, 1993, pp. 171–182. Preliminary version in Eurocrypt'91, Springer-Verlag, LNCS 547.

[31] A. Schönhage, *Factorization of univariate integer polynomials by Diophantine approximation and an improved basis reduction algorithm*, in Automata, Languages and Programming, 11th Colloquium, J. Paredaens, ed., Lecture Notes in Comput. Sci. 172, Antwerp, Belgium, 1984, Springer-Verlag, New York, pp. 436–447.

[32] A. Shamir, *A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem*, IEEE Trans. Inform. Theory, 30 (1984), pp. 699–704. Preliminary version in FOCS'82.

[33] L. J. Stockmeyer, *The polynomial-time hierarchy*, Theoret. Comput. Sci., 3 (1977), pp. 1–22.

[34] P. van Emde Boas, *Another NP-Complete Problem and the Complexity of Computing Short Vectors in a Lattice*, Tech. Report 81-04, Mathematische Instituut, Universiry of Amsterdam, 1981; also available online from http://turing.wins.uva.nl/∼peter/.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 2036–2050

DUAL-BOUNDED GENERATING PROBLEMS: PARTIAL AND MULTIPLE TRANSVERSALS OF A HYPERGRAPH*

ENDRE BOROS†, VLADIMIR GURVICH†, LEONID KHACHIYAN‡, AND KAZUHISA MAKINO§

Abstract. We consider two generalizations of the notion of transversal to a finite hypergraph, the so-called *multiple* and *partial* transversals. Multiple transversals naturally arise in 0-1 programming, while partial transversals are related to data mining and machine learning. We show that for an arbitrary hypergraph the families of multiple and partial transversals are both dual-bounded in the sense that the size of the corresponding dual hypergraph is bounded by a polynomial in the cardinality and the length of description of the input hypergraph. Our bounds are based on new inequalities of extremal set theory and threshold Boolean logic, which may be of independent interest. We also show that the problems of generating all multiple and all partial transversals for a given hypergraph are polynomial-time reducible to the generation of all ordinary transversals for another hypergraph, i.e., to the well-known dualization problem for hypergraphs. As a corollary, we obtain incremental quasi-polynomial-time algorithms for both of the above problems, as well as for the generation of all the minimal binary solutions for an arbitrary monotone system of linear inequalities.

Key words. transversal, independent set, hypergraph, dualization, monotone Boolean function, incremental polynomial time, threshold function, data mining, maximal frequent set, minimal infrequent set

AMS subject classifications. 68R05, 68Q25, 68Q32

PII. S0097539700370072

1. Introduction.

In this paper we consider problems involving the generation of all sets of some implicitly given set families. The most well-known problem of this type, the generation of all minimal *transversals* of a given hypergraph, has applications in combinatorics [32], graph theory [21, 23, 33, 35], artificial intelligence [13], game theory [16, 17, 31], reliability theory [10, 31], database theory [1, 27, 36], and learning theory [2, 12].

Given a finite set V of $n = |V|$ points and a hypergraph (set family) $\mathcal{A} \subseteq 2^V$, a subset $B \subseteq V$ is called a *transversal* of the family $\mathcal{A}$ if $A \cap B \neq \emptyset$ for all sets $A \in \mathcal{A}$; it is called a *minimal transversal* if no proper subset of B is a transversal of $\mathcal{A}$. The hypergraph $\mathcal{A}^d$ consisting of all minimal transversals of $\mathcal{A}$ is called the *dual* (or *transversal*) hypergraph of $\mathcal{A}$. It is easy to see that if $A \in \mathcal{A}$ is not minimal in $\mathcal{A}$, that is, if $A' \subset A$ for some $A' \in \mathcal{A}$, then $(\mathcal{A} \setminus \{A\})^d = \mathcal{A}^d$. We can therefore assume that all sets in $\mathcal{A}$ are minimal; in other words, the hypergraph $\mathcal{A}$ is Sperner. (The dual hypergraph $\mathcal{A}^d$ is Sperner by definition.) It is then easy to verify that $(\mathcal{A}^d)^d = \mathcal{A}$ and $\bigcup_{A\in\mathcal{A}} A = \bigcup_{B\in\mathcal{A}^d} B$.

*Received by the editors April 4, 2000; accepted for publication November 22, 2000; published electronically April 18, 2001. A preliminary version of this paper appeared in [7].
http://www.siam.org/journals/sicomp/30-6/37007.html
†RUTCOR, Rutgers University, 640 Bartholomew Road, Piscataway, NJ 08854 (boros@rutcor.rutgers.edu, gurvich@rutcor.rutgers.edu). The research of the first two authors was supported in part by Office of Naval Research grant N00014-92-J-1375, National Science Foundation grant DMS 98-06389, and DIMACS.
‡Department of Computer Science, Rutgers University, 110 Frelinghuysen Road, Piscataway, NJ 08854 (leonid@cs.rutgers.edu). The research of this author was supported in part by National Science Foundation grant CCR-9618796.
§Division of Systems Science, Graduate School of Engineering Science, Osaka University, Toyonaka, Osaka 560-8531, Japan (makino@sys.es.osaka-u.ac.jp).

For a subset $X \subseteq V$, let $X^c = V \setminus X$ denote the complement of X and let $\mathcal{A}^c = \{A^c | A \in \mathcal{A}\}$ be the *complementary* hypergraph of $\mathcal{A}$. Then, for instance, $\mathcal{A}^{dc}$ consists of all maximal subsets containing no hyperedge of $\mathcal{A}$, while the hypergraph $\mathcal{A}^{cd}$ consists of all minimal subsets of V which are not contained in any hyperedge of $\mathcal{A}$.

1.1. Dualization. Given a Sperner hypergraph $\mathcal{A}$, a frequently arising task is the generation of the transversal hypergraph $\mathcal{A}^d$. This problem, known as *dualization*, can be stated as follows: *Given a complete list of all hyperedges of $\mathcal{A}$ and a set of minimal transversals $\mathcal{B} \subseteq \mathcal{A}^d$, either prove that $\mathcal{B} = \mathcal{A}^d$, or find a new transversal $X \in \mathcal{A}^d \setminus \mathcal{B}$.*

Clearly, we can generate all of the hyperedges of $\mathcal{A}^d$ by initializing $\mathcal{B} = \emptyset$ and iteratively solving the above problem $|\mathcal{A}^d| + 1$ times. Also note that in general, $|\mathcal{A}^d|$ can be exponentially large in both $|\mathcal{A}|$ and $|V|$. For this reason, the complexity of generating $\mathcal{A}^d$ is customarily measured in the input and output sizes. In particular, we say that $\mathcal{A}^d$ can be generated in *incremental polynomial time* if the dualization problem can be solved in time, polynomial in $|V|, |\mathcal{A}|$, and $|\mathcal{B}|$.

The dualization problem can be efficiently solved for many classes of hypergraphs. For example, if the sizes of all the hyperedges of $\mathcal{A}$ are limited by a constant r, then dualization can be executed in incremental polynomial time (see, e.g., [6, 12, 13]), and it can also be done efficiently in parallel (see [8]). In the quadratic case, i.e., when $r = 2$, there are dualization algorithms that run with *polynomial delay*, i.e., in $poly(|V|, |\mathcal{A}|)$ time, where $\mathcal{B}$ is systematically enlarged from $\mathcal{B} = \emptyset$ during the generation process of $\mathcal{A}^d$ (see, e.g., [21, 23, 35]). Efficient algorithms also exist for the dualization of 2-monotonic, threshold, matroid, read-bounded, acyclic, and some other classes of hypergraphs (see, e.g., [4, 9, 11, 26, 29, 30]).

Even though no incremental polynomial-time algorithm for the dualization of arbitrary hypergraphs is known, an incremental *quasi-polynomial-time* algorithm exists (see [14]). This algorithm solves the dualization problem in $O(nm) + m^{o(\log m)}$ time, where $n = |V|$ and $m = |\mathcal{A}| + |\mathcal{B}|$ (see also [19] for more detail).

In this paper, we consider two natural generalizations of minimal transversals, so-called *multiple transversals* and *partial transversals.* See section 5 for related hypergraphs in the data mining and machine learning literature.

1.2. Multiple transversals. Given a hypergraph $\mathcal{A} \subseteq 2^V$ and a nonnegative weight b_A associated with every hyperedge $A \in \mathcal{A}$, a subset X is called a *multiple transversal* (or *b-transversal*) if $|X \cap A| \geq b_A$ holds for all $A \in \mathcal{A}$. The family of all minimal b-transversals can then also be viewed as the family of support sets of minimal feasible binary solutions to the system of inequalities

$$Ax \geq b, \tag{1.1}$$

where the rows of $A = A_{\mathcal{A}}$ are exactly the characteristic vectors of the hyperedges $A \in \mathcal{A}$ and the corresponding component of b is equal to b_A. Clearly, $b = (1, 1, \ldots, 1)$ corresponds to the case of (ordinary) transversals in which case (1.1) is also known as a *set covering* problem.

Generalizing further and giving up the binary nature of A as well, we shall consider the family $\mathcal{F} = \mathcal{F}_{A,b}$ of (support sets of) all minimal feasible binary vectors to (1.1) for a given $m \times n$-matrix A and a given m-vector b. We assume that (1.1) is a *monotone* system of inequalities: if $x \in \{0,1\}^n$ satisfies (1.1), then any vector $y \in \{0,1\}^n$ such that $y \geq x$ is also feasible. For instance, (1.1) is monotone if A is

nonnegative. Note that for a monotone system (1.1) the dual hypergraph $\mathcal{F}^d = \mathcal{F}^d_{A,b}$ is (the complementarity hypergraph of) the collection of (supports of) all maximal infeasible vectors for (1.1). In what follows, we assume that the hypergraph $\mathcal{F}_{A,b}$ is represented by the system (1.1) and not given explicitly, i.e., by a list of all its hyperedges. In particular, this means that the generation of $\mathcal{F}_{A,b}$ and its dual $\mathcal{F}^d_{A,b}$ are both nontrivial.

Let us consider in general a Sperner hypergraph $\mathcal{F} \subseteq 2^V$ on a finite set V represented in some implicit way and let GEN($\mathcal{F}$) denote the problem of generating all the hyperedges of $\mathcal{F}$: *Given $\mathcal{F}$ and a list of hyperedges $\mathcal{H} \subseteq \mathcal{F}$, either prove that $\mathcal{H} = \mathcal{F}$ or find a new hyperedge in $\mathcal{F} \setminus \mathcal{H}$.*

It is known that problem GEN($\mathcal{F}^d_{A,b}$) is NP-hard even for binary matrices A (see [24]). In contrast, we show that the tasks of generating multiple and ordinary transversals are polynomially related.

THEOREM 1.1. *Problem GEN($\mathcal{F}_{A,b}$) is polytime reducible to dualization.*

In particular, for any monotone system of linear inequalities (1.1), all minimal binary solutions of (1.1) can be generated in quasi-polynomial incremental time. It is also easy to see that if the number of nonzero coefficients in each inequality of $Ax \geq b$ is bounded, then problem GEN($\mathcal{F}_{A,b}$) can be solved in incremental polynomial time.

REMARK 1. *Even though generating all maximal infeasible binary points for* (1.1) *is hard, there is a polynomial randomized scheme for nearly uniform sampling from the set of all binary infeasible points for arbitrary, not necessarily monotone, system* (1.1). *Such a scheme can be obtained by combining the algorithm* [22] *for approximating the size of set-unions with the rapidly mixing random walk* [28] *on the binary cube truncated by a single linear inequality. On the other hand, a similar randomized scheme for nearly uniform sampling from within the set of all binary (or all minimal binary) solutions to a given monotone system* (1.1) *would imply that any NP-complete problem can be solved in polynomial time by a randomized algorithm with arbitrarily small one-sided failure probability. By using the amplification technique of* [20]*, this can already be shown for systems* (1.1) *with two nonzero coefficients per inequality; see* [18] *for more detail.*

1.3. Partial transversals. Given a hypergraph $\mathcal{A} \subseteq 2^V$ and a nonnegative threshold $k < |\mathcal{A}|$, a subset $X \subseteq V$ is said to be a *partial transversal*, or more precisely, a *k-transversal*, to the family $\mathcal{A}$ if it intersects all but at most k of the subsets of $\mathcal{A}$, i.e., if $|\{A \in \mathcal{A} | A \cap X = \emptyset\}| \leq k$.

Denote by $\mathcal{A}^{d_k}$ the family of all minimal k-transversals of $\mathcal{A}$. Clearly, 0-transversals are exactly the standard transversals defined above, i.e., $\mathcal{A}^{d_0} = \mathcal{A}^d$. In what follows, we assume that the hypergraph $\mathcal{A}^{d_k}$ is represented by a list of all the edges of $\mathcal{A}$ along with the value of $k \in \{0, 1, \ldots, |\mathcal{A}| - 1\}$.

Define a *k-union* from $\mathcal{A}$ as the union of some k subsets of $\mathcal{A}$ and let $\mathcal{A}^{u_k}$ denote the family of all minimal k-unions of $\mathcal{A}$. In other words, $\mathcal{A}^{u_k}$ is the family of all the minimal subsets of V which contain at least k hyperedges of $\mathcal{A}$. By the above definitions, k-union and k-transversal families are both Sperner (even if the input hypergraph $\mathcal{A}$ is not). It is also easy to see that the families of all minimal k-transversals and $(k+1)$-unions are in fact dual:

$$\mathcal{A}^{d_k} = (\mathcal{A}^{u_{k+1}})^d, \quad k = 0, 1, \ldots, |\mathcal{A}| - 1.$$

The tasks of generating partial and ordinary transversals also turn out to be polynomially equivalent.

THEOREM 1.2. *Problem GEN($\mathcal{A}^{d_k}$) is polytime reducible to dualization.*

It should be mentioned that the dual problem $\mathrm{GEN}(\mathcal{A}^{u_{k+1}})$ is NP-hard (see [25]).

1.4. Bounding dual hypergraphs. Our proofs of Theorems 1.1 and 1.2 make use of the fact that the Sperner hypergraphs $\mathcal{F}_{A,b}$ and $\mathcal{A}^{d_k}$ are *dual-bounded* in the sense that in both cases, the size of the dual hypergraph can be bounded by a polynomial in the size and the length of description of the primal hypergraph.

THEOREM 1.3. *For any monotone system* (1.1) *of m linear inequalities in n binary variables,*

$$|\mathcal{F}^d_{A,b}| \le mn|\mathcal{F}_{A,b}|.$$

Moreover,

$$|\mathcal{H}^d \cap \mathcal{F}^d_{A,b}| \le mn|\mathcal{H}| \quad \text{for any } \mathcal{H} \subseteq \mathcal{F}_{A,b}. \tag{1.2}$$

THEOREM 1.4. *For any hypergraph $\mathcal{A} \subseteq 2^V$ of $m = |\mathcal{A}|$ hyperedges and any threshold $k = 0, \dots, m-1$, we have*

$$|\mathcal{A}^{u_{k+1}}| \le (m-k)|\mathcal{A}^{d_k}|.$$

Moreover, for any hypergraph $\mathcal{H} \subseteq \mathcal{A}^{d_k}$,

$$|\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}| \le (m-k)|\mathcal{H}|. \tag{1.3}$$

We derive Theorem 1.3 from the following lemma.

LEMMA 1.5. *Let $h : \{0,1\}^n \to \{0,1\}$ be a monotone Boolean function such that*

$$h(x) = 1 \text{ and } x \in \{0,1\}^n \Rightarrow wx \stackrel{\text{def}}{=} \sum_{i=1}^n w_i x_i \ge t,$$

where $w = (w_1, \dots, w_n)$ is a given weight vector and t is a threshold. If $h \not\equiv 0$, then

$$|\max F(h) \cap \{x \mid wx < t\}| \le \sum_{x \in \min T(h)} ex,$$

where $\max F(h) \subset \{0,1\}^n$ is the set of all maximal false points of h, $\min T(h) \subseteq \{0,1\}^n$ is the set of all minimal true points of h, and e is the vector of all ones. In particular,

$$|\max F(h) \cap \{x \mid wx < t\}| \le n|\min T(h)|.$$

If the function h is threshold ($h(x) = 1 \Leftrightarrow wx \ge t$), then $|\max F(h)| \le n|\min T(h)|$ and, by symmetry, $|\min T(h)| \le n|\max F(h)|$ hold, which are well-known inequalities (see [4, 11, 29, 30]). Lemma 1.5 thus extends the above threshold inequalities to arbitrary monotone functions h.

As we shall see, Theorem 1.4 can be derived from the following combinatorial inequality.

LEMMA 1.6. *Given a base set U of size $|U| = m$ and a threshold $k \in \{0, \dots, m-1\}$, let $\mathcal{S} = \{S_1, \dots, S_\alpha\}$ and $\mathcal{T} = \{T_1, \dots, T_\beta\}$ be two families of subsets of U for which*

$$\begin{aligned} |S| &\ge k+1 \quad \text{for all } S \in \mathcal{S}, \\ |T| &\le k \quad \text{for all } T \in \mathcal{T}, \end{aligned} \tag{1.4}$$

and such that for each of the $\alpha(\alpha-1)/2$ *pairs* $S', S'' \in \mathcal{S}$ *there exists a* $T \in \mathcal{T}$ *with* $S' \cap S'' \subseteq T$. *Then*

$$\alpha \leq (m-k)\beta \tag{1.5}$$

whenever $\alpha \geq 2$.

Let us first remark that if $\alpha = 1$, then the family $\mathcal{T}$ might be empty, which would violate the inequality (1.5); otherwise $\beta \geq 1$ must hold whenever $\alpha \geq 2$, since at least one set is needed to cover the intersections of the $\binom{\alpha}{2} \geq 1$ pairs of sets from $\mathcal{S}$. In addition, the assumptions of the lemma imply that the family $\mathcal{S}$ must be Sperner, since $S_i \subseteq S_j$ for some $i \neq j$ would give $S_i = S_i \cap S_j \subseteq T_l$ for some $T_l \in \mathcal{T}$, contradicting (1.4). We can further assume the family $\mathcal{T}$ to be Sperner, without any loss of generality, for otherwise we could replace $\mathcal{T}$ by the family of all maximal sets of $\mathcal{T}$.

Let us mention next the special case of Lemma 1.6 for $\mathcal{T} = \mathcal{S}^{\cap}$, where $\mathcal{S}^{\cap}$ is the family of all the maximal subsets of U which can be obtained as the intersection of two distinct hyperedges of $\mathcal{S}$. For a Sperner hypergraph $\mathcal{S}$ on m vertices such that each hyperedge of $\mathcal{S}$ contains at least $k+1$ vertices and $|S' \cap S''| \leq k$ for any two distinct hyperedges S', S'' of $\mathcal{S}$, Lemma 1.6 yields the inequality (cf. [7])

$$|\mathcal{S}| \leq (m-k)|\mathcal{S}^{\cap}|. \tag{1.6}$$

We discuss some generalizations of this inequality in section 4.4. (Several other inequalities on hypergraphs with restricted intersections can be found in Chapter 4 of [3].)

Let us remark finally that the thresholdness condition (1.4) is essential for the validity of Lemma 1.6—without (1.4) the size of $\mathcal{S}$ can be exponentially larger than that of $\mathcal{S}^{\cap}$. In section 4.2 we give examples of Sperner hypergraphs $\mathcal{S}$ on m vertices for which $|\mathcal{S}^{\cap}| = m/5$ and $|\mathcal{S}| = 3^{m/5} + 2m/5$ or $|\mathcal{S}^{\cap}| = (m-2)^2/9$ and $|\mathcal{S}| = 3^{(m-2)/3} + 2(m-2)/3$.

The remainder of the paper is organized as follows. In section 2 we discuss the complexity of jointly generating a pair of dual hypergraphs defined via a superset oracle. For a polynomial-time superset oracle the above problem reduces to dualization. This reduction, along with the bounds stated in Theorems 1.3 and 1.4, yield Theorems 1.1 and 1.2. In section 3 we consider minimal 0-1 solutions to systems of monotone linear inequalities and prove Lemma 1.5 and Theorem 1.3. Section 4 deals with partial transversals and proves Lemma 1.6 and Theorem 1.4. Finally, section 5 discusses some of the related set families and results, and section 6 contains our concluding remarks.

2. Joint and separate generation of dual hypergraphs.

2.1. Superset oracles. Let $\mathcal{G} \subseteq 2^V$ be a Sperner hypergraph on n vertices. In many applications, $\mathcal{G}$ is represented by a *superset oracle* and not given explicitly. Such an oracle can be viewed as an algorithm which, given an input description $\mathfrak{D}$ of $\mathcal{G}$ and a vertex set $X \subseteq V$, can decide whether or not X contains a hyperedge of $\mathcal{G}$. We assume that the length $|\mathfrak{D}|$ of the input description of $\mathcal{G}$ is at least n and denote by $T_s = T_s(|\mathfrak{D}|)$ the worst-case running time of the oracle on any superset query "Does X contain a hyperedge of $\mathcal{G}$?" In particular, $\mathfrak{D}$ is polynomial-time if $T_s \leq poly(|\mathfrak{D}|)$. In what follows, we do not distinguish the superset oracle and the input description $\mathfrak{D}$ of $\mathcal{G}$. Note that a vertex set X contains a hyperedge of $\mathcal{A}^d$ if and only if the complement

of X contains no hyperedge of $\mathcal{A}$. For this reason, $\mathfrak{O}$ also specifies (a superset oracle for) the dual hypergraph $\mathcal{G}^d$. We list below several simple examples.

(1) *Multiple transversals.* Let (1.1) be a monotone system of linear inequalities and let $\mathcal{G} = \mathcal{F}_{A,b}$ be the hypergraph introduced in section 1.2. Then the input description $\mathfrak{O}$ is (A, b). Clearly, for any input set $X \subseteq V$, we can decide whether X contains a hyperedge of $\mathcal{F}_{A,b}$ by checking the feasibility of (the characteristic vector of) X for (1.1).

(2) *Partial transversals.* Let $\mathcal{G} = \mathcal{A}^{d_k}$ be the hypergraph of the minimal k-transversals of a family $\mathcal{A}$ (see section 1.2). Then $\mathcal{G}$ is given by the threshold value k and a complete list of all hyperedges of $\mathcal{A}$, i.e., $\mathfrak{O} \sim (k, \mathcal{A})$. For a subset $X \subseteq V$, determining whether X contains a hyperedge in $\mathcal{A}^{d_k}$ is equivalent to checking if X is intersecting at least $|\mathcal{A}| - k$ hyperedges of $\mathcal{A}$.

(3) *Monotone Boolean formulae.* Let f be a $(\vee, \wedge)$-formula with n variables and let $\mathcal{G} = \mathcal{A}_f$ be the supporting sets of all the minimal true vectors for f. Then $\mathfrak{O} \sim f$ and the superset oracle checks if (the characteristic vector of) $X \subseteq V$ satisfies f. The dual hypergraph $\mathcal{G}^d$ is the set of all the (complements to the support sets of) maximal false vectors of f.

(4) *Two-terminal connectivity.* Consider a digraph Γ with a source s and a sink t, each arc of which is assigned a *relay* $r \in V$ (two or more distinct edges may be assigned identical relays). Let $\mathcal{G}$ be the set of *relay s-t paths*, i.e., minimal subsets of relays that connect s and t. Then $\mathfrak{O} \sim \Gamma$, and for a given relay set $X \subseteq V$, the superset oracle can use breadth-first search to check the reachability of t from s via a path consisting of relays in X. Note that the dual hypergraph $\mathcal{G}^d$ is the set of all relay *s-t cuts*, i.e., minimal subsets of relays that disconnect s and t.

(5) *Helly's systems of polyhedra.* Consider a family of n convex polyhedra $P_i \subseteq \mathbb{R}^r$, $i \in V$, and let $\mathcal{G}$ denote the minimal subfamilies with no point in common. Then $\mathcal{G}^{dc}$ is the family of all maximal subfamilies with a nonempty intersection. (In particular, if $P_1, \ldots, P_n$ are the facets of a convex polytope Q, then $\mathcal{G}^{dc}$ corresponds to the set of vertices of Q.) We have $\mathfrak{O} \sim (P_1, \ldots, P_n)$ and, given subsets of polytopes $X \subseteq V$, the superset oracle can use linear programming to check whether $\cap_{i \in X} P_i \neq \emptyset$.

2.2. Joint generation of dual pairs of hypergraphs. In all of the above examples, we have pairs of dual Sperner hypergraphs given by polynomial-time superset oracles. Let $\mathcal{G}, \mathcal{G}^d \subseteq 2^V$ be a pair of dual Sperner hypergraphs given by a superset oracle $\mathfrak{O}$. Consider the problem GEN$(\mathcal{G}, \mathcal{G}^d)$ of *generating jointly* all the hyperedges of $\mathcal{G}$ and $\mathcal{G}^d$: *Given two explicitly listed set families $\mathcal{A} \subseteq \mathcal{G}$ and $\mathcal{B} \subseteq \mathcal{G}^d$, either prove that these families are complete, $(\mathcal{A}, \mathcal{B}) = (\mathcal{G}, \mathcal{G}^d)$, or find a new set in $(\mathcal{G} \setminus \mathcal{A}) \cup (\mathcal{G}^d \setminus \mathcal{B})$.*

For the special case when $\mathcal{A} = \mathcal{G}$ and $\mathfrak{O}$ is a list of all the sets in $\mathcal{G}$, we obtain the dualization problem as stated in section 1.1. In fact, as observed in [5, 18], for any polynomial-time superset oracle $\mathfrak{O}$, problem GEN$(\mathcal{G}, \mathcal{G}^d)$ can be reduced in polynomial time to dualization. This can be done via the following Algorithm $\mathcal{J}$.

Step 1. Check whether each element of $\mathcal{B}$ is a minimal transversal to $\mathcal{A}$, i.e., $\mathcal{B} \subseteq \mathcal{A}^d$. (Recall that $\mathcal{A}$ and $\mathcal{B}$ are given explicitly.) Note that each set $X \in \mathcal{B}$ is a transversal to $\mathcal{A}$ because $\mathcal{A} \subseteq \mathcal{G}$ and $\mathcal{B} \subseteq \mathcal{G}^d$. If some transversal $X \in \mathcal{B}$ is not *minimal* for $\mathcal{A}$, then we can easily find a proper subset Y of X such that Y is also a transversal to $\mathcal{A}$. Since Y is a proper subset of X, and X is a minimal transversal to $\mathcal{G}$, Y must miss some hyperedges of $\mathcal{G}$. Hence Y^c, the complement of Y, contains a hyperedge of $\mathcal{G}$. By querying the superset oracle $\mathfrak{O}$ at most $|Y^c|$ times we can find such a hyperedge $Z \in \mathcal{G}$. Note that $Z \cap Y = \emptyset$, whereas $A \cap Y \neq \emptyset$ for all hyperedges $A \in \mathcal{A}$. This means that

Z is a new hyperedge of $\mathcal{G}$. Thus, if the inclusion $\mathcal{B} \subseteq \mathcal{A}^d$ is not satisfied, we can obtain an element in $\mathcal{G} \setminus \mathcal{A}$ and halt.

Step 2. Step 2 is similar to Step 1. Check whether $\mathcal{A} \subseteq \mathcal{B}^d$. If $\mathcal{A}$ contains a nonminimal transversal to $\mathcal{B}$, find a new element in $\mathcal{G}^d \setminus \mathcal{B}$ and halt.

Step 3. Suppose that $\mathcal{B} \subseteq \mathcal{A}^d$ and $\mathcal{A} \subseteq \mathcal{B}^d$. Then $\mathcal{B} = \mathcal{A}^d \Rightarrow (\mathcal{A}, \mathcal{B}) = (\mathcal{G}, \mathcal{G}^d)$. (This is because any hyperedge $X \in \mathcal{G} \setminus \mathcal{A}$ would be a transversal for $\mathcal{B}$, which would then imply that X contains some hyperedge of $\mathcal{A} = \mathcal{B}^d$, a contradiction. By symmetry, the duality of $\mathcal{A}$ and $\mathcal{B}$ also implies the emptiness of $\mathcal{G}^d \setminus \mathcal{B}$.) Hence $(\mathcal{A}, \mathcal{B}) = (\mathcal{G}, \mathcal{G}^d) \Leftrightarrow \mathcal{B} = \mathcal{A}^d$. The condition $\mathcal{B} = \mathcal{A}^d$ can be checked by solving the dualization problem for $\mathcal{A}$ and $\mathcal{B}$. If $\mathcal{B} \neq \mathcal{A}^d$, we obtain a new minimal transversal $X \in \mathcal{A}^d \setminus \mathcal{B}$; see section 1.1. By definition, X contains no hyperedge in $\mathcal{B}$ and X^c contains no hyperedge in $\mathcal{A}$. Due to the duality of $\mathcal{G}$ and $\mathcal{G}^d$, either (i) X^c contains a hyperedge of $\mathcal{G}$ or (ii) X contains a hyperedge of $\mathcal{G}^d$ but not both. We can call the superset oracle to decide which of the two cases holds. In case (i) we obtain a new hyperedge in $\mathcal{G} \setminus \mathcal{A}$ by querying the superset oracle at most $|X^c|$ times. Similarly, in case (ii) we get a new hyperedge in $\mathcal{G}^d \setminus \mathcal{B}$ in at most $|X|$ calls to the oracle.

Algorithm $\mathcal{J}$ readily implies the following result.

PROPOSITION 2.1 (see [5, 18]). *Problem GEN$(\mathcal{G}, \mathcal{G}^d)$ can be solved in $n\,(poly(|\mathcal{A}|, |\mathcal{B}|) + T_s(|\mathfrak{D}|)) + T_{dual}$ time, where T_{dual} denotes the time required for solving the dualization problem with $\mathcal{A}$ and $\mathcal{B}$.*

In particular, for any (quasi-)polynomial-time oracle $\mathfrak{D}$, problem GEN$(\mathcal{G}, \mathcal{G}^d)$ can be solved in quasi-polynomial time. Thus, for each of the five examples above we can jointly generate all the hyperedges of $(\mathcal{G}, \mathcal{G}^d)$ in incremental quasi-polynomial time. Note, however, that separately generating all the hyperedges of $\mathcal{G}$ or all the hyperedges of $\mathcal{G}^d$ may be substantially harder. For instance, as shown in [18], both problems GEN$(\mathcal{G})$ and GEN$(\mathcal{G}^d)$ are NP-hard for examples 3–5 above. In fact, in example 3 these problems are already NP-hard for $\vee, \wedge$-formulae of depth 3; if the depth is 2, then the formula is either a CNF or a DNF, and we get exactly dualization.

2.3. Dual-bounded hypergraphs. Algorithm $\mathcal{J}$ may not be efficient for solving either of the problems GEN$(\mathcal{G})$ or GEN$(\mathcal{G}^d)$ separately for the simple reason that we do not control which of the families $\mathcal{G} \setminus \mathcal{A}$ and $\mathcal{G}^d \setminus \mathcal{B}$ contains each new hyperedge produced by the algorithm. Suppose we want to generate $\mathcal{G}$ and the family $\mathcal{G}^d$ is exponentially larger than $\mathcal{G}$. Then, if we are unlucky, we can get hyperedges of $\mathcal{G}$ with exponential delay while getting large subfamilies of $\mathcal{G}^d$ (which are not needed at all) in between.

Such a problem will not arise and simultaneous generation of $(\mathcal{G}, \mathcal{G}^d)$ can be used to produce $\mathcal{G}$ efficiently, in some sense, if the size of $\mathcal{G}^d$ is polynomially limited in the size of $\mathcal{G}$ and in the input size $|\mathfrak{D}|$, or more precisely, when there exists a polynomial p such that

$$|\mathcal{G}^d| \leq p(|V|, |\mathfrak{D}|, |\mathcal{G}|). \tag{2.1}$$

We call such Sperner hypergraphs $\mathcal{G}$ *dual-bounded*.

If $\mathcal{G}$ is dual-bounded, we can generate both $\mathcal{G}$ and $\mathcal{G}^d$ in $|\mathcal{G}^d| + |\mathcal{G}| \leq poly(|V|, |\mathfrak{D}|, |\mathcal{G}|)$ rounds of Algorithm $\mathcal{J}$, and hence obtain all the hyperedges of $\mathcal{G}$ in *total quasi-polynomial* time.

This approach, however, may still be inefficient incrementally, i.e., for obtaining a single hyperedge of $\mathcal{G}$ as required in problem GEN$(\mathcal{G})$. It is easy to see that the decision problem "given a family $\mathcal{A} \subseteq \mathcal{G}$, determine whether $\mathcal{A} = \mathcal{G}$" is polynomially reducible

to dualization for any dual-bounded hypergraphs represented by a polynomial-time superset oracle. If $\mathcal{A}$ is much smaller than $\mathcal{G}$, however, getting a new hyperedge in $\mathcal{G} \setminus \mathcal{A}$ may require exponentially many (in $|\mathcal{A}|$) rounds of $\mathcal{J}$.

2.4. Uniformly dual-bounded hypergraphs. Let us call a Sperner hypergraph $\mathcal{G}$ *uniformly dual-bounded* if

$$|\mathcal{H}^d \cap \mathcal{G}^d| \leq p(|V|, |\mathfrak{D}|, |\mathcal{H}|) \text{ for any hypergraph } \mathcal{H} \subseteq \mathcal{G}.$$

Note that for $\mathcal{H} = \mathcal{G}$ the above condition gives (2.1).

PROPOSITION 2.2. *Problem GEN($\mathcal{G}$) is polytime reducible to dualization for any uniformly dual-bounded hypergraph $\mathcal{G}$ defined by a polynomial-time superset oracle.*

Proof. Given a proper subfamily $\mathcal{A}$ of $\mathcal{G}$, we wish to find a new hyperedge $\mathcal{G} \setminus \mathcal{A}$. Start with the given $\mathcal{A}$ and $\mathcal{B} = \emptyset$ and run Algorithm $\mathcal{J}$ repeatedly until it outputs a required hyperedge. Suppose this takes t rounds of $\mathcal{J}$; then $\mathcal{J}$ produces $t-1$ hyperedges of $\mathcal{G}^d$, each of which is also a hyperedge of $\mathcal{A}^d$; see Step 1 of the algorithm. Since $\mathcal{G}$ is uniformly dual-bounded, we have $t-1 \leq |\mathcal{A}^d \cap \mathcal{G}^d| \leq p(|V|, |\mathfrak{D}|, |\mathcal{A}|)$. ∎

Theorems 1.3 and 1.4 state that the hypergraphs $\mathcal{F}_{A,b}$ and $\mathcal{A}^{d_k}$ are both uniformly dual-bounded. In view of Proposition 2.2, this means that Theorems 1.3 and 1.4 imply Theorems 1.1 and 1.2, respectively.

3. Minimal solutions of monotone 0-1 inequalities. In this section, we prove Lemma 1.5 and Theorem 1.3.

3.1. Proof of Lemma 1.5. Suppose that some of the coefficients $w_1, \ldots, w_n$ are negative, say, $w_1 < 0, \ldots, w_r < 0$ and $w_{r+1} \geq 0, \ldots, w_n \geq 0$. Since $wx = w_1x_1 + \ldots + w_nx_n \geq t$ for any true point of $h(x)$ and $h(x)$ is monotone, each true point of $h(x)$ satisfies $w'x \geq t'$, where $w' = (0, \ldots, 0, w_{r+1}, \ldots, w_n) \geq 0$ and $t' = t - (w_1 + \cdots + w_r)$. Since $wx < t \Rightarrow w'x < t'$ for any binary x, it suffices to prove the lemma for w' and t'. Hence we may assume without loss of generality that all the n components of w are nonnegative. Now we prove the lemma by induction on n with the trivial base $n = 1$.

For $i \in \{1, 2, \ldots, n\}$, let $h_i = h|_{x_i=0}$ denote the restriction of h on the hyperplane $\{x \mid x_i = 0\}$. We split the proof into two cases.

Case 1. Suppose that for some variable, say, x_1, we have $h_1(x_2, x_3, \ldots, x_n) \equiv 0$, i.e., $h = x_1 q(y)$, where $y = (x_2, x_3, \ldots, x_n)$ and $q(y)$ is some monotone Boolean function. Then $\min T(h) = \{(1, y) \mid y \in \min T(q)\}$; consequently, $|\min T(h)| = |\min T(q)|$ and

$$\sum_{x \in \min T(h)} ex = |\min T(q)| + \sum_{y \in \min T(q)} ey. \tag{3.1}$$

Similarly, we have

$$\max F(h) = \{(1, y) \mid y \in \max F(q)\} \cup \{(0, e)\},$$

which implies

$$|\max F(h) \cap \{x \mid wx < t\}| \leq 1 + |\max F(q) \cap \{y \mid w'y < t - w_1\}|,$$

where $w' = (w_2, \ldots, w_n)$. Since $q \not\equiv 0$, we have

$$|\max F(q) \cap \{y \mid w'y < t - w_1\}| \leq \sum_{y \in \min T(q)} ey$$

by the induction hypothesis. Hence

$$|\max F(h) \cap \{x \mid wx < t\}| \leq 1 + \sum_{y \in \min T(q)} ey.$$

Therefore,

$$1 + \sum_{y \in \min T(q)} ey \leq |\min T(q)| + \sum_{y \in \min T(q)} ey,$$

which in view of (3.1) completes the inductive proof for Case 1.

Case 2. Let us now assume that $h_i \not\equiv 0$ for all $i = 1, \ldots, n$. It is easy to see that

$$\min T(h) \cap \{x \mid x_i = 0\} = \{(0, y) \mid y \in \min T(h_i)\}, \tag{3.2}$$

$$\max F(h) \cap \{x \mid x_i = 0\} \subseteq \{(0, y) \mid y \in \max F(h_i)\}. \tag{3.3}$$

Since $h_i \not\equiv 0$, we can apply the inductive hypothesis to each h_i to obtain the inequalities

$$|\{(0, y) \mid y \in \max F(h_i)\} \cap \{x \mid wx < t\}| \leq \sum_{x \in \min T(h_i)} ex,$$

from which by (3.3) we get

$$|\max F(h) \cap \{x \mid x_i = 0, wx < t\}| \leq \sum_{x \in \min T(h_i)} ex$$

for $i = 1, \ldots, n$. By multiplying the above inequalities by the nonnegative weights w_i and summing up the resulting bounds for all i, we obtain

$$\sum_{\substack{x \in \max F(h) \\ x_i = 0, wx < t}} w_i \leq \sum_{i=1}^{n} w_i \sum_{x \in \min T(h_i)} ex. \tag{3.4}$$

We can rewrite the left-hand side as

$$\sum_{\substack{x \in \max F(h) \\ x_i = 0, wx < t}} w_i = \sum_{\substack{x \in \max F(h) \\ wx < t}} \sum_{i : x_i = 0} w_i = \sum_{x \in \max F(h) \cap \{x \mid wx < t\}} w\bar{x},$$

where $\bar{x}$ is the complement of x, i.e., $\bar{x}_i = 1 - x_i$ for all i. Similarly, by using (3.2) on the right-hand side of (3.4) we can get

$$\sum_{i=1}^{n} w_i \sum_{x \in \min T(h_i)} ex = \sum_{x \in \min T(h)} ex \sum_{i : x_i = 0} w_i = \sum_{x \in \min T(h)} w\bar{x} \cdot ex.$$

Thus, (3.4) can be written equivalently as

$$\sum_{x \in \max F(h) \cap \{x \mid wx < t\}} w\bar{x} \leq \sum_{x \in \min T(h)} w\bar{x} \cdot ex.$$

Let $\omega = w_1 + \cdots + w_n$; then $w\bar{x} = \omega - wx \leq \omega - t$ for all $x \in \min T(h)$. In addition, we have $\omega - t < w\bar{x}$ for any point in the open half-space $\{x \mid wx < t\}$. Hence

$$(\omega - t)|\max F(h) \cap \{x \mid wx < t\}| \leq (\omega - t) \sum_{x \in \min T(h)} ex.$$

If $\omega - t \leq 0$, then $\omega = t$ (since $h \neq 0$) and thus $\min T(h)$ contains exactly one point from which the lemma follows. Otherwise we can cancel out the multiplicative factor $\omega - t$ and complete the proof. □

3.2. Proof of Theorem 1.3. To show (1.2) for a hypergraph $\mathcal{H} \subseteq \mathcal{F}_{A,b}$, define the monotone Boolean function $h(x)$ by the condition

$$h(x) = 1 \quad \Leftrightarrow \quad x \geq char(H) \text{ for some } \ H \in \mathcal{H},$$

where $char(H)$ is the characteristic vector of H. Note that since $\mathcal{H} \subseteq \mathcal{F}_{A,b}$ and the characteristic vector of any hyperedge of $\mathcal{F}_{A,b}$ satisfies the monotone system (1.1), each true point of $h(x)$ satisfies (1.1), i.e.,

$$h(x) = 1 \ \text{ and } \ x \in \{0,1\}^n \quad \Rightarrow \quad Ax \geq b. \tag{3.5}$$

Also note that since $\mathcal{H} \subseteq \mathcal{F}_{A,b}$ is Sperner, we have

$$|\min T(h)| = |\mathcal{H}|. \tag{3.6}$$

In addition, if $X \in \mathcal{H}^d$, then X^c contains no hyperedge of $\mathcal{H}$ and hence $char(X^c)$ is a false point of h. It is easily seen that $char(X^c)$ is a *maximal* false point of h, for otherwise X would not be a minimal transversal for $\mathcal{H}$. Thus,

$$char(X^c) \in \max F(h) \text{ for any } \ X \in \mathcal{H}^d. \tag{3.7}$$

By the definition of $\mathcal{F}^d_{A,b}$, the inclusion $X \in \mathcal{F}^d_{Ab}$ implies that $char(X^c)$ is infeasible for (1.1). In view of (3.7), this gives

$$|\mathcal{H}^d \cap \mathcal{F}^d_{A,b}| \leq \sum_{i=1}^{m} |\max F(h) \cap \{x \mid a_i x < b_i\}|,$$

where $a_i x \geq b_i$ is the ith inequality of (1.1). By (3.5) we can apply Lemma 1.5 to the monotone Boolean function h to obtain

$$|\max F(h) \cap \{x \mid a_i x < b_i\}| \leq n |\min T(h)|.$$

Now (1.2) follows from (3.6). □

4. Partial transversals. In this section, we prove Lemma 1.6 and Theorem 1.4.

4.1. Proof of Lemma 1.6. We shall prove the lemma by induction on k. If $k = 0$, then $\mathcal{T} = \{\emptyset\}$ by condition (1.4), thus implying that $\beta = 1$ and that the sets of $\mathcal{S}$ are pairwise disjoint. Hence $\alpha \leq m = (m-k)\beta$ follows.

In a general step, let us define subfamilies $\mathcal{S}_v = \{S \setminus \{v\} \mid S \in \mathcal{S},\ v \in S\}$ and $\mathcal{T}_v = \{T \setminus \{v\} \mid T \in \mathcal{T},\ v \in T\}$ for each $v \in U$. Let us further introduce the notations $\alpha_v = |\mathcal{S}_v|$, and $\beta_v = |\mathcal{T}_v|$.

For vertices $v \in U$ for which $\alpha_v \geq 2$ (and thus $\beta_v \geq 1$) the families $\mathcal{S}_v$ and $\mathcal{T}_v$ satisfy all the assumptions of the lemma with $m' = m - 1$ and $k' = k - 1$, and hence

$$\alpha_v \leq (m' - k')\beta_v = (m-k)\beta_v \tag{4.1}$$

follows by the inductive hypothesis. Then let us consider the partition $U = U_1 \cup U_2$, where $U_1 = \{v \in U \mid \alpha_v \leq 1\}$ and $U_2 = \{v \in U \mid \alpha_v \geq 2\}$. Summing up the inequalities (4.1) for all $v \in U_2$, we obtain

$$\sum_{v \in U_2} \alpha_v \leq (m-k) \sum_{v \in U_2} \beta_v. \tag{4.2}$$

On the left-hand side, using the thresholdness conditions (1.4) and the definition of α_v we obtain

$$\begin{aligned}\alpha(k+1)-|U_1| &\le \sum_{S\in\mathcal{S}} |S|-|U_1| \\ &\le \sum_{S\in\mathcal{S}} (|S|-|S\cap U_1|) \\ &= \sum_{S\in\mathcal{S}} |S\cap U_2| \\ &= \sum_{v\in U_2} \alpha_v,\end{aligned} \tag{4.3}$$

where the first inequality follows by $|S|\ge k+1$ for $S\in\mathcal{S}$, the second one is implied by $|U_1|\ge\sum_{S\in\mathcal{S}}|S\cap U_1|$ following from the definition of U_1, while the equations follow from the definitions of α_v, U_1, and U_2.

On the right-hand side of (4.1) we can write

$$\begin{aligned}\sum_{v\in U_2}\beta_v &= \sum_{T\in\mathcal{T}} |T\cap U_2| \\ &\le \sum_{T\in\mathcal{T}} |T| \\ &\le \beta k,\end{aligned} \tag{4.4}$$

where the first equality follows by the definition of β_v and $\mathcal{T}_v$ and the last inequality follows by the conditions $|T|\le k$ for $T\in\mathcal{T}$.

Putting together (4.2), (4.3), and (4.4) we obtain $(k+1)\alpha-|U_1|\le(m-k)k\beta$, or equivalently that

$$\alpha \le \frac{|U_1|}{k+1}+\frac{k}{k+1}(m-k)\beta. \tag{4.5}$$

If $|U_1|\le m-k$, then

$$\frac{|U_1|}{k+1}+\frac{k}{k+1}(m-k)\beta \le (m-k)\beta,$$

and hence $\alpha\le(m-k)\beta$ by (4.5). On the other hand, if $|U_1|>m-k$, then for each set $S\in\mathcal{S}$ we have $|S\cap U_1|\ge|S|-|U_2|\ge(k+1)-|U_2|>1$. Now by the definition of the set U_1 we obtain $\alpha\le|U_1|/(k+1-|U_2|)=(m-|U_2|)/(k+1-|U_2|)\le m-k\le(m-k)\beta$. ∎

4.2. Exponentially large Sperner families with few maximal intersections. Let s be a positive integer, and let U be a finite set of $m=2+3s$ elements, consisting of two special vertices a,b and s disjoint sets $U_1,\dots,U_s$ of size 3 each. Let us consider the hypergraph $\mathcal{S}$ with $\alpha=2s+3^s$ hyperedges of the following three types:

(a) $U\setminus(\{a\}\cup U_i)$, $i=1,\dots,s$,
(b) $U\setminus(\{b\}\cup U_i)$, $i=1,\dots,s$,.
(c) 3^s hyperedges obtained by selecting exactly one vertex from each U_i.

It is easy to see that $\mathcal{S}^\cap$ consists of exactly $s+2\binom{s}{2}=s^2$ maximal intersections, namely, the sets $U\setminus(\{a\}\cup\{b\}\cup U_i)$, for $i=1,\dots,s$, and $U\setminus(\{a\}\cup U_i\cup U_j)$ and $U\setminus(\{b\}\cup U_i\cup V_j)$ for $1\le i<j\le s$.

As another example, let $m = 5s$ and let U contain two additional vertices a_i, b_i for every 3-element set U_i. Consider the hypergraph $\mathcal{S}'$ defined by the 3^s hyperedges of the form as in (c) above and by $2s$ additional edges of the form $\{a_i\} \cup (W \setminus U_i)$ and $\{b_i\} \cup (W \setminus U_i)$, where $W = \cup_{j=1}^{s} U_j$ and $i = 1, \dots, s$. Then $\mathcal{S}'$ has exactly s maximal intersections, namely, the sets $W \setminus U_i$ for $i = 1, 2, \dots, s$.

4.3. Proof of Theorem 1.4. Given a hypergraph $\mathcal{A} \subseteq 2^V$ of m hyperedges on an n-element vertex set V, let $\phi : 2^V \to 2^{\{1,\dots,m\}}$ be the monotone mapping which assigns to a vertex set $X \subseteq V$ the subset $\phi(X) \subseteq U \doteq \{1, 2, \dots, m\}$ of indices of all those hyperedges of $\mathcal{A}$ which are contained in X, i.e., $\phi(X) = \{i \mid A_i \subseteq X,\ 1 \le i \le m\}$. Note that for any two sets $X, Y \subseteq V$ we have the identity

$$\phi(X) \cap \phi(Y) \equiv \phi(X \cap Y). \tag{4.6}$$

Let $\mathcal{H} = \{H_1, \dots, H_\beta\} \subseteq 2^V$ be an arbitrary nonempty collection of k-transversals for $\mathcal{A}$. To show that $\mathcal{H}$ satisfies inequality (1.3), consider the (multi-)hypergraph $\mathcal{T} = \{\phi(H_1^c), \dots, \phi(H_\beta^c)\}$. Since each $H_l \in \mathcal{H}$ is a k-transversal to $\mathcal{A}$, the complement of H_l contains at most k hyperedges of $\mathcal{A}$. Hence $|\phi(H_l^c)| \le k$, i.e., the size of each hyperedge of $\mathcal{T}$ is at most k.

Let the hypergraph $\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}$ consist of α hyperedges, say, $\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}} = \{X_1, \dots, X_\alpha\} \subseteq 2^V$. Consider the (multi-)hypergraph $\mathcal{S} = \{\phi(X_1), \dots, \phi(X_\alpha)\}$. Since $X_1, \dots, X_\alpha$ are $(k+1)$-unions, each hyperedge of $\mathcal{S}$ has size at least $k+1$.

Now let us show that for any two distinct indices $1 \le i < j \le \alpha$, the intersection of the hyperedges $\phi(X_i)$ and $\phi(X_j)$ is contained in a hyperedge of $\mathcal{T}$. In view of (4.6) we have to show that $\phi(X_i \cap X_j)$ is contained in some hyperedge of $\mathcal{T}$. To this end, observe that $\mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}$ is a Sperner hypergraph, and hence $X_i \cap X_j$ is a proper subset of X_i. However, $X_i \in \mathcal{H}^d \cap \mathcal{A}^{u_{k+1}}$ is a minimal transversal to $\mathcal{H}$. For this reason, $X_i \cap X_j$ misses a hyperedge of $\mathcal{H}$, say, H_l. Equivalently, $X_i \cap X_j \subseteq H_l^c$ which implies $\phi(X_i \cap X_j) \subseteq \phi(H_l^c) \in \mathcal{T}$. Now inequality (1.3) and Theorem 1.4 readily follow from Lemma 1.6. □

4.4. Generalized intersection inequalities. In this section we list some generalizations of the intersection inequality (1.6).

A hypergraph $\mathcal{S}$ on a base set V of size $m = |U|$ is called *l-covering* if every l points of U belong to at least two hyperedges of $\mathcal{S}$ or, in other words, to a maximal intersection in $\mathcal{S}^\cap$. Let $\mathcal{S}$ be an l-covering hypergraph satisfying the following thresholdness conditions: $|S| \ge k+1$ for all $S \in \mathcal{S}$ and $|S' \cap S''| \le k$ for any pair of distinct hyperedges $S', S'' \in \mathcal{S}$. Then the following inequality can be shown to hold for any $l \le k$:

$$|\mathcal{S}| \le |\mathcal{S}^\cap|(m-k)[1 - l/(k+1)].$$

Since every hypergraph is 0-covering, this is always at least as strong as (1.6).

Next let us assume that $|S| \ge k+1$ for all $S \in \mathcal{S}$, while $|T| \le k'$ for all $T \in \mathcal{S}^\cap$ for some integers $0 \le k' \le k < m$. Then

$$|\mathcal{S}| \le |\mathcal{S}^\cap|(m-k')/(k-k'+1).$$

In the case $k = k'$ we get the inequality (1.6) back again.

Finally, instead of pairwise intersections, let us consider r-wise intersections. Let $\mathcal{S}_r^\cap$ denote the hypergraph of all maximal intersections of r distinct edges of a (multi-) hypergraph $\mathcal{S}$. Then for $r \ge 2$ we can show that

$$|\mathcal{S}| \le |\mathcal{S}_r^\cap|(m-k)(r-1),$$

assuming, as before, that each hyperedge of $\mathcal{S}$ contains at least $k+1$ vertices and that every r-wise intersection has size at most k. Again for $r = 2$ this gives (1.6).

5. Related set-families. The notion of frequent sets appears in the data mining literature (see [1, 27]) and can be related naturally to the families considered above. More precisely, following a definition of [34], given a $(0,1)$-matrix and a threshold k, a subset of the columns is called *frequent* if there are at least k rows having a 1 entry in each of the corresponding positions. The problems of generating all *maximal frequent* sets and their duals, the so-called *minimal infrequent* sets (for a given binary matrix), were proposed and the complexity of the corresponding decision problems were asked in [34]. Results of [25] imply that it is NP-hard to determine whether a family of maximal frequent sets is incomplete, while our results prove that generating all minimal infrequent sets polynomially reduces to dualization.

Since the family $\mathcal{A}^{d_k}$ consists of all the minimal k-transversals to $\mathcal{A}$, that is, subsets of V which are disjoint from at most k hyperedges of $\mathcal{A}$, the family $\mathcal{A}^{cd_k}$ consists of all the minimal subsets of V which are contained in at most k hyperedges of $\mathcal{A}$. It is easy to recognize that these are the *minimal infrequent sets* in a matrix, the rows of which are the characteristic vectors of the hyperedges of $\mathcal{A}$. Furthermore, the family $\mathcal{A}^{d_k c}$ consists of all the maximal subsets of V, which are supersets of at most k hyperedges of $\mathcal{A}$.

Due to our results above, all these families can be generated in incremental quasi-polynomial time.

In the special case, if $\mathcal{A}$ is a quadratic set-family, i.e., if all hyperedges of $\mathcal{A}$ are of size 2, the family $\mathcal{A}$ can also be interpreted as the edge set of a graph G on vertex set V. Then, $\mathcal{A}^{d_k c}$ is also known as the family of the so-called *fairly independent sets* of the graph G consisting of all the vertex subsets which induce at most k edges (see [34].)

As it was defined above, the family $\mathcal{A}^{u_k}$ consists of all the minimal k-unions of $\mathcal{A}$, i.e., all minimal subsets of V which contain at least k hyperedges of $\mathcal{A}$, and hence the family $\mathcal{A}^{cu_k}$ consists of all the minimal subsets which contain at least k hyperedges of $\mathcal{A}^c$. Thus, the family $\mathcal{A}^{cu_k c}$ consists of all the maximal k-intersections, that is, maximal subsets of V which are subsets of at least k hyperedges of $\mathcal{A}$. These sets can be recognized as the *maximal frequent sets* in a matrix, the rows of which are the characteristic vectors of the hyperedges of $\mathcal{A}$. Finally, the family $\mathcal{A}^{u_k c}$ consists of all the maximal subsets of V which are disjoint from at least k hyperedges of $\mathcal{A}$.

As it follows from the mentioned results (see, e.g., [25]), generating all hyperedges for each of these families is NP-hard unless k (or $|\mathcal{A}| - k$) is bounded by a constant.

Let us add finally that since $|\mathcal{A}^c| = |\mathcal{A}|$, $|\mathcal{A}^{cu_{k+1}c}| = |\mathcal{A}^{u_{k+1}}|$, and $|\mathcal{A}^{d_k c}| = |\mathcal{A}^{d_k}|$ hold, Theorem 1.4 implies that for any binary matrix A with m rows and for any threshold value k $(0 \le k < m)$, the number of maximal frequent column sets of A is at most $(m-k)$ times the number of minimal infrequent column sets of A.

6. General closing remarks. In this paper we considered the problems of generating all partial and all multiple transversals. Both problems are formally more general than dualization, but in fact both are polynomially equivalent to it because the corresponding pairs of hypergraphs are uniformly dual-bounded. We close with the following negative result: *Generating all the minimal partial binary solutions to a system of inequalities $Ax \ge b$ is NP-hard even if A is binary and $b = (2, 2, \dots, 2)$.* To show this we can use arguments analogous to those of [24, 25]. Consider the well-known NP-hard problem of determining whether a given graph $G = (V, E)$ contains an independent vertex set of size t, where $t \ge 2$ is a given threshold; see [15]. Introduce

$|V|+1$ binary variables x_0 and $x_v,\ v \in V$, and write t inequalities $x_u + x_v \geq 2$ for each edge $e = (u, v) \in E$ followed by the inequalities $x_0 + x_v \geq 2,\ v \in V$. It is easily seen that the characteristic vector of any edge $e = (u, v)$ is a minimal binary solution satisfying at least t inequalities of the resulting system. Deciding whether there are other minimal binary solutions satisfying at least t inequalities of the system is equivalent to determining whether G has an independent set of size t.

Acknowledgment. The authors are thankful to József Beck for helpful discussions.

REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo, *Fast discovery of association rules*, in Advances in Knowledge Discovery and Data Mining, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, eds., AAAI Press, Menlo Park, California, 1996, pp. 307–328.

[2] M. Anthony and N. Biggs, *Computational Learning Theory. An Introduction,* Cambridge University Press, Cambridge, UK, 1992.

[3] L. Babai and P. Frankl, *Linear Algebra Methods in Combinatorics*, manuscript, University of Chicago, Chicago, IL, 1992.

[4] P. Bertolazzi and A. Sassano, *An $O(mn)$ time algorithm for regular set-covering problems*, Theoret. Comput. Sci., 54 (1987), pp. 237–247.

[5] J. C. Bioch and T. Ibaraki, *Complexity of identification and dualization of positive Boolean functions*, Inform. Comput., 123 (1995), pp. 50–63.

[6] E. Boros, V. Gurvich, and P. L. Hammer, *Dual subimplicants of positive Boolean functions*, Optim. Methods Softw., 10 (1998), pp. 147-156.

[7] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino, *Generating partial and multiple transversals of a hypergraph*, in Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP 2000), U. Montanari, J.D.P. Rolim, and E. Welzl, eds., Lecture Notes in Comput. Sci. 1853, Springer-Verlag, Berlin, Heidelberg, New York, 2000, pp. 588-599; also available online from http://dimacs.rutgers.edu/TechnicalReports/1999.html.

[8] E. Boros, K. Elbassioni, V. Gurvich, and L. Khachiyan, *An incremental RNC algorithm for generating all maximal independent sets in hypergraphs of bounded dimension*, Parallel Process. Lett., to appear; also available online from http://dimacs.rutgers.edu/TechnicalReports/2000.html.

[9] E. Boros, P. L. Hammer, T. Ibaraki, and K. Kawakami, *Polynomial-time recognition of 2-monotonic positive Boolean functions given by an oracle*, SIAM J. Comput., 26 (1997), pp. 93–109.

[10] C. J. Colbourn, *The Combinatorics of Network Reliability*, Oxford University Press, New York, 1987.

[11] Y. Crama, *Dualization of regular Boolean functions*, Discrete Appl. Math., 16 (1987), pp. 79–85.

[12] C. Domingo, N. Mishra, and L. Pitt, *Efficient read-restricted monotone CNF/DNF dualization by learning with membership queries*, Machine Learning, 37 (1999), pp. 89–110.

[13] T. Eiter and G. Gottlob, *Identifying the minimal transversals of a hypergraph and related problems*, SIAM J. Comput., 24 (1995), pp. 1278–1304.

[14] M. L. Fredman and L. Khachiyan, *On the complexity of dualization of monotone disjunctive normal forms*, J. Algorithms, 21 (1996), pp. 618–628.

[15] M. R. Garey and D. S. Johnson, *Computers and Intractability*, W. H. Freeman, San Francisco, 1979.

[16] V. Gurvich, *To the theory of multistep games*, U.S.S.R. Comput. Math. and Math. Phys., 13 (1973), pp. 1485–1500.

[17] V. Gurvich, *Nash-solvability of games in pure strategies*, U.S.S.R. Comput. Math. and Math. Phys., 15 (1975), pp. 357–371.

[18] V. Gurvich and L. Khachiyan, *On generating the irredundant conjunctive and disjunctive normal forms of monotone Boolean functions*, Discrete Appl. Math., 96-97 (1999), pp. 363–373.

[19] V. Gurvich and L. Khachiyan, *On the frequency of the most frequently occurring variable in dual DNFs*, Discrete Math., 169 (1997), pp. 245–248.

[20] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani, *Random generation of combinatorial structures from a uniform distribution*, Theoret. Comput. Sci., 43 (1986), pp. 169–188.
[21] D. S. Johnson, M. Yannakakis, and C. H. Papadimitriou, *On generating all maximal independent sets*, Inform. Process. Lett., 27 (1988), pp. 119–123.
[22] R. Karp and M. Luby, *Monte-Carlo algorithms for enumeration and reliability problems*, in Proceedings of the 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 56–64.
[23] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Generating all maximal independent sets: NP-hardness and polynomial-time algorithms*, SIAM J. Comput., 9 (1980), pp. 558–565.
[24] K. Makino and T. Ibaraki, *Interior and exterior functions of Boolean functions*, Discrete Appl. Math., 69 (1996), pp. 209–231.
[25] K. Makino and T. Ibaraki, *Inner-core and outer-core functions of partially defined Boolean functions*, Discrete Appl. Math., 96-97 (1999), pp. 307–326.
[26] K. Makino and T. Ibaraki, *A fast and simple algorithm for identifying* 2*-monotonic positive Boolean functions*, J. Algorithms, 26 (1998), pp. 291–305.
[27] H. Mannila and K. J. Räihä, *Design by example: An application of Armstrong relations*, J. Comput. System Sci., 22 (1986), pp. 126-141.
[28] B. Morris and A. Sinclair, *Random walks on truncated cubes and sampling* 0-1 *knapsack problem*, in Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, 1999, pp. 230–240.
[29] U. N. Peled and B. Simeone, *Polynomial-time algorithm for regular set-covering and threshold synthesis*, Discrete Appl. Math., 12 (1985), pp. 57–69.
[30] U. N. Peled and B. Simeone, *An* $O(nm)$*-time algorithm for computing the dual of a regular Boolean function*, Discrete Appl. Math., 49 (1994), pp. 309–323.
[31] K. G. Ramamurthy, *Coherent Structures and Simple Games*, Kluwer Academic Publishers, Dordrecht, the Netherlands, 1990.
[32] R. C. Read, *Every one a winner, or how to avoid isomorphism when cataloging combinatorial configurations*, Ann. Discrete Math., 2 (1978), pp. 107–120.
[33] R. C. Read and R. E. Tarjan, *Bounds on backtrack algorithms for listing cycles, paths, and spanning trees*, Networks, 5 (1975), pp. 237–252.
[34] R. H. Sloan, K. Takata, and G. Turan, *On frequent sets of Boolean matrices*, Ann. Math. Artificial Intelligence, 24 (1998), pp. 193–209.
[35] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, *A new algorithm for generating all the maximal independent sets*, SIAM J. Comput., 6 (1977), pp. 505–517.
[36] J. D. Ullman, *Principles of Database and Knowledge Base Systems*, Vols. 1 and 2, Computer Science Press, Rockville, MD, 1988.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 2051–2068

A CONSTANT-FACTOR APPROXIMATION ALGORITHM FOR PACKET ROUTING AND BALANCING LOCAL VS. GLOBAL CRITERIA*

ARAVIND SRINIVASAN† AND CHUNG-PIAW TEO‡

Abstract. We present the first constant-factor approximation algorithm for a fundamental problem: the *store-and-forward packet routing problem on arbitrary networks*. Furthermore, the queue sizes required at the edges are bounded by an absolute constant. Thus, this algorithm balances a *global* criterion (routing time) with a *local* criterion (maximum queue size) and shows how to get simultaneous good bounds for both. For this particular problem, approximating the routing time well, even without considering the queue sizes, was open. We then consider a class of such local vs. global problems in the context of covering integer programs and show how to improve the local criterion by a logarithmic factor by losing a constant factor in the global criterion.

Key words. packet routing, approximation algorithms, linear programming, multicommodity flow, rounding theorems, randomized algorithms, covering integer programs, randomized rounding, discrete ham-sandwich theorems

AMS subject classifications. 68Q15, 05C65

PII. S0097539798335596

1. Introduction.

Recent research on approximation algorithms has focused a fair amount on *bicriteria* (or even multicriteria) minimization problems, attempting to simultaneously keep the values of two or more parameters "low" (see, e.g., [11, 21, 22, 29, 30, 32]). One motivation for this is that real-world problems often require such balancing. In this work, we consider a family of bicriteria problems that involve balancing a *local capacity* constraint (e.g., the maximum queue size at the links of a packet routing network, the maximum number of facilities per site in facility location) with a global criterion (resp., routing time, total cost of constructing the facilities). Since these global criteria are NP-hard to minimize even with no constraint on the local criterion, we shall seek good approximation algorithms.

1.1. Packet routing.

Our main result is a constant-factor approximation algorithm for store-and-forward packet routing, a fundamental routing problem in interconnection networks (see Leighton's book and survey [14, 15]); furthermore, the queue sizes will all be bounded by a constant. This packet routing problem has received considerable attention for more than 15 years and is as follows.

DEFINITION 1.1 (store-and-forward packet routing). *We are given an arbitrary N-node routing network (directed or undirected graph) G and a set $\{1, 2, \ldots, K\}$ of packets which are initially resident (respectively) at the (multi-)set of nodes $\{s_k : 1 \leq$*

*Received by the editors March 18, 1998; accepted for publication (in revised form) September 7, 2000; published electronically April 18, 2001. A preliminary version of this work appeared in *Proceedings of the ACM Symposium on Theory of Computing*, El Paso, TX, 1997, pp. 636–643.
http://www.siam.org/journals/sicomp/30-6/33559.html

†Bell Laboratories, Lucent Technologies, 600–700 Mountain Ave., Murray Hill, NJ 07974-0636 (srin@research.bell-labs.com). Part of this work was done while the author was at the School of Computing, National University of Singapore, Singapore 119260 and was supported in part by National University of Singapore Academic Research Fund grants RP950662, RP960620, and RP970607.

‡Department of Decision Sciences, National University of Singapore, Singapore 119260, Republic of Singapore (fbateocp@nus.edu.sg). This author was supported in part by National University of Singapore Academic Research Fund grant RP3970021 and a Fellowship from the Singapore-MIT Alliance Program in High-Performance Computation for Engineered Systems.

$k \leq K\}$ of G. Each packet k is a message that needs to be routed to some given destination node t_k in G. We have to route each packet k from s_k to t_k, subject to the following: (i) *each packet k must follow some path in G;* (ii) *each edge traversal takes one unit of time;* (iii) *no two packets can traverse the same edge at the same unit of time; and* (iv) *packets are only allowed to queue along the edges of G during the routing stage. There are no other constraints on the paths taken by the packets: these can be arbitrary paths in G. The NP-hard objective is to select a path for each packet and to coordinate the routing so that the elapsed time by which all packets have reached their destinations is minimized; i.e., we wish to keep this routing time as small as possible.*

Extensive research has been conducted on this problem: see [14, 15] and the references therein. The most desirable type of algorithm here would, in addition to keeping the routing time and queue sizes low, also be *distributed*: given a set of incoming packets and their (source, destination) values, any switch (node of G) decides what to do with them next, without any other knowledge of the (multi-)set $\{(s_k, t_k) : 1 \leq k \leq K\}$. This would be ideal for parallel computing. (Distributed algorithms in this context are also termed *on-line* algorithms in the literature.) Several such ingenious results are known for specific networks such as the mesh, butterfly, or hypercube. For instance, given any routing problem with N packets on an N-node butterfly, there is a randomized on-line routing algorithm that, with high probability, routes the packets in $O(\log N)$ time using $O(1)$-sized queues [28]. (We let e denote the base of the natural logarithm, and, for $x > 0$, $\lg x$, $\ln x$, and $\ln^+ x$, respectively, denote $\log_2 x$, $\log_e x$, and $\max\{\log_e x, 1\}$. Also, Z_+ will denote the set of nonnegative integers.)

Good on-line algorithms here, however, are not always feasible or required for the following reasons:

- A large body of research in routing is concerned with *fault-tolerance*: the possibility of G being a reasonable routing network when its nodes are subject to (e.g., random or worst-case) faults. See, e.g., Kaklamanis et al. [12], Leighton, Maggs, and Sitaraman [18], and Cole, Maggs, and Sitaraman [6]. In this case, we do not expect good on-line algorithms, since the fault-free subgraph $\hat{G}$ of G has an unpredictable structure. Indeed, a fair amount of research in this area, e.g., [6, 18], focuses on showing that $\hat{G}$ is still a reasonably good routing network under certain fault models, assuming global information about $\{(s_k, t_k)\}$ and the fault structure.
- Ingenious on-line algorithms for specific networks such as the butterfly in the fault-free case [28] are only *existentially* (near-)optimal. For instance, the $O(\lg N)$ routing time of [28] is existentially optimal to within a constant factor, since there are families of routing instances that require $\Theta(\lg N)$ time. However, the worst-case approximation ratio can be $\Theta(\lg N)$. It seems very hard (potentially impossible) to devise on-line algorithms that are near-optimal on *each instance*.
- The routing problem can be considered as a variant of unit-demand multicommodity flow, where all arc capacities are the same, queuing is allowed, and where delivery time is also a crucial criterion. (Algorithms for this problem that require just $O(1)$ queue sizes, such as ours, will also scale with network size.) For such flow problems, the routing problems often have to be run repeatedly. It is therefore reasonable to study *off-line* approximation algorithms, i.e., efficient algorithms that use the knowledge of the network and of

$\{(s_k, t_k)\}$ and have a good approximation ratio.

Furthermore, it seems like a difficult problem to construct on-line routing algorithms for *arbitrary* networks, even with, say, a polylogarithmic approximation guarantee. See Ostrovsky and Rabani [26] for good on-line packet scheduling algorithms, *given the path to be traversed for each packet.*

By combining some new ideas with certain powerful results of Leighton, Maggs, and Rao [16], Leighton, Maggs, and Richa [17], Karp et al. [13], and Lin and Vitter [20], we present the first polynomial-time off-line *constant-factor* approximation algorithm for the store-and-forward packet routing problem. Furthermore, the queue sizes of the edges are bounded by $O(1)$. No approximation algorithms with a sublogarithmic approximation guarantee were known for this problem, to the best of our knowledge. For instance, a result from the seminal work of Leighton and Rao [19] leads to routing algorithms that are *existentially* good. Their network embedding of G ensures that there is some routing instance on G for which their routing time is to within an $O(\lg N)$ factor of optimal, but no good worst-case performance guarantee is known. We may attempt randomized rounding on some suitable linear programming (LP) relaxation of the problem; however, apart from difficulties like controlling path lengths, it seems hard to get a constant-factor approximation using this approach for families of instances where the LP optimal value grows as $o(\lg(N+K))$. Our approach uses the rounding theorem of [13] to select the set of paths that will be used in the routing algorithm of [17]. The analysis involves an interesting trade-off between the "dilation" criterion (maximum path length) and the "congestion" criterion (maximum number of paths using any edge).

1.2. Covering integer programs. Let v^T denote the transpose of a (column) vector v. In the second part of the paper, we continue to address the problem of simultaneously obtaining good bounds on two criteria of a problem. We focus on the NP-hard family of *covering integer programs* (CIPs), which includes the well-known set cover problem. This class of problems exhibits features similar to our packet routing problem: the latter can be formulated as a covering problem with side packing constraints. In CIPs, the packing constraints are upper bound constraints on the variables.

DEFINITION 1.2 (CIPs). *Given $A \in [0,1]^{m\times n}$, $b \in [0,\infty)^m$, and $c \in [0,1]^n$, a CIP seeks to minimize $c^T \cdot x$ subject to $Ax \geq b$, $x \in Z_+^n$, and $0 \leq x_j \leq d_j$ for each j (the $d_j \in Z_+$ are given integers). If $A \in \{0,1\}^{m\times n}$, then we assume without loss of generality (w.l.o.g.) that each b_i is a positive integer. Define $B = \min_i b_i$; w.l.o.g., we may assume $B \geq 1$. A CIP is* uncapacitated *if for all j, $d_j = \infty$.*

It is well known that the two assumptions above are w.l.o.g. (i) If $A \in \{0,1\}^{m\times n}$, then we can clearly replace each b_i by $\lceil b_i \rceil$. (ii) Given a CIP with some $A_{i,j} > b_i$, we can *normalize* it by first setting $A_{i,j} := b_i$ for each such (i,j) and then scaling A and b uniformly so that for all k, ($b_k \geq 1$ and $\max_\ell A_{k,\ell} \leq 1$). This is easily seen to result in an equivalent CIP.

To motivate the model, we consider a concrete CIP example: a facility location problem that generalizes the set cover problem. Here, given a digraph G, we want to place facilities on the nodes suitably so that every node has at least B facilities in its out-neighborhood. Given a cost-per-facility c_j of placing facilities at node j, we desire to place the facilities in a way that will minimize the total cost. It is easy to see that this NP-hard problem is a CIP, with the matrix A having only zeroes and ones. This problem illustrates one main reason for the constraints $\{x_j \leq d_j\}$: for reasons of capacity, security, or fault-tolerance (not many facilities will be damaged if, for

instance, there is an accident/failure at a node), we may wish to upper bound the number of facilities that can be placed at individual sites. The more general problem of "file sharing" in a network has been studied by Naor and Roth [24], where again, the maximum load (number of facilities) per node is balanced with the global criterion of total construction cost. For similar reasons, CIPs typically include the constraints $\{x_j \leq d_j : 1 \leq j \leq n\}$. In fact, the case where $d_j = 1$ for all j is quite common.

Dobson [7] and Fisher and Wolsey [8] study a natural greedy algorithm $\mathcal{GA}$ for CIPs. For a given CIP, let OPT denote the value of its optimal *integral* solution. We define $\gamma_1 \doteq \min_{i,j}\{A_{i,j}/c_j : A_{i,j} \neq 0\}$ and $\gamma_2 \doteq \max_j(\sum_{i=1}^m A_{i,j}/c_j)$. Then, it is shown in [8] that $\mathcal{GA}$ produces a solution of value at most $OPT(1 + \ln(\gamma_2/\gamma_1))$. If each row of the linear system $Ax \geq b$ is scaled so that the minimum nonzero entry in the row is at least 1, it is shown in [7] that $\mathcal{GA}$'s output is at most $OPT(1 + \ln(\max_j \sum_{i=1}^n A_{i,j}))$.

Another well-known approach to CIPs is to start with their LP relaxation, wherein each x_j is allowed to be a *real* in the range $[0, d_j]$. Throughout, we shall let y^* denote the LP optimum of a given CIP. Clearly, y^* is a lower bound on OPT. Bertsimas and Vohra [5] conduct a detailed study of approximating CIPs and present an approximation algorithm which finds a feasible solution whose value is $O(y^* \lg m)$ [5]. Previous work of this paper's first author [31] presents an algorithm that computes an $x \in Z_+^n$ such that $Ax \geq b$ and

$$c^T \cdot x \leq a_0 y^* \cdot \max\{\ln^+(mB/y^*)/B, \sqrt{\ln^+(mB/y^*)/B}\} \tag{1.1}$$

for some absolute constant $a_0 > 0$.[1] The bound "$x_j \leq d_j$" may not hold for all j, but we will have for all j that

$$x_j \leq d_j \cdot \left(1 + a_1 \max\left\{\ln^+(mB/y^*)/B, \sqrt{\ln^+(mB/y^*)/B}\right\}\right) \tag{1.2}$$

for a certain absolute constant $a_1 > 0$. A related result is presented in [24] for file-sharing.

If B is "large" (greater than a certain threshold), then these results significantly improve previous results in the "global" criterion of keeping $c^T \cdot x$ small while compromising somewhat on the "local" capacity constraints $\{x_j \leq d_j\}$. This is a common approach in bicriteria approximation: losing a small amount in each criterion to keep the maximum such loss "low." In particular, if y^* grows at least as fast as $me^{-O(B)}$, then the output value here is $O(y^*)$, while maintaining $x_j = O(d_j)$ for all j. (Also, if the CIP is uncapacitated, then the above is a significant improvement if B is large.)

We see from (1.2) that in the case where $\ln^+(mB/y^*) \leq B$, both $(c^T \cdot x)/y^*$ and the maximum "violation" $\max_j\ x_j/d_j$ are bounded by constants, which is reasonable. Thus, we consider the case where $\ln^+(mB/y^*) > B$. Here, however, the violation $\max_j\ x_j/d_j$ can be as high as $1 + a_1 \ln^+(mB/y^*)/B$, which is unsatisfactory. If it is not feasible (e.g., for capacity/fault-tolerance reasons) to deviate from the local constraints by this much, then even the gain in the global criterion (caused by the large value of B) will not help justify such a result. Therefore, a natural question is: is it possible to lose a small amount in the global criterion, while losing much less in the local criterion (i.e., in $\max_j\ x_j/d_j$), in the case where $\ln^+(mB/y^*) > B$? We answer this in the affirmative.

[1] Recall that $\ln^+(x)$ denotes $\max\{\ln(x), 1\}$. To parse the term "$\ln^+(mB/y^*)/B$", note that it is $\ln(mB/y^*)/B$ if $y^* \leq me^{-B}$ and is $O(1)$ otherwise.

(a) For the important special case of *unweighted* CIPs (for all j, $c_j = 1$), consider the case $\ln^+(mB/y^*) > B$. Then, for any parameter ϵ, $0 < \epsilon < 1$, we present an algorithm that outputs an x with

(i) $x_j \leq \lceil d_j/(1-\epsilon) \rceil$ for all j, where
(ii) the objective function value is at most $a_2 y^*(1/(1-\epsilon)+(1/\epsilon^2)\ln^+(mB/y^*)/B)$ for an absolute constant $a_2 > 0$.

Note the significant improvement over (1.1) and (1.2), particularly if ϵ is a constant: by losing just a constant factor in the output value of the objective function, we have ensured that each x_j/d_j is bounded by a *constant* (at most $1/(1-\epsilon)+1/d_j$). This is an improvement over the bound stated in (1.2). In our view, ensuring little loss in the local criterion here is quite important as it involves *all* the variables x_j (e.g., all the nodes of a graph in facility location) and since $\max_j x_j/d_j$ may be *required* to be low due to physical and other constraints.

(b) For the case where the coefficient matrix A has only zeroes and ones and where a feasible solution (i.e., for all j, $x_j \leq d_j$) to a (possibly weighted) CIP is really required, we present an approximation algorithm with output value at most $O(y^* \ln^+(m/y^*))$. This works whether $\ln^+(mB/y^*) > B$ or not. While incomparable with the results of [7, 8], this is better if y^* is bigger than a certain threshold. This is also seen to be an improvement over the $O(y^* \lg m)$ bound of [5] if $y^* \geq m^a$, where $a \in (0, 1)$ is an absolute constant.

Thus, this work presents improved local vs. global balancing for a family of problems: the basic packet routing problem (the first constant-factor approximation) and CIPs (gaining more than a constant factor in the local criterion while losing a constant factor in the global criterion). The structure of the rest of the paper is as follows. In section 2, we discuss the algorithm for the packet routing problem, which consists mainly of three steps: (1) constructing and solving an LP relaxation (section 2.1); (2) obtaining a set of routes via suitable rounding (section 2.2); and (3) scheduling the packets (section 2.3) using the algorithm of [17]. The nature of our LP relaxation also provides an interesting re-interpretation of our result, as shown by Theorem 2.4 in section 2.3. We discuss in section 2.4 an extension of our idea to a more general setting, where the routing problem is replaced by a canonical covering problem. In section 3, we discuss our results for the general CIPs. We present our improved local vs. global balancing for unweighted CIPs in section 3.1; the case where $x_j \leq d_j$ is really required for all j is handled in section 3.2 for the case where the coefficient matrix has only zeroes and ones. (Note, for instance, that the coefficient matrix has only zeroes and ones for the facility location problem discussed in section 1.2.)

2. Approximating the routing time to within a constant factor. We refer the reader to the introduction for the definition and motivation for packet routing. Leighton, Maggs, and Rao, in a seminal paper, studied the issue of scheduling the movement of the packets *given the path to be traversed by each packet* [16]. They showed that the packets can be routed in time proportional to the sum of the *congestion* and *dilation* of the paths selected for each packet (see the sentence preceding section 1.2 for the definition of these two parameters). However, they did not address the issue of path selection; one motivation for their work is that paths can plausibly be selected using, e.g., the well-known "random intermediate destinations" idea [33, 34]. However, no general results on path selection, and hence on the time needed for packet routing, were known for *arbitrary* networks G. We address this issue here by studying the path selection problem.

THEOREM 2.1. *There are constants $c', c'' > 0$ such that the following holds. For any packet routing problem on any network, there is a set of paths and a corresponding schedule that can be constructed in polynomial time such that the routing time is at most c' times the optimal. Furthermore, the maximum queue size at each edge is bounded by c''.*

We shall denote any path from s_k to t_k as an (s_k, t_k)-path. Given a (directed) path P, $E(P)$ will denote its set of (directed) edges.

2.1. A linear programming relaxation. Consider any given packet routing problem. Let us consider any feasible solution for it, where packet k is routed on path P_k. Let D denote the *dilation* of the paths selected, i.e., D is the length of a longest path among the P_k. Clearly, the time to route all the packets is bounded below by D. Similarly, let C denote the congestion of the paths selected, i.e., the maximum number of packets that must traverse any single edge during the entire course of the routing. Clearly, C is also a lower bound on the time needed to route the packets.

Let N denote the number of nodes in the network and K the number of packets in the problem. We now present an LP relaxation for the problem; some of the notation used in this relaxation is explained in the following paragraph.

$$\text{(ROUTING)}\ \min(C+D)/2 \text{ subject to}$$

$$\sum_{k=1}^{K} x_f^k \leq C \quad \forall f \in E(G), \tag{2.1}$$

$$\sum_{f \in E(G)} x_f^k \leq D \quad \forall k \in \{1, 2, \ldots, K\}, \tag{2.2}$$

$$\mathcal{N}^k x^k = b^k \quad \forall k \in \{1, 2, \ldots, K\}, \tag{2.3}$$

$$0 \leq x_f^k \leq 1 \quad \forall k \in \{1, 2, \ldots, K\}\ \forall f \in E(G).$$

The vector x above is basically a "fractional flow" in G, where x_f^k denotes the amount of "flow" of packet k on edge $f \in E(G)$. The superscript k merely indexes a packet and *does not* mean a kth power. The constraints "$\mathcal{N}^k x^k = b^k$" model the requirement that for packet k, (i) a total of one unit of flow leaves s_k and reaches t_k, and (ii) at all other nodes, the net inflow of the flow corresponding to packet k, equals the net outflow of the flow corresponding to packet k. For conciseness, we have avoided explicitly writing out this (obvious) set of constraints above. Constraints (2.1) say that the "fractional congestion" on any edge f is at most C. Constraints (2.2) say that the "fractional dilation" $\sum_f x_f^k$ is at most D. This is a somewhat novel way of relaxing path lengths to their fractional counterparts.

It is easy to see that any path-selection scheme for the packets, i.e., any *integral* flow (where all the x_f^k are either 0 or 1) with congestion C and dilation D, satisfies the above system of inequalities. Thus, since C and D are both lower bounds on the length of the routing time for such a path-selection strategy, so is $(C+D)/2$. Hence, the optimum value of the LP is indeed a lower bound on the routing time for a given routing problem: it is indeed a relaxation. Note that the LP has polynomial size since it has "only" $O(Km)$ variables and $O(Km)$ constraints, where m denotes the number of edges in the network. Thus, it can be solved in polynomial time. Let $\{\overline{x}, \overline{C}, \overline{D}\}$ denote an optimal solution to the program. In section 2.2, we will conduct a certain type of "filtering" on $\overline{x}$. Section 2.3 will then construct a path for each packet and then invoke the algorithm of [17] for packet scheduling.

2.2. Path filtering. The main ideas now are to decompose $\overline{x}$ into a set of "flow paths" via the "flow decomposition" approach and then to adapt the ideas in Lin-Vitter [20] to "filter" the flow paths by effectively eliminating all flow paths of length more than $2\overline{D}$.

The reader is referred to section 3.5 of [1] for the well-known flow decomposition approach. This approach efficiently transforms $\overline{x}$ into a set of flow paths that satisfy the following conditions. For each packet k, we get a collection $\mathcal{Q}^k$ of flows along (s_k, t_k)-paths; each $\mathcal{Q}^k$ has at most m paths. Let $P_{k,i}$ denote the ith path in $\mathcal{Q}^k$. $P_{k,i}$ has an associated flow value $z_{k,i} \geq 0$ such that for each k, $\sum_i z_{k,i} = 1$. (In other words, the unit flow from s_k to t_k has been decomposed into a convex combination of (s_k, t_k)-paths.) The total flow on any edge f will be at most $\overline{C}$:

$$\sum_{(k,i):f \in E(P_{k,i})} z_{k,i} = \sum_{k=1}^{K} \overline{x}_f^k \leq \overline{C}; \tag{2.4}$$

the inequality in (2.4) follows from (2.1). Also, let $|P|$ denote the length of (i.e., the number of edges in) a path P. Importantly, the following bound will hold for each k:

$$\sum_i z_{k,i} |P_{k,i}| = \sum_{f \in E(G)} \overline{x}_f^k \leq \overline{D} \tag{2.5}$$

with the inequality following from (2.2).

The main idea now is to "filter" the flow paths so that only paths of length at most $2\overline{D}$ remain. For each k, define

$$g_k = \sum_{i:|P_{k,i}| > 2\overline{D}} z_{k,i}.$$

It is to easy to check via (2.5) that $g_k \leq 1/2$ for each k. Thus, suppose we define new flow values $\{y_{k,i}\}$ as follows for each k: $y_{k,i} = 0$ if $|P_{k,i}| > 2\overline{D}$, and $y_{k,i} = z_{k,i}/(1 - g_k)$ if $|P_{k,i}| \leq 2\overline{D}$. We still have the property that we have a convex combination of flow values: $\sum_i y_{k,i} = 1$. Also, since $g_k \leq 1/2$ for all k, we have $y_{k,i} \leq 2z_{k,i}$ for all k, i. Therefore, (2.4) implies that the total flow on any edge f is at most $2\overline{C}$:

$$\sum_{(k,i):f \in E(P_{k,i})} y_{k,i} \leq 2\overline{C}. \tag{2.6}$$

Most importantly, by setting $y_{k,i} = 0$ for all the "long" paths $P_{k,i}$ (those of length more than $2\overline{D}$), we have ensured that all the flow paths under consideration are of length at most $O(\overline{D})$. We denote the collection of flow paths for packet k by $\mathcal{P}^k$. For ease of exposition, we will also let y_P denote the flow value of any general flow path P.

Remarks. We now point out two other LP relaxations which can be analyzed similarly and which yield slightly better constants in the approximation guarantee.

- It is possible to directly bound path-lengths in the LP relaxation so that filtering need not be applied; one can show that this improves the approximation guarantee somewhat. On the other hand, such an approach leads to a somewhat more complicated relaxation, and furthermore, binary search has to be applied to get the "optimal" path-length. This, in turn, entails potentially $O(\lg N)$ calls to an LP solver, which increases the running time. Thus, there is a trade-off involved between the running time and the quality of approximation.

- In our LP formulation, we could have used a variable W to stand for $\max\{C, D\}$ in place of C and D; the problem would have been to minimize W subject to the fractional congestion and dilation being at most W. Since W is a lower bound on the optimal routing time, this is indeed a relaxation; using our approach with this formulation leads to a slightly better constant in the quality of our approximation. Nevertheless, we have used our approach to make the relationship between C and D explicit.

2.3. Path selection and routing. Note that $\{y_P : P \in \bigcup_{k=1}^{K} \mathcal{P}^k\}$ is a fractional feasible solution to the following set of inequalities:

$$\sum_k \sum_{P \in \mathcal{P}^k, (i,j) \in E(P)} y_P \leq 2\overline{C} \quad \forall f \in E(G),$$
$$\sum_{P \in \mathcal{P}^k} y_P = 1 \quad \forall k.$$

To select one path from $\mathcal{P}^k$ for each packet k, we need to modify the above fractional solution to an integral 0-1 solution. To ensure that the congestion does not increase by much, we shall use the following rounding algorithm of [13].

THEOREM 2.2 (see [13]). *Let A be a real valued $r \times s$ matrix and y be a real-valued s-vector. Let b be a real-valued vector such that $Ay = b$ and t be a positive real number such that, in* every column *of A,* (i) *the sum of all the positive entries is at most t* and (ii) *the sum of all the negative entries is at least $-t$. Then we can compute an integral vector $\overline{y}$ such that for every i, either $\overline{y}_i = \lfloor y_i \rfloor$ or $\overline{y}_i = \lceil y_i \rceil$ and $A\overline{y} = \overline{b}$ where $\overline{b}_i - b_i < t$ for all i. Furthermore, if y contains d nonzero components, the integral approximation can be obtained in time $O(r^3 \lg(1 + s/r) + r^3 + d^2 r + sr)$.*

To use Theorem 2.2, we first transform our linear system above to an equivalent system

$$\sum_k \sum_{P \in \mathcal{P}^k, (i,j) \in E(P)} y_P \leq 2\overline{C} \quad \forall\ (i,j) \in E(G),$$
$$\sum_{P \in \mathcal{P}^k} (-2\overline{D}) y_P = -2\overline{D} \quad \forall\ k.$$

The set of variables above is $\{y_P : P \in \bigcup_{k=1}^{K} \mathcal{P}^k\}$. Note that $y_P \in [0,1]$ for all these variables. Furthermore, in this linear system, the positive column sum is bounded by the maximum length of the paths in $\mathcal{P}^1 \cup \mathcal{P}^2 \cup \cdots \cup \mathcal{P}^K$. Since each path in any $\mathcal{P}^k$ is of length at most $2\overline{D}$ due to our filtering, each positive column sum is at most $2\overline{D}$. Each negative column sum is clearly $-2\overline{D}$. Thus, the parameter t for this linear system, in the notation of Theorem 2.2, can be taken to be $2\overline{D}$. Hence by Theorem 2.2, we can obtain in polynomial time an *integral* solution $\overline{y}$ satisfying

$$\sum_k \sum_{P \in \mathcal{P}^k, f \in E(P)} \overline{y}_P \leq 2\overline{C} + 2\overline{D} \quad \forall\ f \in E(G), \tag{2.7}$$

$$\sum_{P \in \mathcal{P}^k} (-2\overline{D}) \overline{y}_P < 0 \quad \forall\ k, \tag{2.8}$$

$$\overline{y}_P \in \{0,1\} \quad \forall\ P \in \bigcup_{k=1}^{K} \mathcal{P}^k. \tag{2.9}$$

For each packet k, by conditions (2.8) and (2.9), we have $\sum_{P \in \mathcal{P}^k} \overline{y}_P \geq 1$. (Note the crucial role of the *strict* inequality in (2.8).) Thus, for each packet k, we have selected at least one path from s_k to t_k with length at most $2\overline{D}$; furthermore, the congestion is bounded by $2\overline{C}+2\overline{D}$ (from (2.7)). If there are two or more such (s_k, t_k)-paths, we can arbitrarily choose one among them, which of course cannot increase the congestion. The next step is to schedule the packets, given the set of paths selected for each packet. To this end, we use the following result of [17], which provides an algorithm for the existential result of [16].

THEOREM 2.3 (see [17]). *For any set of packets with edge-simple paths having congestion c and dilation d, a routing schedule having length $O(c+d)$ and constant maximum queue size can be found in random polynomial time.*

Applying this theorem to the paths selected from the previous stage, which have congestion $c \leq 2\overline{C} + 2\overline{D}$ and dilation $d \leq 2\overline{D}$, we can route the packets in time $O(\overline{C} + \overline{D})$. Recall that $(\overline{C} + \overline{D})/2$ is a lower bound on the length of the optimal schedule. Thus, we have presented a constant-factor approximation algorithm for the off-line packet routing problem; furthermore, the queue sizes are also bounded by an absolute constant in the routing schedule produced. An interesting related point is that our LP relaxation is reasonable: its integrality gap (worst-case ratio between the optima of the integral and fractional versions) is bounded above by $O(1)$.

An alternative view. There is an equivalent interesting interpretation of Theorem 2.1.

THEOREM 2.4. *Suppose we have an arbitrary routing problem on an arbitrary graph $G = (V, E)$; let L be any nonnegative parameter (e.g., $O(1)$, $O(\lg n)$, $O(\sqrt{n})$). Let $\{(s_k, t_k) : 1 \leq k \leq K\}$ be the set of source-destination pairs for the packets. Suppose we can construct a probability distribution D_k on the (s_k, t_k)-paths for each k such that if we sample, for each packet k, an (s_k, t_k)-path from D_k independently of the other packets, then we have (a) for any edge $e \in E(G)$, the expected congestion on e is at most L, and (b) for each k, the expected length of the (s_k, t_k)-path chosen is at most L. Then, there is a choice of paths for each packet such that the congestion and dilation are $O(L)$. Thus, the routing can be accomplished in $O(L)$ time using constant-sized queues; such a routing can also be constructed off-line in time polynomial in $|V|$ and K.*

We remark that the converse of Theorem 2.4 is trivially true: if an $O(L)$ time routing can be accomplished, we simply let D_k place all the probability on the (s_k, t_k)-path used in such a routing.

Proof of Theorem 2.4. Let π_P^k denote the probability measure of any (s_k, t_k)-path P under the distribution D_k. Let $supp(D_k)$ denote the *support* of D_k, i.e., the set of (s_k, t_k)-paths on which D_k places nonzero probability. The proof follows from the fact that for any $(i, j) \in E(G)$,

$$x_{i,j}^k \equiv \sum_{P:(i,j)\in E(P),\ P\in supp(D_k)} \pi_P^k$$

is a feasible solution to (ROUTING), with C, D replaced by L. Hence, by our filter-round approach, we can construct one path for each packet k such that the congestion and dilation are $O(L)$. As seen above, the path selection and routing strategies can be found in polynomial time. □

We consider the above interesting because many fault-tolerance algorithms use very involved ideas to construct a suitable (s_k, t_k)-path for (most) packets [6]. These paths will need to simultaneously have small lengths and lead to small edge congestion.

Theorem 2.4 shows that much more relaxed approaches could work: a distribution that is "good" in expectation on individual elements (edges, paths) is sufficient. Recall that in many "discrete ham-sandwich theorems" (Beck and Spencer [4], Raghavan and Thompson [27]), it is easy to ensure good expectation on individual entities (e.g., the constraints of an integer program), but it is much more difficult to construct one solution that is simultaneously good on all these entities. Our result shows one natural situation where there is just a constant-factor loss in the process.

2.4. Extensions. The result above showing a constant integrality gap for packet routing can be extended to a general family of combinatorial packing problems as follows. Let S_k be the family of all the subsets of vertices S such that $s_k \in S$ and $t_k \notin S$. Recall that the (s_k, t_k)-shortest path problem can be solved as an LP via the following covering formulation:

$$\min \sum_{i,j} c_{i,j} x^k_{i,j} \text{ subject to}$$

$$\begin{aligned} \sum_{(i,j)\in E:\ i\in S, j\notin S} x^k_{i,j} &\geq 1 \quad \forall S \in S_k, \\ x^k_{i,j} &\geq 0 \quad \forall\ (i,j) \in E(G). \end{aligned} \tag{2.10}$$

Constraint (2.10) expresses the idea that "flow" crossing each s-t cut is at least 1. The following is an alternative relaxation for the packet routing problem:

$$\text{(ROUTING-II)} \quad \min(C+D)/2 \text{ subject to}$$

$$\begin{aligned} \sum_{k=1}^{K} x^k_{i,j} &\leq C \quad \forall\ (i,j) \in E(G), \\ \sum_{i,j} x^k_{i,j} &\leq D \quad \forall\ k, \\ \sum_{(i,j)\in E:\ i\in S, j\notin S} x^k_{i,j} &\geq 1 \quad \forall S \in S_k, \\ x^k_{i,j} &\in [0,1] \quad \forall\ k,i,j. \end{aligned} \tag{2.11}$$

We can use the method outlined in sections 2.1, 2.2, and 2.3 to show that the optimal solution of (ROUTING-II) is within a constant factor of the optimal routing time. A natural question that arises is whether the above conclusion holds for more general combinatorial packing problems. To address this question, we need to present an alternative (polyhedral) perspective of our (path) selection routine. First we recall some standard definitions from polyhedral combinatorics. The reader is referred to [10] for related concepts.

Suppose we are given a finite set $\mathcal{N} = \{1, 2, \ldots, n\}$ and a family $\mathcal{F}$ of subsets of $\mathcal{N}$. For any $S \subseteq \mathcal{N}$, let $\chi_S \in \{0,1\}^n$ denote the incidence vector of S. We shall consider the problem

$$(OPT) \quad \min\{c^T \chi_F : F \in \mathcal{F}\},$$

where $c \in \Re^n_+$ is a weight function on the elements of $\mathcal{N}$.

DEFINITION 2.5 (see [25]). *The blocking clutter of $\mathcal{F}$ is the family $B(\mathcal{F})$, whose members are precisely those $H \subseteq \mathcal{N}$ that satisfy the following:*

P1. *Intersection: $H \cap F \neq \emptyset$ for all $F \in \mathcal{F}$.*

P2. *Minimality: If H' is any proper subset of H, then H' violates property* P1.

A natural LP relaxation for (OPT) is

$$\min\{c^T x : x \in Q\}, \text{ where } Q = \{x^T \chi_H \geq 1 \text{ for all } H \in B(\mathcal{F}),\ x_i \geq 0 \text{ for all } i\}.$$

Q is known as the blocking polyhedron of $\mathcal{F}$. The following result is well known and easy to check:

$$Q \cap Z^n = \{x \in Z^n :\ \exists F \in \mathcal{F} \text{ such that } \ \forall i \in F,\ x_i \geq 1\}.$$

For several classes of clutters (set-systems), it is known that the extreme points of Q are the integral vectors that correspond to incidence vectors of elements in $\mathcal{F}$. By Minkowski's theorem [25], every element in Q can be expressed as a convex combination of the extreme points and extreme rays in Q. For blocking polyhedra, the set of rays is

$$\{x \in \Re^n :\ \forall i,\ x_i \geq 0\}.$$

Suppose we have a generic integer programming problem that is similar to (ROUTING-II), except for the fact that for each k, (2.11) is replaced by the constraint

$$\sum_{i \in H} x_i^k \geq 1 \ \ \forall\ H \in B(\mathcal{F}^k);$$

$\mathcal{F}^k$ can be any clutter that is well-characterized by its blocking polyhedron Q^k (i.e., the extreme points of the blocking polyhedron Q^k are incidence vectors of the elements of the clutter $\mathcal{F}^k$). Thus, we have a generalization of (ROUTING-II):

$$\text{(BLOCK)} \quad \min(C + D)/2 \text{ subject to}$$

$$\sum_{k=1}^{K} x_i^k \quad \leq \quad C \ \ \forall\ i \in \mathcal{N},$$

$$\sum_{i} x_i^k \quad \leq \quad D \ \ \forall\ k, \tag{2.12}$$

$$\sum_{i \in H} x_i^k \quad \geq \quad 1 \ \ \forall\ k \text{ and } \forall\ H \in B(\mathcal{F}^k),$$

$$x_i^k \quad \in \quad \{0, 1\} \ \ \forall\ k \text{ and } \forall i \in \mathcal{N}. \tag{2.13}$$

Note that the variables x are now indexed by elements of the set $\mathcal{N}$. In the previously discussed special cases, the elements of $\mathcal{N}$ are edges or pairs of nodes.

The LP relaxation of (BLOCK) replaces the constraint (2.13) by

$$0 \leq x_i^k \leq 1 \ \ \forall\ k = 1, \ldots, K\ \forall i \in \mathcal{N}.$$

THEOREM 2.6. *The optimal integral solution value of (BLOCK) is at most a constant factor times the optimal value of the LP relaxation.*

Proof. Let $(\overline{x}_i^k : k = 1, \ldots, K; i \in \mathcal{N})$ denote an optimal solution to the LP relaxation. By Caratheodory's theorem [25], for each fixed k, $(\overline{x}_i^k : i \in \mathcal{N})$ can be expressed as a convex combination of extreme points and extreme rays of the blocking polyhedron Q^k. However, note that the objective function can improve only by decreasing the value of $(\overline{x}_i^k)$ coordinatewise, as long as the solution remains

feasible. Furthermore, the extreme rays of the blocking polyhedron correspond to vectors v with each v_i nonnegative. Thus, w.l.o.g., we may assume that the LP optimum is *lexicographically minimal*. This ensures that the optimal solution $(\overline{x}_i^k)$ can be expressed as a convex combination of the extreme points of the polyhedron alone. As seen above, the extreme points in this case are incidence vectors of elements of the kth clutter (we use polyhedral language to let "kth clutter" denote the set-system $\mathcal{F}^k$).

Let $\overline{C}$ and $\overline{D}$ denote the fractional congestion and fractional dilation of the optimal solution obtained by the LP relaxation of (BLOCK). Let $A_1^k, A_2^k, \ldots$ denote incidence vectors of the elements in the kth clutter, and let $A_j^k(i)$ be the ith coordinate of A_j^k. Then we have a convex combination for each k:

$$\forall i,\ \overline{x}_i^k = \sum_j \alpha_j^k \cdot A_j^k(i),\ \text{where}$$

$$\alpha_j^k \geq 0\ \forall j,\ \text{and}\ \sum_j \alpha_j^k = 1.$$

Thus, by constraints (2.12), $\sum_{j:|A_j^k| \leq 2\overline{D}} \alpha_j^k \geq 1/2$, since $\sum_j \alpha_j^k |A_j^k| \leq \overline{D}$.

By filtering out those A_j^k with size greater than $2\overline{D}$, we obtain a set of canonical objects for each k, whose sizes are at most $2\overline{D}$. By scaling the α_j^k by a suitable factor, we also obtain a new set of $\overline{\alpha}_j^k$ such that

$$\sum_{j:|A_j^k| \leq 2\overline{D}} \overline{\alpha}_j^k = 1,\quad \overline{\alpha}_j^k \leq 2\alpha_j^k.$$

Using these canonical objects and $\{\overline{\alpha}_j^k\}$ as the input to Theorem 2.2, we obtain a set of objects (one from each clutter) such that the dilation is not more than $2\overline{D}$ and the congestion not more than $2(\overline{C} + \overline{D})$. Hence the solution obtained is at most $O(1)$ times the LP optimum. □

Remark. As pointed out by one of the referees, it is not clear whether the lexicographically minimal optimal solution can be constructed in polynomial time. The above result is thus only about the quality of the LP relaxation. It would be nice to find the most general conditions under which the above can be turned into a polynomial-time approximation algorithm.

3. Improved local vs. global balancing for covering. Coupled with the results of [16, 17], our approximation algorithm for the routing time (a *global* criterion) also simultaneously kept the maximum queue size (a *local capacity* constraint) constant; our approach there implicitly uses the special structure of the cut covering formulation. We now continue the study of such balancing in the context of CIPs. The reader is referred to section 1.2 for the relevant definitions and history of CIPs. In section 3.1, we will show how to approximate the global criterion well without losing much in the "local" constraints $\{x_j \leq d_j\}$. In section 3.2, we present approximation algorithms for a subfamily of CIPs where $x_j \leq d_j$ is *required* for all j. One of the key tools used in sections 3.1 and 3.2 is Theorem 3.3, which builds on an earlier rounding approach (Theorem 3.2) of [31].

3.1. Balancing local with global. The main result of section 3.1 is Corollary 3.5. This result is concerned with unweighted CIPs and the case where $\ln^+(mB/y^*) > B$. In this setting, Corollary 3.5 shows how the local capacity constraints can be violated much less in comparison with the results of [31], while keeping the objective function value within a constant factor of that of [31].

Let $\exp(x)$ denote e^x; given any nonnegative integer k, let $[k]$ denote the set $\{1, 2, \ldots, k\}$. We start by reviewing the Chernoff–Hoeffding bounds in Theorem 3.1. Let $G(\mu, \delta) \doteq (\exp(\delta)/(1+\delta)^{(1+\delta)})^{\mu}$, $H(\mu, \delta) \doteq \exp(-\mu\delta^2/2)$.

THEOREM 3.1 (see [23]). *Let $X_1, X_2, \ldots, X_\ell$ be* independent *random variables, each taking values in $[0,1]$, $R = \sum_{i=1}^{\ell} X_i$, and $E[R] = \mu$. Then, for any $\delta \geq 0$, $\Pr(R \geq \mu(1+\delta)) \leq G(\mu, \delta)$. Also, if $0 \leq \delta \leq 1$, $\Pr(R \leq \mu(1-\delta)) \leq H(\mu, \delta)$.*

We shall use the following easy fact:

$$\forall \mu \geq 0 \ \forall \delta \in [0,1], \ G(\mu, \delta) \leq \exp(-\mu\delta^2/3). \tag{3.1}$$

From now on, we will let $\{x_j^* : j \in [n]\}$ be the set of values for the variables in an arbitrary feasible solution to the LP relaxation of the CIP; thus, $0 \leq x_j^* \leq d_j$. (In particular, x^* could be an optimal LP solution.) Let $y^* = c^T \cdot x^*$. Recall that the case where $\ln^+(mB/y^*) \leq B$ is handled well in [31]; thus we shall assume $\ln^+(mB/y^*) > B$. We now summarize the main result of [31] for CIPs as a theorem. A key ingredient of Theorem 3.2 is the FKG inequality [9].

THEOREM 3.2 (see [31]). *For any given CIP, suppose we are given any $1 \leq \beta < \alpha < \kappa$ such that*

$$(1 - H(B\alpha, 1 - \beta/\alpha))^m > G(y^*\alpha, \kappa/\alpha - 1) \tag{3.2}$$

holds. Then we can find in deterministic polynomial time a vector $z = (z_1, z_2, \ldots, z_n)$ of nonnegative integers *such that* (a) $(Az)_i \geq b_i\beta$ *for each $i \in [m]$,* (b) $\sum_j c_j z_j \leq y^*\kappa$, *and* (c) $z_j \leq \lceil \alpha x_j^* \rceil \leq \lceil \alpha d_j \rceil$ *for each $j \in [n]$.*

The next theorem presents a rounding algorithm by building on Theorem 3.2.

THEOREM 3.3. *There are positive constants a_3 and a_4 such that the following holds. Given any parameter ϵ, $0 < \epsilon < 1$, let α be any value such that $\alpha \geq (a_3/\epsilon^2)\max\{\ln^+(mB/y^*)/B, 1\}$. Then we can find in deterministic polynomial time a vector $z = (z_1, z_2, \ldots, z_n)$ of nonnegative integers such that* (a) $(Az)_i \geq b_i\alpha(1-\epsilon)$ *for each $i \in [m]$,* (b) $c^T \cdot z \leq a_4 y^* \alpha$, *and* (c) $z_j \leq \lceil \alpha x_j^* \rceil \leq \lceil \alpha d_j \rceil$ *for each $j \in [n]$.*

Remark. It will be shown in the proof of Theorem 3.3 that we can choose, for instance, $a_3 = 3$ and $a_4 = 2$. Since there is a trade-off between a_3 and a_4 that can be fine-tuned for particular applications, we have avoided using specific values for a_3 and a_4 in the statement of Theorem 3.3.

The following simple proposition will also be useful.

PROPOSITION 3.4. *If $0 < x < 1/e$, then $1 - x > \exp(-1.25x)$.*

Proof of Theorem 3.3. We choose $a_3 = 3$ and $a_4 = 2$. In the notation of Theorem 3.2, we take $\beta = \alpha(1-\epsilon)$ and $\kappa = a_4\alpha$. Our goal is to validate (3.2); by (3.1), it suffices to show that

$$\exp(-y^*\alpha/3) < (1 - \exp(-B\alpha\epsilon^2/2))^m. \tag{3.3}$$

Note that the left- and right-hand sides of (3.3), respectively, decrease and increase with increasing α; thus, since $\alpha \geq \alpha_0 \doteq (3/\epsilon^2)\max\{\ln^+(mB/y^*)/B, 1\}$ it is enough to prove (3.3) for $\alpha = \alpha_0$. We consider two cases.

Case I. $\ln^+(mB/y^*) \leq B$.

Thus, $\alpha_0 = 3/\epsilon^2$ here. Since $B \geq 1$, we have $\exp(-B\alpha_0\epsilon^2/2) < 1/e$. Therefore, Proposition 3.4 implies that in order to prove (3.3), it suffices to show that

$$y^*\alpha_0/3 \geq 1.25m\exp(-B\alpha_0\epsilon^2/2),$$

i.e., that $y^*/\epsilon^2 \geq 1.25m\exp(-1.5B)$. This is true from the facts that (i) $m/y^* \leq \exp(B)$ (which follows from the fact that $\ln(m/y^*) \leq \ln^+(mB/y^*) \leq B$), and (ii) $\exp(1.5B) \geq \sqrt{e}\exp(B) \geq 1.25\epsilon^2\exp(B)$.

Case II. $\ln^+(mB/y^*) > B$.

Here, it suffices to show that

$$\exp(-y^*\ln^+(mB/y^*)/(B\epsilon^2)) < (1-\exp(-1.5\cdot\ln^+(mB/y^*)))^m. \tag{3.4}$$

Recall that $\ln^+(mB/y^*) > B \geq 1$. Therefore, we have $mB/y^* > e$, i.e., $y^*/(mB) < 1/e$. Thus,

$$(1-\exp(-1.5\cdot\ln^+(mB/y^*)))^m = \left(1-\left(\frac{y^*}{mB}\right)^{1.5}\right)^m > \exp(-1.25m(y^*/(mB))^{1.5}).$$

The inequality follows from Proposition 3.4. Therefore, to establish (3.4), we just need show that

$$y^*\ln^+(mB/y^*)/(B\epsilon^2) \geq 1.25\cdot\sqrt{\frac{y^*}{mB}}\cdot\frac{y^*}{B},$$

i.e., that $\ln^+(mB/y^*)/\epsilon^2 \geq 1.25/\sqrt{e}$, which in turn follows from the facts that $\ln^+(mB/y^*) \geq 1$ and $1/\epsilon^2 \geq 1$. This completes the proof. □

Our required result is as follows.

COROLLARY 3.5. *Given any* unweighted *CIP with* $\ln^+(mB/y^*) > B$ *and any parameter* ϵ, $0 < \epsilon < 1$, *we can find in deterministic polynomial time a vector* $v = (v_1, v_2, \ldots, v_n)$ *of nonnegative integers such that* (a) $Av \geq b$, (b) $\sum_j v_j \leq a_2 y^*(1/(1-\epsilon) + (1/\epsilon^2)\ln^+(mB/y^*)/B)$, *where* $a_2 > 0$ *is an absolute constant, and* (c) $v_j \leq \lceil d_j/(1-\epsilon)\rceil$ *for all* j.

Proof. Let $\alpha = \lceil (a_3/\epsilon^2)\ln^+(mB/y^*)/B\rceil$ and z be as in the statement of Theorem 3.3. Define $v_j = \lceil z_j/(\alpha(1-\epsilon))\rceil$ for each j. Conditions (a) and (c) are easy to check, given Theorem 3.3. Since the z_j's are all nonnegative integers and since the CIP is unweighted ($c_j = 1$ for all j), condition (b) of Theorem 3.3 shows that at most $a_4 y^*\alpha$ of them can be nonzero. Thus, condition (b) follows since $v_j \leq z_j/(\alpha(1-\epsilon))+1$ if $z_j > 0$ and since $v_j = 0$ if $z_j = 0$. □

As mentioned in section 1, this improves the value of $\max_j x_j/d_j$ from $O(\ln^+(mB/y^*)/B)$ [31] to $O(1/(1-\epsilon))$, while keeping $(c^T\cdot x)/y^*$ relatively small at $O((1/\epsilon^2)\cdot\ln^+(mB/y^*)/B)$ (as long as ϵ is a constant bounded away from 1).

3.2. Handling stringent constraints. We now handle the case where the constraints $x_j \leq d_j$ have to be satisfied and where the coefficient matrix A has only zeroes and ones. Recall from section 1 that there is a family of facility location problems where the coefficient matrix has only zeroes and ones; this is an example of the CIPs to which the following results apply.

We start with a technical lemma.

LEMMA 3.6. *For any* $0 = u_0 < u_1 \leq u_2 \leq \cdots \leq u_i$ *and any* $\ell > 0$, *the sum* $s_i = \sum_{j=1}^{i}(u_j - u_{j-1})\ln^+(\ell/u_j)$ *is at most* $u_i\ln^+(\ell/u_i) + u_i$.

Proof. If $u_1 \geq \ell/e$, then $s_i = \sum_{j=1}^{i}(u_j - u_{j-1}) = u_i$. Otherwise, let $r \geq 1$ be the highest index such that $u_r < \ell/e$. Thus, $s_i = t_r + u_i - u_r$, where $t_r = \sum_{j=1}^{r}(u_j - u_{j-1})\ln(\ell/u_j)$. Since

$$\sum_{j=1\ldots r}(u_j - u_{j-1})\ln(l/u_j) \leq \int_0^{u_r} \ln(l/x)dx = (x\ln(l/x) + x)|_0^{u_r},$$

it follows that $t_r \leq u_r \ln(\ell/u_r) + u_r$. Hence,

$$s_i = t_r + u_i - u_r \leq u_r \ln(\ell/u_r) + u_i \leq u_i \ln^+(\ell/u_i) + u_i;$$

the last inequality follows from the fact that for any $x \leq y$ such that $x < \ell/e$, $x\ln(\ell/x) \leq y\ln^+(\ell/y)$. □

The following simple proposition will also help.

PROPOSITION 3.7. *For any $\ell > 0$ and $\ell' \geq 1$, $\ln^+(\ell) \geq (\ln^+(\ell\ell'))/\ell'$.*

Proof. The proposition is immediate if $\ell\ell' \leq e$. Next note that for any $a \geq e$, the function $g_a(x) = \ln(ax)/x$ decreases as x increases from 1 to infinity. Therefore, if $\ell \leq e$ and $\ell\ell' > e$, then

$$(\ln^+(\ell\ell'))/\ell' = (\ln(\ell\ell'))/\ell' = g_\ell(\ell') \leq g_e(\ell') \leq g_e(1) = 1 = \ln^+(\ell).$$

Finally, if $\ell > e$ and $\ell\ell' > e$, then $(\ln^+(\ell\ell'))/\ell' = g_\ell(\ell') \leq g_\ell(1) = \ln^+(\ell)$. □

THEOREM 3.8. *Suppose we are given a CIP with the matrix A having only zeroes and ones. In deterministic polynomial time, we can construct a feasible solution z to the CIP with $z_j \leq d_j$ for each j, and such that the objective function value $c^T \cdot z$ is $O(y^* \ln^+(m/y^*))$.*

Proof. Let a_3 and a_4 be as in the proof of Theorem 3.3. Define $a_5 = \max\{2, 4a_3\}$ and, for any $S \subseteq [n]$, $y^*_S = \sum_{j\in S} c_j x^*_j$. Starting with $S_0 = [n]$, we construct a sequence of sets $S_0 \supset S_1 \supset \cdots$ as follows. Suppose we have constructed $S_0, S_1, \ldots, S_i$ so far. Let $h_i = y^*_{S_i}$. If $S_i = \emptyset$, we stop; or else, if all $j \in S_i$ satisfy $a_5 \ln^+(m/h_i)x^*_j \leq d_j$, we stop. If not, define the proper subset S_{i+1} of S_i to be $\{j \in S_i : a_5 \ln^+(m/h_i)x^*_j > d_j\}$. For all $j \in (S_i - S_{i+1})$, we fix z_j to be $d_j \geq x^*_j$: note that for all such j, $z_j \leq a_5 \ln^+(m/h_i)x^*_j$.

Let S_t be the final set we construct. If $S_t = \emptyset$, we do nothing more; since $z_j \geq x^*_j$ for all j, we will have $Az \geq b$ as required. Also, it is easy to check that $z_j \leq d_j$ for all j. Therefore suppose $S_t \neq \emptyset$. Let $\alpha = a_5 \ln^+(m/h_t)$. Since we stopped at the nonempty set S_t, we see that $\alpha x^*_j \leq d_j$ for all $j \in S_t$. Recall that for all $j \notin S_t$, we have fixed the value of z_j to be $d_j \geq x^*_j$. Let w denote the vector of the remaining variables, i.e., the restriction of x^* to S_t. Let A' be the submatrix of A induced by the columns corresponding to S_t. We will now focus on rounding each x^*_j ($j \in S_t$) to a suitable nonnegative integer $z_j \leq d_j$.

Define, for each $i \in [m]$,

$$b'_i = b_i - \sum_{j\notin S_t} A_{i,j} z_j;$$

since $z_j \geq x^*_j$ for all $j \notin S_t$, we get

$$(A'w)_i = \sum_{j\in S_t} A_{i,j} x^*_j \geq b'_i \ \forall i \in [m].$$

Since each b_i and $A_{i,j}$ is an integer, so is each b'_i. Suppose $b'_i \leq 0$ for some i. Then, whatever nonnegative integers z_j we round the $j \in S_t$ to, we will satisfy the constraint $(Az)_i \geq b_i$. Therefore, we can ignore such indices i and assume w.l.o.g. that $B' \doteq \min_i b'_i \geq 1$. (The constraints corresponding to indices i with $b'_i \leq 0$ can be retained as "dummy constraints.") Define $\epsilon = 1/2$; recall that $\alpha = a_5 \ln^+(m/h_t)$. Therefore Proposition 3.7 shows that

$$\alpha \geq 4a_3 \max\{\ln^+(mB'/h_t)/B', 1\},$$

i.e., that $\alpha \geq (a_3/\epsilon^2) \max\{\ln^+(mB'/h_t)/B', 1\}$. Thus, by Theorem 3.3, we can round each x^*_j ($j \in S_t$) to some nonnegative integer $z_j \leq \lceil \alpha x^*_j \rceil \leq d_j$ in such a manner that

$$\sum_{j \in S_t} c_j z_j = O(h_t \alpha), \text{ and } \forall i \in [m], (A'z)_i \geq b_i \alpha(1-\epsilon) \geq b_i; \tag{3.5}$$

the last inequality (i.e., that $\alpha(1-\epsilon) \geq 1/2$) follows from the fact that $\alpha \geq a_5 \geq 2$. Therefore we can check that the final solution is indeed feasible. We need only to bound the objective function value, which we proceed to do now.

We first bound

$$\sum_{j \notin S_t} c_j z_j = \sum_{i=0}^{t-1} \sum_{j \in (S_i - S_{i+1})} c_j z_j. \tag{3.6}$$

Fix any i, $0 \leq i \leq t-1$. Recall that for each $j \in (S_i - S_{i+1})$, we set $z_j = d_j \leq a_5 \ln^+(m/h_i) x^*_j$. Thus,

$$\sum_{j \in (S_i - S_{i+1})} c_j z_j \leq O(\ln^+(m/h_i) \sum_{j \in (S_i - S_{i+1})} c_j x^*_j) = O(\ln^+(m/h_i) \cdot (h_i - h_{i+1})). \tag{3.7}$$

Setting $u_i = h_{t+1-i}$ and substituting (3.7) into (3.6),

$$\sum_{j \notin S_t} c_j z_j = O\left(\sum_{i=2}^{t+1} (u_i - u_{i-1}) \ln^+(m/u_i)\right), \tag{3.8}$$

where $u_{t+1} = y^*$. Now, if $S_t = \emptyset$, (3.8) gives the final objective function value. Otherwise, if $S_t \neq \emptyset$, (3.5) shows that

$$\sum_{j \in S_t} c_j z_j = O(h_t \alpha) = O(u_1 \ln^+(m/u_1)).$$

This, in combination with (3.8) and Lemma 3.6, shows that $\sum_j c_j z_j = O(y^* \ln^+(m/y^*))$.

This completes the proof. □

4. Conclusion. In this paper, we analyze various classes of problems in the context of balancing global vs. local criteria.

Our main result is the first constant-factor approximation algorithm for the off-line packet routing problem on arbitrary networks: for certain positive constants c' and c'', we show that given any packet routing problem, the routing time can efficiently be approximated to within a factor of c', while ensuring that all edge-queues are of size at most c''. Our result builds on the work of [16, 17], while exploiting an interesting trade-off between a (hard) congestion criterion and an (easy) dilation

criterion. Furthermore, we show that the result can be applied to a more general setting, by providing a polyhedral perspective of our technique. Our approach of appropriately using the rounding theorem of [13] has subsequently been applied by Bar-Noy et al. [3] to develop approximation algorithms for a family of multicasting problems. It has also been applied for a family of routing problems by Andrews and Zhang [2].

The second major result in the paper improves upon a class of results in multicriteria CIPs. We show that the local criterion of unweighted CIPs can be improved from an approximately logarithmic factor to a constant factor with the global criterion not deteriorating by more than a constant factor (i.e., we maintain a logarithmic factor approximation).

The third main result improves upon a well-known bound for CIPs, in the case where the coefficient matrix A has only zeroes and ones. We show that the approximation ratio can be improved from $O(y^* \lg m)$ to $O(y^* \ln^+(m/y^*))$.

Some open questions are as follows. It would be interesting to study our packet routing algorithm empirically and to fine-tune the algorithm based on experimental observation. It would also be useful to determine the best (constant) approximation possible in approximating the routing time. An intriguing open question is whether there is a *distributed* packet routing algorithm with a constant-factor approximation guarantee. Finally, in the context of covering integer programs, can we approximate the objective function to within bounds such as ours, with (essentially) no violation of the local capacity constraints?

Acknowledgments. We thank Bruce Maggs, the STOC 1997 program committee member(s) and referee(s), and the journal referees for their helpful comments. These have helped improve the quality of this paper a great deal. In particular, one of the journal referees simplified our original proof of Lemma 3.6.

REFERENCES

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network Flows: Theory, Algorithms, and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1993.

[2] M. Andrews and L. Zhang, *Packet routing with arbitrary end-to-end delay requirements*, in Proceedings of the ACM Symposium on Theory of Computing, Atlanta, GA, 1999, pp. 557–565.

[3] A. Bar-Noy, S. Guha, J. (Seffi) Naor, and B. Schieber, *Multicasting in heterogeneous networks*, in Proceedings of the ACM Symposium on Theory of Computing, Dallas, TX, 1998.

[4] J. Beck and J. H. Spencer, *Integral approximation sequences*, Math. Programming, 30 (1984), pp. 88–98.

[5] D. Bertsimas and R. Vohra, *Rounding algorithms for covering problems*, Math. Programming, 80 (1998), pp. 63–89.

[6] R. Cole, B. M. Maggs, and R. Sitaraman, *Routing on butterfly networks with random faults*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, Milwaukee, WI, 1995, pp. 558–570.

[7] G. Dobson, *Worst-case analysis of greedy heuristics for integer programming with nonnegative data*, Math. Oper. Res., 7 (1982), pp. 515–531.

[8] M. L. Fisher and L. A. Wolsey, *On the greedy heuristic for continuous covering and packing problems*, SIAM J. Alg. Disc. Meth., 3 (1982), pp. 584–591.

[9] C. M. Fortuin, J. Ginibre, and P. N. Kasteleyn, *Correlational inequalities for partially ordered sets*, Comm. Math. Phys., 22 (1971), pp. 89–103.

[10] D. R. Fulkerson, *Blocking polyhedra*, in Graph Theory and Its Applications, B. Harris, ed., Academic Press, New York, 1970, pp. 93–112.

[11] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, *Scheduling to minimize average completion time: Off-line and on-line approximation algorithms*, Math. Oper. Res., 22 (1997), pp. 513–544.

[12] C. Kaklamanis, A. R. Karlin, F. T. Leighton, V. Milenkovic, P. Raghavan, S. B. Rao, C. Thomborson, and A. Tsantilas, *Asymptotically tight bounds for computing with faulty arrays of processors*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, St. Louis, MO, 1990, pp. 285–296.

[13] R. M. Karp, F. T. Leighton, R. L. Rivest, C. D. Thompson, U. V. Vazirani, and V. V. Vazirani, *Global wire routing in two-dimensional arrays*, Algorithmica, 2 (1987), pp. 113–129.

[14] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, San Mateo, CA, 1992.

[15] F. T. Leighton, *Methods for message routing in parallel machines*, in Proceedings of the ACM Symposium on the Theory of Computing, Victoria, Canada, 1992, pp. 77–96.

[16] F. T. Leighton, B. M. Maggs, and S. B. Rao, *Packet routing and job-shop scheduling in O(congestion+dilation) steps*, Combinatorica, 14 (1994), pp. 167–186.

[17] F. T. Leighton, B. M. Maggs, and A. Richa, *Fast algorithms for finding O(congestion+dilation) packet routing schedules*, Combinatorica, 19 (1999), pp. 375–401.

[18] F. T. Leighton, B. M. Maggs, and R. K. Sitaraman, *On the fault tolerance of some popular bounded-degree networks*, SIAM J. Comput., 27 (1998), pp. 1303–1333.

[19] F. T. Leighton and S. B. Rao, *An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms*, in Proceedings of the IEEE Symposium on Foundations of Computer Science, White Plains, NY, 1988, pp. 422–431.

[20] J. H. Lin and J. S. Vitter, *ϵ-approximations with minimum packing constraint violation*, in Proceedings of the ACM Symposium on Theory of Computing, Victoria, Canada, 1992, pp. 771–782.

[21] M. V. Marathe, R. Ravi, and R. Sundaram, *Service-constrained network design problems*, Nordic J. Comput., 3 (1996), pp. 367–387.

[22] S. T. McCormick and M. L. Pinedo, *Scheduling n independent jobs on m uniform machines with both flow time and makespan objectives: A parametric approach*, ORSA J. Comput., 7 (1992), pp. 63–77.

[23] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, Cambridge, UK, 1995.

[24] M. Naor and R. M. Roth, *Optimal file sharing in distributed networks*, SIAM J. Comput., 24 (1995), pp. 158–183.

[25] G. L. Nemhauser and L. A. Wolsey, *Integer and Combinatorial Optimization*, Wiley and Sons, New York, 1988.

[26] R. Ostrovsky and Y. Rabani, *Universal O(congestion + dilation $+\log^{1+\epsilon} N$) local control packet switching algorithms*, in Proceedings of the ACM Symposium on Theory of Computing, El Paso, TX, 1997, pp. 644–653.

[27] P. Raghavan and C. D. Thompson, *Randomized rounding: A technique for provably good algorithms and algorithmic proofs*, Combinatorica, 7 (1987), pp. 365–374.

[28] A. Ranade, *How to emulate shared memory*, J. Comput. System Sci., 41 (1991), pp. 307–326.

[29] R. Ravi, M. V. Marathe, S. S. Ravi, D. J. Rosenkrantz, and H. B. Hunt III, *Many birds with one stone: Multi-objective approximation algorithms*, in Proceedings of the ACM Symposium on Theory of Computing, San Diego, CA, 1993, pp. 438–447.

[30] D. B. Shmoys and É. Tardos, *An approximation algorithm for the generalized assignment problem*, Math. Programming, 62 (1993), pp. 461–474.

[31] A. Srinivasan, *Improved approximation guarantees for packing and covering integer programs*, SIAM J. Comput., 29 (1999), pp. 648–670.

[32] C. Stein and J. Wein, *On the existence of schedules that are near-optimal for both makespan and total weighted completion time*, Oper. Res. Lett., 21 (1997), pp. 115–122.

[33] L. G. Valiant, *A scheme for fast parallel communication*, SIAM J. Comput., 11 (1982), pp. 350–361.

[34] L. G. Valiant and G. J. Brebner, *Universal schemes for parallel communication*, in Proceedings of the ACM Symposium on Theory of Computing, Milwaukee, WI, 1981, pp. 263–277.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 2069–2083

VARIABLE-SIZED BIN PACKING: TIGHT ABSOLUTE WORST-CASE PERFORMANCE RATIOS FOR FOUR APPROXIMATION ALGORITHMS*

CHENGBIN CHU† AND RÉMY LA†

Abstract. In this paper we consider a one-dimensional bin packing problem where the bins do not have identical sizes, or a variable-sized bin packing problem, to minimize the bin consumption, i.e., the total size of the opened bins. The identical size problem has been extensively studied in the literature both for static and dynamic settings. The worst-case or average-case performance has been analyzed. Our problem setting particularly arises in metal cutting industries. Therefore, it presents a great practical relevance. Four greedy approximation algorithms based on a construction approach called largest object first with least absolute waste (LFLAW), largest object first with least relative waste (LFLRW), least absolute waste (LAW), and least relative waste (LRW) are examined. Their absolute worst-case performances are analyzed. The worst case bounds are 2 for LFLAW and LFLRW, 3 for LAW, and $2 + \ln 2$ for LRW. We also show that these worst-case bounds are tight.

Key words. variable-sized bin packing, absolute worst-case performance ratios, approximation algorithms

AMS subject classifications. 68Q25, 68R05, 90B99

PII. S009753979834669X

1. Introduction.

This paper addresses the worst-case performance of four approximation algorithms to solve a one-dimensional bin packing problem. The performance of an approximation algorithm can be measured by worst-case performance bounds. Two types of bounds can be distinguished: asymptotic worst-case performance bounds and absolute worst-case performance bounds. If, for a given instance I, $C_{\mathrm{H}}(I)$ and $C^*(I)$ are the criterion value given by approximation algorithm H and the optimal criterion value, respectively, an *asymptotic worst-case analysis* consists of finding a relation of the form

$$C_{\mathrm{H}}(I) \leq R \times C^*(I) + A \qquad \text{for any } I,$$

where A and R are nonnegative constants independent of I. An *absolute worst-case analysis* establishes a relation of the form

$$C_{\mathrm{H}}(I) \leq \Omega \times C^*(I) \qquad \text{for any } I.$$

The smallest values of R and Ω satisfying the relations above are called *asymptotic worst-case performance ratio* and *absolute worst-case performance ratio* and denoted as R_{H} and Ω_{H}, respectively. Sometimes, R_{H} and Ω_{H} are also said to be *tight asymptotic worst-case bound* and *tight absolute worst-case bound*, respectively. Therefore, we have

$$R_{\mathrm{H}} = \lim_{k \to +\infty} \sup_I \left\{ \frac{C_{\mathrm{H}}(I)}{C^*(I)} \middle| C^*(I) \geq k \right\},$$

$$\Omega_{\mathrm{H}} = \sup_I \left\{ \frac{C_{\mathrm{H}}(I)}{C^*(I)} \right\} = \inf \left\{ \rho \middle| \frac{C_{\mathrm{H}}(I)}{C^*(I)} \leq \rho \quad \forall I \right\}.$$

*Received by the editors November 6, 1998; accepted for publication (in revised form) September 29, 2000; published electronically April 18, 2001.
http://www.siam.org/journals/sicomp/30-6/34669.html

†Laboratoire d'Optimisation des Systèmes Industriels (LOSI), Université de Technologie de Troyes, 12 rue Marie Curie - BP 2060, 10010 Troyes Cedex, France (chu@univ-troyes.fr, remy.la@sgr.saint-gobain.com).

Note that there may be no instance I such that $\frac{C_{\mathrm{H}}(I)}{C^*(I)} = \Omega_{\mathrm{H}}$. In this case, Ω_{H} can only be asymptotically reached; i.e., for any $\varepsilon > 0$, there is an instance I such that $0 \leq \Omega_{\mathrm{H}} - \frac{C_{\mathrm{H}}(I)}{C^*(I)} \leq \varepsilon$. By definition, the following relation is true:

$$R_{\mathrm{H}} \leq \Omega_{\mathrm{H}}.$$

The problem considered in this paper is a one-dimensional bin packing problem. The classical one-dimensional bin packing problem, called unit-capacity bin packing in the literature, can be described as follows: Given a list of m objects $\mathcal{L} = \{1, 2, \ldots, m\}$ with sizes $s_i \leq 1$ $(i = 1, 2, \ldots, m)$, pack all the objects into a minimum number of bins of size 1. Because of the NP-hardness of the problem and the limitation on computation time due to specific application environments (telecommunications and computer systems, for instance), low complexity methods are needed. Many approximation algorithms, sometimes even with linear complexity, have been proposed with evaluated performance. These methods can be classified into on-line algorithms, such as first fit (FF) and best fit (BF), and off-line algorithms, such as first fit decreasing (FFD) and best fit decreasing (BFD). The performance is evaluated either in average-case [7], [5] or in worst-case [4], [6]. An excellent survey of the research on this problem was given by Coffman, Garey, and Johnson, [4]. This survey was again updated in 1996 [6]. Johnson [12] examined the asymptotic worst-case performance of four approximation algorithms: FF, BF, FFD, and BFD. Tighter bounds on the performance of FF and BF are given in [10] and those of FFD and BFD by Baker [2], respectively. Ivanyi [11] studied the absolute worst-case performances of these four approximation algorithms. Simchi–Levi [17] proved that the absolute worst-case performance ratio is 1.5 for FFD and BFD and it is at least 1.7, but no more than 1.75, for FF and BF. The asymptotic worst-case ratios for best-k-fit and next-k-fit are given in [14], [15], respectively. That for HARMONIC$_M$ is given in [13].

In this paper, we study a slightly different problem, a so-called variable-sized bin packing problem: Given a set $\mathcal{L}$ of m objects $\{1, 2, \ldots, m\}$ with sizes s_i $(i = 1, 2, \ldots, m)$ and given M types of bins of sizes S_j $(i = 1, 2, \ldots, M)$, pack all the objects so that the sum of the sizes of the bins used (called *bin consumption* hereafter) is minimum. It is generally assumed that any object of $\mathcal{L}$ can be packed into at least one type of bin and there is an inexhaustible supply of bins of each type. Compared with the unit-capacity bin packing problem, variable-sized bin packing received less attention, because it is more general and therefore more difficult, but is receiving more and more interest from researchers [9], [16], [8], [18] due to its variety of interpretations from computer storage allocation to stock cutting. As for unit-capacity bin packing, both on-line [8], [18] and off-line [9], [16] settings are studied. However, only asymptotic worst-case performance ratios are given for proposed algorithms. In this paper, we propose four greedy approximation algorithms and we analyze their absolute worst-case performance. This work is motivated by a real life cutting stock problem for which we developed effective approximation algorithms to fulfill industrial requirements [1], [3]. The real life problem has a more complex model and the proposed algorithms are more sophisticated than those analyzed in this paper. To some extent, the results of this paper can be seen as a preliminary step for the worst-case analysis of algorithms proposed in [1] and [3].

Section 2 describes the four approximation algorithms. In section 3, their absolute worst-case performance is analyzed.

2. Description of the approximation algorithms. Let M be the number of bin types. Let S_j be the size of bins of type j $(j = 1, 2, \ldots, M)$. The number of bins

in each type is assumed to be infinite. Without loss of generality, we assume that the objects of $\mathcal{L}$ are arranged into nonincreasing size order ($s_1 \geq s_2 \geq \cdots \geq s_m$). Before properly describing the approximation algorithms, we introduce a subproblem.

2.1. A subproblem. For any approximation algorithm considered in this paper, there is a subproblem to be solved: Pack a subset of objects from a given set E into a given type j of bin. This subproblem, which is a knapsack problem, can be approximately solved by the FFD method: At each step, the lowest indexed (i.e., the longest) remaining object is packed among those fitting the remaining capacity of the bin. This process is repeated until either there is no more unpacked object or no object fits the remaining size of the bin.

Another way to describe this procedure is the following. The objects are arranged in a list in increasing order of their indexes. The first object in the list fitting the remaining capacity of the bin is packed and then removed from the list. This process is repeated until either no more object in the list fits the remaining capacity of the bin or the list becomes empty.

Note that if an object does not fit the remaining capacity of the bin at a step, it does not fit the remaining capacity of the bin at a subsequent step, since the remaining capacity decreases along with the procedure. As a consequence, the object packed at a step is necessarily larger indexed than those packed at previous steps.

Throughout the paper, $\sigma(j, E)$, $\delta(j, E)$, and $\omega(j, E)$ denote the set of packed objects, the total size of these objects, and the resulting waste (unused capacity of the bin), respectively. By notation, we have

$$\delta(j, E) = \sum_{i \in \sigma(j,E)} s_i,$$
$$\omega(j, E) = S_j - \delta(j, E).$$

An important property of the solution is the following.

PROPERTY 2.1. *For any unpacked object $i \in E - \sigma(j, E)$, we have*

$$s_i > S_j - \sum_{i' \in \sigma(j,E), i' < i} s_{i'} = \omega(j, E) + \sum_{i' \in \sigma(j,E), i' > i} s_{i'}.$$

Proof. Assume that there is an object $i^* \in E - \sigma(j, E)$ such that $s_{i^*} \leq S_j - \sum_{i' \in \sigma(j,E), i' < i^*} s_{i'}$. Let $A = \{i | i \in \sigma(j, E), i < i^*\}$. After the objects in A are packed, the remaining capacity is $S_j - \sum_{i \in A} s_i$. By assumption, object i^* fits the remaining capacity of the bin. Therefore, the index of the next object to be packed must be less than or equal to i^*, which is in contradiction with the fact that the next object actually packed belongs to $\sigma(j, E) - A = \{i | i \in \sigma(j, E), i > i^*\}$. □

The consequence of this property is the following corollaries.

COROLLARY 2.2. *For any unpacked object $i \in E - \sigma(j, E)$, if $i < \inf \sigma(j, E)$, and in particular if $s_i > \delta(j, E)$, then $s_i > S_j$.*

This corollary holds, since if $i < \inf \sigma(j, E)$, then $\sum_{i' \in \sigma(j,E), i' < i} s_{i'} = 0$.

COROLLARY 2.3. *For any unpacked object $i \in E - \sigma(j, E)$, we have $s_i > \omega(j, E)$.*

This corollary holds since $\sum_{i' \in \sigma(j,E), i' > i} s_{i'} \geq 0$.

PROPERTY 2.4. *Let E' be a set of objects such that $E \subseteq E'$ and $\inf(E' - E) > \sup \sigma(j, E)$. Then $\sigma(j, E) \subseteq \sigma(j, E')$.*

Proof. In the list corresponding to E', the objects are arranged so that objects in E are at the head of the list and the remaining ones are arranged at the tail. According

to the procedure, an object in $E' - E$ can be packed only if no more object in E can fit the remaining capacity of the bin. Before this stage, the procedure proceeds as if there were only objects in E; i.e., all objects in $\sigma(j, E)$ are packed. □

2.2. Approximation algorithms. The proposed approximation algorithms are iterative procedures. At each step, a subset of objects is packed into one chosen bin. Let $\mathcal{L}_k$ be the set of unpacked objects at the *beginning* of the kth step ($\mathcal{L}_1 = \mathcal{L}$). At step k, the set of appropriate bin types T_k is identified and for each appropriate bin type $j \in T_k$, the subproblem corresponding to the bin type j and the set of unpacked objects $\mathcal{L}_k$ is solved and the corresponding waste $\omega(j, \mathcal{L}_k)$ is computed.

One bin is actually chosen according to a given indicator specific to each approximation algorithm. Let t_k be the type of the bin chosen at step k. The subset $\sigma(t_k, \mathcal{L}_k)$ of objects which are packed into one bin of type t_k is removed from $\mathcal{L}_k$. This gives the set of unpacked objects at the beginning of the $k + 1$st step; namely, $\mathcal{L}_{k+1} = \mathcal{L}_k - \sigma(t_k, \mathcal{L}_k)$.

Let largest object first with least absolute waste (LFLAW), largest object first with least relative waste (LFLRW), least absolute waste (LAW), and least relative waste (LRW) be the names of the four approximation algorithms, respectively.

In LFLAW and LFLRW, only those bin types at least as large as the longest unpacked object are considered to be appropriate (i.e., $T_k = \{j | S_j \geq s_i \ \forall\, i \in \mathcal{L}_k\}$) while in LAW and LRW, all bin types are appropriate at each step (i.e., $T_k = \{1, 2, \ldots, M\}$ for any k).

In LFLAW and LAW, the bin type t_k is the one that generates the lowest absolute waste $\omega(j, \mathcal{L}_k)$:

$$t_k = \arg\min_{j \in T_k} \omega(j, \mathcal{L}_k) = \arg\min_{j \in T_k} \{S_j - \delta(j, \mathcal{L}_k)\}.$$

In LFLRW and LRW, the bin type t_k is the one that generates the lowest relative waste:

$$t_k = \arg\min_{j \in T_k} \frac{\omega(j, \mathcal{L}_k)}{S_j},$$

or equivalently

$$t_k = \arg\max_{j \in T_k} \frac{\delta(j, \mathcal{L}_k)}{S_j} = \arg\min_{j \in T_k} \frac{\omega(j, \mathcal{L}_k)}{\delta(j, \mathcal{L}_k)}.$$

It can be shown that the four approximation algorithms have the same complexity, which is $O(m^2 M)$.

3. Worst-case performance analysis. Throughout the paper, we assume, without loss of generality, that $\sum_{i=a}^{b} x_i = 0$ whenever $b < a$. The following notation is also used in this paper.

NOTATION 3.1.

N: number of bins used to pack all objects of $\mathcal{L}$ according to the considered approximation algorithm, which is also equal to the number of steps needed for the approximation algorithm to pack all of the objects;

P_k: set of objects packed at step k: $P_k = \sigma(t_k, \mathcal{L}_k)$;

l_k: total size of the objects packed at step k: $l_k = \delta(t_k, \mathcal{L}_k) = \sum_{i \in P_k} s_i$;

L_k: size of the bin chosen at step k: $L_k = S_{t_k}$;

w_k: waste generated at step k: $w_k = L_k - l_k = \omega(t_k, \mathcal{L}_k)$;

u_k: *the largest indexed (i.e., the shortest) unpacked object at the* beginning *of step* k: $u_k = \sup \mathcal{L}_k$;

v_k: *the smallest indexed (i.e., the longest) unpacked object at the* beginning *of step* k: $v_k = \inf \mathcal{L}_k$;

Λ: *total size of all objects in* $\mathcal{L}$;

C_{H}: *total size of the bins used to pack the objects of* $\mathcal{L}$ *according to approximation algorithm* H;

C^*: *total size of the bins used to pack the objects of* $\mathcal{L}$ *in an optimal solution;*

α: *the smallest* k *such that* L_k *is the longest.*

By notation, we have the following properties.

PROPERTY 3.2. *In the solution given by LFLAW or LFLRW, we have* $v_k \in P_k$ *for any* $1 \le k \le N$.

PROPERTY 3.3. *In a solution given by any one of the approximation algorithms proposed, we have* $L_k < L_\alpha$ *for any* k *such that* $1 \le k < \alpha$ *and* $L_k \le L_\alpha$ *for any* k *such that* $\alpha \le k \le N$.

PROPERTY 3.4. *For any* k $(k = 1, 2, \ldots, N-1)$ *and for any* $i \in \mathcal{L}_{k+1}$ *we have* $w_k < s_i$.

Proof. This relation is straightforward after Corollary 2.3 by taking into account the fact that $\mathcal{L}_{k+1} = \mathcal{L}_k - \sigma(t_k, \mathcal{L}_k)$. □

The following corollary is one of the consequences of this property.

COROLLARY 3.5. *For any* k $(k = 1, 2, \ldots, N-1)$, *and for any* k' $(k' = k+1, k+2, \ldots, N)$, *we have* $w_k < l_{k'}$.

Proof. For any object $i \in P_{k'}$, we have $i \in \mathcal{L}_{k+1}$. Therefore $l_{k'} \ge s_i > w_k$, according to Property 3.4. □

PROPERTY 3.6. *For any approximation algorithm considered in this paper, we have* $L_N = \min\{S_j | S_j \ge l_N, j = 1, 2, \ldots, M\}$. *In other words, the last bin is the shortest one among those being capable of containing all objects of* $\mathcal{L}_N$.

Proof. Assume that there is a bin type j such that $S_j \ge l_N$ and $L_N > S_j$. This implies that $\delta(j, \mathcal{L}_N) = \delta(t_N, \mathcal{L}_N) = l_N$ and $\omega(j, \mathcal{L}_N) < \omega(t_N, \mathcal{L}_N)$. This is in contradiction with the fact that $\omega(t_N, \mathcal{L}_N) \le \omega(j, \mathcal{L}_N)$ for LFLAW and LAW and with the fact that $\frac{\omega(t_N, \mathcal{L}_N)}{\delta(t_N, \mathcal{L}_N)} \le \frac{\omega(j, \mathcal{L}_N)}{\delta(j, \mathcal{L}_N)}$ for LFLRW and LRW. □

The following corollary is a consequence of this property.

COROLLARY 3.7. *For any bin type* j *such that* $S_j < L_N$, *we have* $S_j < l_N$.

LEMMA 3.8. *For any* $\tau \in \{1, 2, \ldots, M\}$, *if* $S_\tau \ge s_{v_N}$ *and* $S_\tau < L_N$, *then* $S_\tau < 2\delta(\tau, \mathcal{L}_N)$.

Proof. The fact that $S_\tau \ge s_{v_N}$ implies $v_N \in \sigma(\tau, \mathcal{L}_N)$ and consequently

$$s_{v_N} \le \delta(\tau, \mathcal{L}_N). \tag{3.1}$$

Due to Corollary 3.7, the assumptions of Lemma 3.8 imply

$$S_\tau < l_N. \tag{3.2}$$

Let $G = \mathcal{L}_N - \sigma(\tau, \mathcal{L}_N)$. As S_τ is shorter than l_N (see 3.2), all objects of $\mathcal{L}_N$ cannot be packed into one bin of type τ; G is then nonempty. Furthermore, from Corollary 2.3, we obtain

$$s_i > S_\tau - \delta(\tau, \mathcal{L}_N) \qquad \forall i \in G, \tag{3.3}$$

By notation of v_N, we also have

$$s_i \le s_{v_N} \qquad \forall i \in G. \tag{3.4}$$

From (3.1), (3.3), and (3.4), we derive

$$S_\tau - \delta(\tau, \mathcal{L}_N) < s_{v_N} \le \delta(\tau, \mathcal{L}_N).$$

Therefore,

$$S_\tau < 2\delta(\tau, \mathcal{L}_N). \qquad \square$$

3.1. Worst-case analysis of algorithms LFLAW and LFLRW. An interesting property of the solution yielded by LFLAW or LFLRW is that all bins are used more than half, except perhaps the last one. This is shown in the following lemma.

LEMMA 3.9. *In the solution given by LFLAW or LFLRW, for any k such that $1 \le k < N$, we have $L_k < 2l_k$.*

Proof. According to LFLAW and LFLRW, for any k such that $1 \le k < N$, we have $v_k \in P_k$ and consequently $s_{v_k} \le l_k$. According to Corollary 3.4, $w_k < s_{u_{k+1}}$ for any k such that $1 \le k < N$. If there is some k such that $1 \le k < N$ and $l_k \le w_k$, we would have $s_{v_{k+1}} \le s_{v_k} \le l_k \le w_k < s_{u_{k+1}}$, which is in contradiction with the fact that $u_{k+1} \ge v_{k+1}$. $\square$

LEMMA 3.10. *In the solution given by LFLAW or LFLRW, if $\alpha \ne N$, then the bin consumption $C < 2C^*$.*

Proof. The bin consumption C can be upper bounded as follows:

$$C = \sum_{k=1}^{N} L_k \le \sum_{k=1, k\ne\alpha}^{N-1} L_k + 2L_\alpha = \sum_{k=1, k\ne\alpha}^{N-1} L_k + 2(l_\alpha + w_\alpha).$$

Using Lemma 3.9, this relation leads to

$$C < 2 \sum_{k=1, k\ne\alpha}^{N-1} l_k + 2(l_\alpha + w_\alpha).$$

Furthermore, from Corollary 3.5, $w_\alpha < l_N$. Therefore,

$$C < 2 \sum_{k=1, k\ne\alpha}^{N-1} l_k + 2(l_\alpha + l_N) = 2\Lambda \le 2C^*. \qquad \square$$

LEMMA 3.11. *In the solution given by LFLAW or LFLRW, if $\alpha = N$, then $C < 2C^*$.*

Proof. We first prove that

$$L_N < 2l_N. \tag{3.5}$$

By notation, we have $s_{v_N} \le s_{v_{N-1}}$. Since $v_{N-1} \in P_{N-1}$ (see Property 3.2), we should have $s_{v_{N-1}} \le L_{N-1}$. Therefore,

$$s_{v_N} \le L_{N-1}. \tag{3.6}$$

In other words, v_N can be packed into a bin of type t_{N-1}; i.e., $v_N \in \sigma(t_{N-1}, \mathcal{L}_N)$. The fact that $\alpha = N$ implies

$$L_{N-1} < L_N \tag{3.7}$$

according to Property 3.3. According to Lemma 3.8, relations (3.6) and (3.7) imply

$$L_{N-1} < 2\delta(t_{N-1}, \mathcal{L}_N). \tag{3.8}$$

For LFLAW, this relation implies

$$\begin{aligned} w_N &= \omega(t_N, \mathcal{L}_N) \le \omega(t_{N-1}, \mathcal{L}_N) \\ &= L_{N-1} - \delta(t_{N-1}, \mathcal{L}_N) \\ &< \delta(t_{N-1}, \mathcal{L}_N) \le l_N \end{aligned}$$

which is equivalent to (3.5).

For LFLRW, (3.8) implies that

$$L_N/l_N \le L_{N-1}/\delta(t_{N-1}, \mathcal{L}_N) < 2$$

which is equivalent to (3.5).

As a consequence, if $\alpha = N$, (3.5) holds for both the solutions given by LFLAW and LFLRW.

Considering Lemma 3.9 and relation (3.5), we obtain

$$C = \sum_{k=1}^{N} L_k < 2 \sum_{k=1}^{N} l_k = 2\Lambda \le 2C^* \qquad \square$$

The following theorem is straightforward from Lemmas 3.10 and 3.11.

THEOREM 3.12. *For any instance, we have* $C_{\mathrm{LFLAW}}/C^* < 2$ *and* $C_{\mathrm{LFLRW}}/C^* < 2$.

THEOREM 3.13. *The upper bound* 2 *is tight for* C_{LFLAW}/C^* *and* C_{LFLRW}/C^*; *i.e.,* $\Omega_{\mathrm{LFLAW}} = \Omega_{\mathrm{LFLRW}} = 2$.

Proof. For LFLAW, consider the following instance composed of two types of bins with sizes $S_1 = n + 1$ and $S_2 = 2 - 1/n$ (n being a positive integer) and n objects of same sizes $l_1 = l_2 = \cdots = l_n = 1$ have to be packed. In the solution given by LFLAW, each object is packed into one bin of type 2, whereas the optimal solution consists of packing all objects into one bin of type 1. Therefore $C_{\mathrm{LFLAW}} = n(2 - 1/n) = 2n - 1$ and $C^* = n + 1$. When n tends towards $+\infty$, the ratio C_{LFLAW}/C^* tends towards 2.

For LFLRW, consider the following instance composed of three types of bins with sizes $S_1 = 2n$, $S_2 = 2$, and $S_3 = 1$ (n being a positive integer). The $n + 1$ objects to be packed are of sizes $s_1 = 1 + 1/n$, $s_2 = s_3 = \cdots = s_{n+1} = 1$. According to LFLRW, all objects are packed into one bin of type 1, whereas the optimal solution consists of packing object 1 into one bin of type 2 and each object i ($i > 1$) into one bin of type 3. Therefore $C_{\mathrm{LFLRW}} = 2n$ and $C^* = n + 2$. When n tends towards $+\infty$, the ratio C_{LFLRW}/C^* tends towards 2. $\square$

3.2. Worst-case performance of LAW. This subsection is devoted to the worst-case performance of LAW. As for LFLAW and LFLRW, we first prove some lemmas used in the proof of the worst-case bound.

LEMMA 3.14. *In the solution given by LAW,* $L_N \le \max(C^*, 2l_N)$.

Proof. Assume that $L_N > \max(C^*, 2l_N)$; i.e., $L_N > C^*$ and $L_N > 2l_N$, from which we can derive

$$l_N < L_N - l_N. \tag{3.9}$$

Let τ be the type of the bin into which object v_N is packed in an optimal packing. By notation,

$$s_{v_N} \leq S_\tau \leq C^*. \tag{3.10}$$

Together with the assumption that $L_N > C^*$, we obtain

$$S_\tau < L_N. \tag{3.11}$$

From Corollary 3.7, we obtain

$$S_\tau - \delta(\tau, \mathcal{L}_N) < S_\tau < l_N.$$

Considering (3.9), we obtain

$$S_\tau - \delta(\tau, \mathcal{L}_N) < l_N < L_N - l_N$$

which is in contradiction with the principle of LAW. □

THEOREM 3.15. *For any instance,* $C_{\mathrm{LAW}}/C^* < 3$.

Proof. The bin consumption C_{LAW} is given by

$$C_{\mathrm{LAW}} = \sum_{k=1}^{N} L_k = \sum_{k=1}^{N-1} (l_k + w_k) + L_N.$$

According to Corollary 3.5, we have

$$\sum_{k=1}^{N-1} w_k < \sum_{k=2}^{N} l_k.$$

Consequently

$$\begin{aligned} C_{\mathrm{LAW}} &< \sum_{k=1}^{N-1} l_k + \sum_{k=2}^{N} l_k + L_N \\ &< 2\Lambda - l_N + L_N. \end{aligned}$$

According to Lemma 3.14, we obtain

$$C_{\mathrm{LAW}} < 2\Lambda + \max(C^* - l_N, l_N).$$

Further considering the fact that $\Lambda \leq C^*$, $C^* - l_N \leq C^*$, and $l_N \leq C^*$, the relation above allows us to prove the lemma. □

THEOREM 3.16. *The upper bound* 3 *is tight for* C_{LAW}/C^*, *i.e.,* $\Omega_{\mathrm{LAW}} = 3$.

Proof. Consider the following instance: Given two types of bins with sizes $S_1 = n + 3$ and $S_2 = 2 - 1/n$ (n being a positive integer), the $n + 1$ objects to be packed are with sizes $s_1 = 2$, $s_2 = s_3 = \cdots = s_{n+1} = 1$.

The packing provided by LAW is as follows: Each of the objects $2, 3, \ldots, n+1$ is packed into one bin of type 2 and object 1 is packed into one bin of type 1. For this instance, $C_{\mathrm{LAW}} = n(2 - 1/n) + n + 3 = 3n + 2$. In the optimal packing all objects are packed into one bin of type 1. Therefore, $C^* = n + 3$. The ratio between C_{LAW} and C^* tends towards 3 when n tends towards $+\infty$. □

3.3. Worst-case performance bound of LRW. In this subsection, we use the following notation.

NOTATION 3.17.

Q_k: the total size of objects packed from step k to step N; i.e., $Q_k = \sum_{j=k}^{N} l_j$;

p: index such that $Q_p \le L_N$ and $Q_{p-1} > L_N$; without loss of generality, p is set to 1 if $L_N \ge Q_1 = \Lambda$;

q: index such that $Q_q \le L_N/2$ and $Q_{q-1} > L_N/2$; without loss of generality, q is set to 1 if $L_N \ge 2Q_1 = 2\Lambda$;

r: the smallest k such that $L_k \ge 2l_k$; if $L_k < 2l_k$ $\forall$ k such that $1 \le k \le N$, then r is set to $N+1$, without loss of generality.

By notation, we have

$$\Lambda = Q_1 > Q_2 > \cdots > Q_N,$$
$$p \le q. \tag{3.12}$$

The last relation can be justified as follows. If $p > q$, we would have $p - 1 \ge q$ and therefore $Q_{p-1} \le Q_q$ from which it follows $L_N < Q_{p-1} \le Q_q \le L_N/2$ which implies $L_N < 0$. This latter relation, however, is impossible.

For any k such that $k \ge p$, we have $Q_k \le L_N$; i.e., all objects in $\mathcal{L}_k$ can be packed into one bin of type t_N. Therefore, $\delta(t_N, \mathcal{L}_k) = Q_k$. On the other hand, $\delta(t_k, \mathcal{L}_k) = l_k$. According to LRW, we must have

$$\frac{l_k}{L_k} = \frac{\delta(t_k, \mathcal{L}_k)}{S_{t_k}} \ge \frac{\delta(t_N, \mathcal{L}_k)}{S_{t_N}} = \frac{Q_k}{L_N}, \quad k = p, p+1, \ldots, N. \tag{3.13}$$

In LRW, if one bin is packed at less than the half of its size, the subsequent bins (in the packing order) are also packed at less than the half of their size. That is the purpose of the following lemma.

LEMMA 3.18. *In the solution given by LRW, if $r < N$, then for any k such that $r \le k < N$, we have $L_k \ge 2l_k$ and $l_{k+1} > L_k$.*

Proof. It is sufficient to prove that for any k such that $1 \le k < N$, if $L_k \ge 2l_k$, then $L_{k+1} \ge 2l_{k+1}$, $l_{k+1} > L_k$.

From the assumption that $L_k \ge 2l_k$, we can derive $w_k = L_k - l_k \ge l_k$. According to Corollary 2.3, we have

$$s_i > w_k \ge l_k \qquad \forall i \in \mathcal{L}_{k+1} \tag{3.14}$$

which means that $\sup \mathcal{L}_{k+1} < \inf P_k = \inf(\mathcal{L}_k - \mathcal{L}_{k+1})$.

From Property 2.4, it follows that $P_{k+1} = \sigma(t_{k+1}, \mathcal{L}_{k+1}) \subseteq \sigma(t_{k+1}, \mathcal{L}_k)$. This implies that

$$w_{k+1} = \omega(t_{k+1}, \mathcal{L}_{k+1}) \ge \omega(t_{k+1}, \mathcal{L}_k). \tag{3.15}$$

On the other hand, according to LRW, we have $\omega(t_{k+1}, \mathcal{L}_k)/L_{k+1} \ge w_k/L_k \ge 1/2$ which implies

$$\omega(t_{k+1}, \mathcal{L}_k) \ge \frac{1}{2} L_{k+1}. \tag{3.16}$$

The two last relations (3.15) and (3.16) give $w_{k+1} \ge \frac{1}{2} L_{k+1}$; namely, $L_{k+1} \ge 2l_{k+1}$.

Relation (3.14) also implies $s_{u_{k+1}} > L_k$, according to Corollary 2.2. Considering the fact that $l_{k+1} \ge s_{u_{k+1}}$, we can conclude that $l_{k+1} > L_k$. ∎

From Lemma 3.18, we can deduce $r \geq q$. This relation is obviously true if $r = N+1$. If $r \leq N$, we have $L_N \geq 2l_N > l_N + L_{N-1} = l_N + 2l_{N-1} > \cdots > l_N + l_{N-1} + \cdots + 2l_r > Q_r$. Therefore $r \geq p$. Considering (3.13), this implies $2 \leq L_r/l_r \leq L_N/Q_r$; i.e., $Q_r \leq L_N/2$. As a consequence $r \geq q$. Together with the relation $p \leq q$, we obtain

$$1 \leq p \leq q \leq r \leq N+1.$$

The following corollary is a consequence of Lemma (3.18).

COROLLARY 3.19. *If $L_\alpha \geq 2l_\alpha$, then $r \leq N$ and $\alpha = N$ ($L_k < L_N$ for any k such that $1 \leq k < N$).*

LEMMA 3.20. *In the solution given by LRW, if $L_\alpha < 2l_\alpha$, then $C_{\mathrm{LRW}} < 2C^*$.*

Proof. The bin consumption is given by

$$\begin{aligned} C_{\mathrm{LRW}} &= \sum_{k=1}^{\alpha-1} L_k + \sum_{k=\alpha}^{N-1} L_k + L_N \\ &\leq \sum_{k=1}^{\alpha-1} L_k + \sum_{k=\alpha}^{N-1} l_k + \sum_{k=\alpha}^{N-1} w_k + l_\alpha + w_\alpha. \end{aligned} \tag{3.17}$$

This relation is true whether or not $\alpha = N$. According to Lemma 3.18, the fact that $L_\alpha < 2l_\alpha$ implies that

$$L_k < 2l_k, \qquad k = 1, 2, \ldots, \alpha. \tag{3.18}$$

From Corollary 3.5, we obtain

$$\sum_{k=\alpha}^{N-1} w_k < \sum_{k=\alpha+1}^{N} l_k \tag{3.19}$$

and

$$w_\alpha < l_N. \tag{3.20}$$

The last relation holds whether or not $\alpha = N$, since $L_\alpha < 2l_\alpha$, or equivalently $w_\alpha < l_\alpha$. Relations (3.17), (3.18), (3.19) and (3.20) give

$$\begin{aligned} C_{\mathrm{LRW}} &< 2\sum_{k=1}^{\alpha-1} l_k + \sum_{k=\alpha}^{N-1} l_k + \sum_{k=\alpha+1}^{N} l_k + l_\alpha + l_N \\ &= 2\Lambda \leq 2C^*. \end{aligned}$$

∎

LEMMA 3.21. *In the solution given by LRW, if $L_N \geq 2l_N$, then $C^* \geq L_N$.*

Proof. Assume that $L_N > C^*$. We would have

$$L_N > C^* \geq \Lambda.$$

Let τ be the type of bin into which object v_N is packed in an optimal solution. This gives $S_\tau \geq s_{v_N}$ and $S_\tau \leq C^*$ by definition. Considering the assumption that $C^* < L_N$, we have $S_\tau < L_N$. According to Lemma 3.8, we have

$$S_\tau < 2\delta(\tau, \mathcal{L}_N). \tag{3.21}$$

On the other hand, from the assumption that $L_N \geq 2l_N$ and the principle of LRW, we must have

$$2 \leq L_N/l_N \leq S_\tau/\delta(\tau, \mathcal{L}_N)$$

which is in contradiction with (3.21). □

From now on, we derive the upper bounds of subsets of bins.

LEMMA 3.22. *In the solution given by LRW, if $r \leq N$, we have $\sum_{k=r}^{N-1} L_k < Q_r$.*

Proof. The lemma is obviously true if $r = N$. If $r < N$, for any k such that $r \leq k < N$, according to Lemma 3.18, we have $L_k < l_{k+1}$. Therefore, $\sum_{k=r}^{N-1} L_k < \sum_{k=r+1}^{N} l_k = Q_{r+1} < Q_r$. □

LEMMA 3.23. *In the solution given by LRW, we have $\sum_{k=q}^{r-1} L_k \leq 2(Q_q - Q_r)$.*

Proof. If $q = r$, we must have $\sum_{k=q}^{r-1} L_k = 0 = 2(Q_q - Q_r)$. The lemma thus is true when $q = r$.

If $q < r$, for any k such that $q \leq k < r$, we have $L_k < 2l_k$, according to the definition of r. Therefore $\sum_{k=q}^{r-1} L_k < 2\sum_{k=q}^{r-1} l_k = 2(\sum_{k=q}^{N} l_k - \sum_{k=r}^{N} l_k) = 2(Q_q - Q_r)$. The lemma also is true when $q < r$. □

LEMMA 3.24. *For any $a > 0$ and $x > 0$, we have $\ln(ax) - x/2 \leq \ln(2a) - 1$.*

Proof. Consider a function $f(x) = \ln(ax) - x/2$. For this function, we have $f'(x) = 1/x - 1/2$ and $f''(x) = -1/x^2 < 0$. Therefore, $f(x)$ is a concave function over $(0, +\infty)$. The maximum is obtained at $f'(x) = 0$, i.e., at $x = 2$. Therefore, $\ln(ax) - x/2 \leq \ln(2a) - 1$. □

LEMMA 3.25. *For any $a \in (0, 1]$ and for any $x > 0$, we have $(1 - a^x)/x \leq -\ln a$.*

Proof. Consider a function $g(x) = 1 - a^x + x \ln a$ defined over $[0, +\infty)$. For this function, we have $g'(x) = (1 - a^x) \ln a$. Due to the fact that $0 < a \leq 1$ and $x \geq 0$, we have $a^x \leq 1$ and $\ln a \leq 0$; namely, $g'(x) \leq 0$. Thus, $g(x)$ is a nonincreasing function. As a consequence, $g(x) \leq g(0) = 0$. As a result, $1 - a^x + x \ln a \leq 0$, or equivalently $(1 - a^x)/x \leq -\ln a$ for any $x > 0$. □

LEMMA 3.26. *In the solution given by LRW, if $L_\alpha \geq 2l_\alpha$, we have*

$$\sum_{k=p}^{q-1} L_k \leq (1 + \ln 2)L_N - 2Q_q. \tag{3.22}$$

Proof. The lemma is obviously true if $q = p$, since $(1+\ln 2)L_N - 2Q_q \geq L_N - 2Q_q \geq 0 = \sum_{k=p}^{q-1} L_k$. To prove the lemma when $q > p$, it is sufficient to show that

$$\sum_{k=p}^{q-1} L_k \leq \left(\ln \frac{Q_p}{Q_{q-1}} + 2 - \frac{L_N}{2Q_{q-1}}\right) L_N - 2Q_q. \tag{3.23}$$

In fact, from Lemma 3.24, $\ln(ax) - x/2 \leq \ln(2a) - 1$ for any $x > 0$ and $a > 0$. By substituting a and x with Q_p/L_N and L_N/Q_{q-1}, respectively, all the conditions are fulfilled and relation (3.23) implies (3.22). What follows shows the correctness of (3.23).

From the fact that $l_{q-1} = Q_{q-1} - Q_q$, we obtain

$$L_{q-1} = \frac{L_{q-1}(L_N/2 - Q_q)}{l_{q-1}} + \frac{L_{q-1}(Q_{q-1} - L_N/2)}{l_{q-1}}.$$

From (3.13) and notation of q, we obtain

$$L_{q-1}/l_{q-1} \leq L_N/Q_{q-1} < 2.$$

The last two relations give

$$\begin{aligned} L_{q-1} &< 2(L_N/2 - Q_q) + L_N/Q_{q-1}(Q_{q-1} - L_N/2) \\ &= \left(2 - \frac{L_N}{2Q_{q-1}}\right) L_N - 2Q_q. \end{aligned} \tag{3.24}$$

Therefore, if $p = q-1$, then $\sum_{k=p}^{q-1} L_k = L_{q-1} < (2 - \frac{L_N}{2Q_{q-1}})L_N - 2Q_q = (\ln \frac{Q_p}{Q_{q-1}} + 2 - \frac{L_N}{2Q_{q-1}})L_N - 2Q_q$; in other words, (3.23) holds when $p = q-1$.

Now consider the case where $p < q-1$. From (3.13), we have

$$l_k/L_k \geq Q_k/L_N, \quad k = p, p+1, \ldots, q-2,$$

or equivalently

$$(Q_k - Q_{k+1})/L_k \geq Q_k/L_N, \quad k = p, p+1, \ldots, q-2,$$

which implies

$$Q_{k+1} \leq (1 - L_k/L_N)Q_k, \quad k = p, p+1, \ldots, q-2.$$

As a matter of results,

$$Q_{q-1} \leq Q_p \prod_{k=p}^{q-2} (1 - L_k/L_N). \tag{3.25}$$

Since $L_\alpha \geq 2l_\alpha$, from Corollary 3.19, we obtain $1 - L_k/L_N > 0$ for any $k = p, \ldots, q-2$. We have, using (3.25) and the property that the geometric mean of positive numbers is less than or equal to the arithmetic mean of the same numbers,

$$\begin{aligned} Q_{q-1} &\leq Q_p \left[\frac{1}{q-1-p} \sum_{k=p}^{q-2} (1 - L_k/L_N)\right]^{q-1-p} \\ &= Q_p \left[1 - \frac{1}{(q-1-p)L_N} \sum_{k=p}^{q-2} L_k\right]^{q-1-p}, \end{aligned}$$

or equivalently

$$\sum_{k=p}^{q-2} L_k \leq (q-1-p)L_N \left[1 - \left(\frac{Q_{q-1}}{Q_p}\right)^{\frac{1}{q-1-p}}\right]. \tag{3.26}$$

From Lemma 3.25, we have $(1 - a^x)/x \leq -\ln a$ for any $a \in (0, 1]$ and for any $x > 0$. By substituting a and x with $\frac{Q_{q-1}}{Q_p}$ and $1/(q-1-p)$, respectively, all the conditions are fulfilled and (3.26) implies

$$\sum_{k=p}^{q-2} L_k \leq L_N \ln \frac{Q_p}{Q_{q-1}}. \tag{3.27}$$

Relation (3.23) is straightforward from (3.24) and (3.27). ∎

LEMMA 3.27. *In the solution given by LRW, if* $L_\alpha \geq 2l_\alpha$, *then*

$$\sum_{k=1}^{p-1} L_k < 2\max(0, \Lambda - L_N) + Q_r. \tag{3.28}$$

Proof. The lemma is obviously true if $p = 1$. When $p > 1$, we first show that

$$L_{p-1} < Q_r. \tag{3.29}$$

In fact, if $L_{p-1} \geq Q_r$, we would have $L_{p-1} \geq Q_r \geq Q_N = l_N$ since $L_\alpha \geq 2l_\alpha$ which implies $r \leq N$ (see Corollary 3.19). This relation would lead to $L_{p-1} \geq L_N$, according to Property 3.6, which would be in contradiction with the fact that $L_\alpha \geq 2l_\alpha$ which implies $L_{p-1} < L_N$ (see Corollary 3.19).

Therefore, if $p = 2$, we would have $\sum_{k=1}^{p-1} L_k = L_{p-1} < Q_r \leq 2\max(0, \Lambda - L_N) + Q_r$, As a matter of result, (3.28) holds if $p = 2$.

If $p > 2$, due to the fact that $p - 1 < r$, we have $L_k < 2l_k$ for any k such that $1 \leq k \leq p - 2$. Therefore,

$$\begin{aligned}\sum_{k=1}^{p-2} L_k < 2\sum_{k=1}^{p-2} l_k &= 2\left(\sum_{k=1}^{N} l_k - \sum_{k=p-1}^{N} l_k\right) \\ &= 2\left(\Lambda - Q_{p-1}\right) < 2(\Lambda - L_N) \\ &\leq 2\max(0, \Lambda - L_N). \end{aligned} \tag{3.30}$$

Relations (3.30) and (3.29) show that (3.28) is also true when $p > 2$. □

LEMMA 3.28. *In the solution given by LRW, if* $L_\alpha \geq 2l_\alpha$ *then* $C_{\mathrm{LRW}} < (2 + \ln 2)C^*$.

Proof. According to Corollary 3.19, we have $\alpha = N$, $L_N \geq 2l_N$ and $r \leq N$. According to Lemma 3.21, we have

$$L_N \leq C^*. \tag{3.31}$$

From Lemmas 3.22–3.27, the bin consumption C_{LRW} can be upper bounded as

$$\begin{aligned} C_{\mathrm{LRW}} &= \sum_{k=1}^{p-1} L_k + \sum_{k=p}^{q-1} L_k + \sum_{k=q}^{r-1} L_k + \sum_{k=r}^{N-1} L_k + L_N \\ &< 2\max(0, \Lambda - L_N) + (1 + \ln 2)L_N + L_N \\ &= \max\left\{(2 + \ln 2)L_N, 2\Lambda + L_N \ln 2\right\}. \end{aligned}$$

By definition, $C^* \geq \Lambda$. From this relation and (3.31), we can deduce

$$\begin{aligned} C_{\mathrm{LRW}} &< \max\left\{(2 + \ln 2)L_N, 2\Lambda + L_N \ln 2\right\} \\ &\leq (2 + \ln 2)C^*. \end{aligned}$$ □

The following theorem is straightforward from Lemmas 3.20 and 3.28.

THEOREM 3.29. *For any instance,* $C_{\mathrm{LRW}}/C^* < 2 + \ln 2$.

THEOREM 3.30. *The bound* $(2+\ln 2)$ *is tight for* C_{LRW}/C^*; *i.e.,* $\Omega_{\mathrm{LRW}} = 2+\ln 2$.

Proof. Consider the following instance composed of two types of bins with sizes $S_1 = 2n + 2$ and $S_2 = 2$, n being a positive integer strictly greater than 2. Let

$y(n) = \frac{\ln 2}{\ln(n^2+n+1)-\ln n^2}$ and $z = \lfloor y(n) \rfloor$. The objects to be packed are as follows: $s_1 = 2\frac{n+1}{n}$, $s_2 = (\frac{n^2+n+1}{n^2})^z$, $s_3 = (\frac{n^2+n+1}{n^2})^{z-1}$, ..., $s_{z+1} = \frac{n^2+n+1}{n^2}$, $s_{z+2} = s_{z+3} = \cdots = s_{n+z-1} = \frac{n+1}{n}$.

All objects except object 1 are strictly longer than 1 and strictly less than 2. Object 1 is strictly longer than 2. In the solution resulting from algorithm LRW, $n + z - 2$ bins of type 2 are used to pack objects 2 to $n + z - 1$, each containing exactly one object. Object 1 is packed into one bin of type 1. This gives rise to a bin consumption $C_{\mathrm{LRW}} = 2(n + z - 2) + (2n + 2) = 2z + 4n - 2$. The optimal packing consists of packing all objects into one bin of type 1. Then $C^* = 2n + 2$. Therefore,

$$\frac{C_{\mathrm{LRW}}}{C^*} = \frac{2n-1}{n+1} + \frac{z}{n+1}.$$

According to the definition of z,

$$y(n) - 1 < z \leq y(n).$$

Then,

$$\frac{2n-2}{n+1} + \frac{y(n)}{n+1} < \frac{C_{\mathrm{LRW}}}{C^*} \leq \frac{2n-1}{n+1} + \frac{y(n)}{n+1}.$$

When n tends towards $+\infty$, both $\frac{2n-2}{n+1}$ and $\frac{2n-1}{n+1}$ tend towards 2, and $\frac{y(n)}{n+1}$ tends towards $\ln 2$. Therefore, the ratio $\frac{C_{\mathrm{LRW}}}{C^*}$ tends towards $2 + \ln 2$. □

3.4. Additional topic. Let LW be the name of the algorithm which consists of selecting the best one among the solutions found by the four algorithms described and analyzed previously. By definition, it is obvious that

$$\Omega_{\mathrm{LW}} \leq 2.$$

On the other hand, there is an instance such that C_{LW}/C^* tends towards $1 + \ln 2$. Therefore, we can claim that

$$1 + \ln 2 \leq \Omega_{\mathrm{LW}} \leq 2.$$

This instance is quite similar to the one in the previous section. Two types of bins are available, respectively, with sizes $S_1 = 2n$ and $S_2 = 2$, n being a positive integer strictly greater than 2. The objects to be packed are as follows: $s_1 = (\frac{n^2+n+1}{n^2})^z$, $s_2 = (\frac{n^2+n+1}{n^2})^{z-1}$, ..., $s_z = \frac{n^2+n+1}{n^2}$, $s_{z+1} = s_{z+2} = \cdots = s_{n+z-2} = \frac{n+1}{n}$.

Proof. All objects are strictly longer than 1 and strictly less than 2. In the solution resulting from algorithm LW (all four algorithms yield the same solution), $n + z - 2$ bins of type 2 are used to pack objects 1 to $n + z - 2$, each containing exactly one object. This gives rise to a bin consumption $C_{\mathrm{LW}} = 2(n + z - 2)$. The optimal packing consists of packing all objects into one bin of type 1. Then $C^* = 2n$. Therefore,

$$\frac{C_{\mathrm{LW}}}{C^*} = \frac{n-2}{n} + \frac{z}{n}.$$

Then,

$$\frac{n-3}{n} + \frac{y(n)}{n} < \frac{C_{\mathrm{LW}}}{C^*} \leq \frac{n-2}{n} + \frac{y(n)}{n}.$$

When n tends towards $+\infty$, both $\frac{n-3}{n}$ and $\frac{n-2}{n}$ tend towards 1, and $\frac{y(n)}{n}$ tends towards $\ln 2$. Therefore, the ratio $\frac{C_{\mathrm{LW}}}{C^*}$ tends towards $1 + \ln 2$. □

4. Conclusion. Four greedy approximation algorithms based on a construction approach have been proposed to pack a given set of objects into bins of different sizes. The worst-case performance has been determined for each approximation algorithm. Current investigations aim at finding worst-case performance bounds for LW and for more sophisticated approximation algorithms developed for a real life cutting stock problem.

Acknowledgment. The authors are grateful to one of the referees for making us aware of some references.

REFERENCES

[1] J. Antonio, F. Chauvet, C. Chu, and J. M. Proth, *The cutting stock problem with mixed objectives: Two heuristics based on dynamic programming*, European J. Oper. Res., 114 (1999), pp. 395–402.

[2] B. Baker, *A new proof of the first-fit decreasing bin packing algorithm*, J. Algorithms, 44 (1985), pp. 49–70.

[3] C. Chu and J. Antonio, *Approximation algorithms to solve real-life multi-criteria cutting stock problems*, Oper. Res., 47 (1999), pp. 495–508.

[4] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Approximation algorithms for bin packing: An updated survey*, in Algorithm Design for Computer System Design, G. Ausiello, M. Lucertini, and P. Serafini, eds., Springer, Berlin, 1983, pp. 49–106.

[5] E. G. Coffman, Jr., S. Halfin, A. Jean-Marie, and P. Robert, *Stochastic analysis of a slotted FIFO communication channel*, IEEE Trans. Inform. Theory, 39 (1993), pp. 1555–1566.

[6] E. G. Coffman, Jr., M. R. Garey, and D. S. Johnson, *Bin packing approximation algorithms: A survey*, in Approximation Algorithms for NP-Hard Problems, D. Hochbaum, ed., PWS, Boston, MA, 1996, pp. 46–93.

[7] E. G. Coffman, Jr., D. S. Johnson, P. W. Shor, and R. R. Weber, *Bin packing with discrete item sizes, part* II*: Tight bounds on first fit*, Random Structures Algorithms, 10 (1997), pp. 69–101.

[8] J. Csirik, *An on-line algorithm for variable-sized bin packing*, Acta Inform., 26 (1989), pp. 697–709.

[9] D. K. Friesen and M. A. Langston, *Variable sized bin packing*, SIAM J. Comput., 15 (1986), pp. 222–230.

[10] M. Garey, R. Graham, D. Johnson, and A. Yao, *Resource constrained scheduling as generalized bin packing*, J. Combin. Theory Ser. A, 21 (1985), pp. 257–298.

[11] A. Ivanyi, *Tight Worst-Case Bounds for Bin Packing Algorithms*, Colloq. Math. Soc. János Bolyai 44, 1984, North-Holland, Amsterdam, pp. 233–240.

[12] D. S. Johnson, *Fast algorithms for bin packing*, J. Comput. System Sci., 8 (1974), pp. 272–315.

[13] C. C. Lee and D. T. Lee, *A simple on-line bin packing algorithm*, J. Assoc. Comput. Mach., 32 (1985), pp. 562–572.

[14] W. Mao, *Best-k-fit bin packing*, Computing, 50 (1993), pp. 265–270.

[15] W. Mao, *Tight worst-case performance bounds for next-k-fit bin packing*, SIAM J. Comput., 22 (1993), pp. 46–56.

[16] F. D. Murgolo, *An efficient approximation scheme for variable-sized bin packing*, SIAM J. Comput, 16 (1987), pp. 149–161.

[17] D. Simchi-Levi, *New worst case results for the bin packing problem*, Naval Res. Logist., 41 (1994), pp. 597–585.

[18] G. Zhang, *Worst-case analysis of the FFH algorithm for on-line variable-sized bin packing*, Computing, 56 (1996), pp. 165–172.

SIAM J. COMPUT.
Vol. 30, No. 6, pp. 2084–2096

FIRST-ORDER SPECIFICATIONS OF PROGRAMMABLE DATA TYPES*

GRAŻYNA MIRKOWSKA†, ANDRZEJ SALWICKI†, MARIAN SREBRNY‡, AND ANDRZEJ TARLECKI§

Abstract. We consider first-order specifications together with the restriction to accept only programmable algebras as models. We provide a criterion which links this approach with the "generation principle": all programmable models of any specification SP that meets this criterion are reachable. We also show an example of a specification which does not satisfy the criterion and admits a programmable yet nonreachable model. Moreover, a general method of showing the existence of programmable but nonreachable models for a class of first-order specifications is given.

Key words. algebraic specification, reachable algebra, data types

AMS subject classifications. 68Q60, 68Q65, 68P05

PII. S0097539797322528

1. Introduction. The problem of characterizing intended algebras is important in various areas of computer science. In the areas of specification, development, and validation of software systems one needs to describe the intended models—typically "no junk" algebras—modelling (representing) possible or actual implementations of software systems being specified.

In this paper we make use of intrinsic conceptual parallelism between software engineering and metamathematics. Some software modules—classes, libraries, packets, etc.—can be conveniently viewed as algebraic structures. The goal of specifying software modules then finds its formal counterpart: specification of algebraic structures. In the terminology of computer scientists one sometimes also says specification of data types. A formal specification may (and should) serve as the only source of information on a data structure $\mathcal{A}$ when one constructs and/or analyzes a program P which uses the data structure $\mathcal{A}$. Suppose that such a data structure $\mathcal{A}$ is described by a specification SP. Is the specification SP sufficiently rich in order to ensure that any proof of a valid semantical property of the program P (such as correctness, termination, etc.) can rely only on the axioms listed in SP? Or should one add some extra constraints—or further information—on data structure $\mathcal{A}$ beyond those contained in the specification SP?

Typically, the answer to the latter question is affirmative: first-order logic is not strong enough to exclude all nonintended models of a specification, and hence nonintended implementations of the specified module. Similar problems arise also with specifications intended to be loose though not "too loose."

Various authors introduce some special strong additional requirements to cope with this problem. For instance, the category-theoretic initial (also, terminal) alge-

*Received by the editors June 10, 1997; accepted for publication (in revised form) August 23, 2000; published electronically May 16, 2001.
http://www.siam.org/journals/sicomp/30-6/32252.html

†LITA, Université de Pau, France, and Institute of Informatics, Białystok University of Technology, Białystok, Poland (mirkowska@pjwstk.waw.pl, salwicki@mimuw.edu.pl).

‡Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland (marians@ipipan.waw.pl). The work of the third author was partially supported by Poland's Scientific Research Committee under grant 2 P301 007 04.

§Institute of Informatics, Warsaw University, and Institute of Computer Science PAS, Warsaw, Poland (tarlecki@mimuw.edu.pl).

bra semantics approach has attracted a lot of attention. In these approaches, one selects as the semantics of a specification only initial (respectively, terminal) objects in the category of all algebras that satisfy given specification axioms; see, e.g., [4], [2], [18]. Another option is to require the algebras to be reachable (generated by the empty set, or standard—see the "generation principle" of [1]). Reachable algebras cannot be characterized by any set of first-order axioms, since every reachable algebra has an elementary extension that is not reachable, by direct application of the Skolem–Löwenheim theorem. None of these extra requirements (initiality, terminality, reachability) is expressible in first-order logic or its fragments (e.g., equational logic) often used in algebraic specifications. They can, however, be expressed in some stronger logics, for instance, in various versions of algorithmic logic (see [12]) or infinitary logics with proof systems involving infinitary proof rules (e.g., a version of the ω-rule) or in the second-order logic with the semantics where second-order variables range either over all subsets of the universe or only over finite subsets (weak second-order logic). Nevertheless, the classical first-order logic is of special interest here, at least because of its familiarity and since most of the contemporary mathematics has been done in it.

There are two possible ways to proceed: either we can restrict the class of models considered to some special structures only, characterized by some extralogical means, or we can work with a stronger logic. In this paper we promote and advocate the first possibility. However, we propose a "new" natural requirement of programmability (computability) of the models as the extralogical selection criterion. This seems more natural and friendly to programmers and to programming environment than other approaches proposed so far. By the Church thesis all possible implementations of any specification are programmable. Therefore, selecting programmable algebras as the only models considered is kind of a "minimal" requirement. It is perhaps surprising that this natural possibility has not been properly exploited in the theory and practice of algebraic specifications. The surprise is even more evident in light of a whole line of interesting results by Bergstra and Tucker [2] on the interplay between computability and equational specifications and on the expressive power of the initial algebra specifications.

To reiterate, we propose the classical first-order logic with the programmable (recursive) algebra semantics to be used for algebraic specifications.

It should be emphasized here with no need of any further details that the programmability requirement is natural and comprehensible to potential software developers, specifiers, and validators, as perhaps opposed to the abstract category-theoretic initiality or terminality criteria. A serious demand for detailed study of this kind of requirements follows, for instance, from a work of Sannella and Tarlecki [14] on a "gap" between the usual viewpoint of algebraic specification based on arbitrary models and that of programming frameworks (such as Extended ML [8]) more realistically based on programmable models. Traditionally in the area, abstract many-sorted algebras are used to model programs: the representation of data is arbitrary and operations are modelled as functions. Formal development of software systems from specifications calls for restriction of these general concepts to the kind of models that underlie semantics of programming languages. For example, the semantics of Standard ML [10] uses rather concrete models, where data values are represented as closed constructor terms and operations are represented as "closures."

We believe that the programmable reachable algebras provide an interesting general framework for algebraic specifications. In this note we present some technical

results to justify this belief and to start a proper investigation of such a framework.

2. Terminology and notation. In this paper we consider specifications as classical first-order (or *elementary*) theories. Unlike in the classical model-theory, we work with many-sorted algebras; a many-sorted algebra consists of a carrier set split into a family of nonempty carriers named by sorts with (a finite number of) operations on them. The carriers and operations of an algebra are named in the algebra's signature. Zero-ary operations are just the distinguished elements, interpreting constants of the language according to the algebra's signature. In general, the operations may be partial functions on their sorts. Throughout the paper we work with first-order logic with equality, i.e., formulas include equalities between terms of the same sort (for each sort) interpreted as the identity of the denotations of the terms. Occasionally we simplify the exposition by leaving the reader with many details of rigorous "sorting" of the formal expressions whose content is sketched only informally.

DEFINITION 2.1. *An algebra (in general, a relational structure) is called* programmable *(or* recursive*, or* computable*) whenever all of its carriers are computable sets and all of its operations (in the general case, relations) are computable functions defined on computable domains.*

We will be talking only about countable algebras. Therefore we can always consider beforehand an isomorphic copy of a given algebra built on the natural numbers and then define the algebra to be recursive whenever this copy is recursive. For example, the natural numbers with the usual arithmetic operations form a programmable algebra $(N; 0, suc, +, \times)$. Given a nonrecursive function f, the algebra $(N; 0, suc, +, \times, f)$ is not programmable.

We illustrate the ideas and results of this paper with various data types of stacks and their first-order specifications. We focus on the stacks because of their relevance for implementing recursive computations and modelling recursively defined collections of data, and because they provide a convenient example that we hope should be familiar to the reader. Stack theory can be formalized in various forms, essentially different in many ways from the metamathematical viewpoint. All of them share in their signature two sorts (one for elements and another for stacks) and the usual stack operations.

Given a finite or infinite set E, by *the true stack theory on* E we mean the first-order theory of the model with two-sorted carrier set E, E^*, where E^* is the set of all finite sequences of the members of E, and the usual operations *top*, *pop*, *push*, the distinguished *empty* sequence, and with all the elements of E as constants in the signature. In this paper we typically consider at most countably infinite E. The operations *top* and *pop* are undefined on *empty*. Here are the usual definitions for each $e \in E$ and $s = (e_0, \ldots, e_n) \in E^*$:

$$\begin{aligned} push(e, s) &= (e_0, \ldots, e_n, e), \\ top(s) &= e_n \quad \text{for nonempty } s, \\ pop(s) &= (e_0, \ldots, e_{n-1}) \quad \text{for nonempty } s. \end{aligned}$$

This yields a family of different true stack theories, one for each E. Modulo the names of the constants used, we in fact consider here one such theory for each (finite or infinite) cardinality of E. Each of them is formally a theory over a different signature (following the usual terminology in mathematical logic, we will say a *language* of a theory, rather than its signature).

In the countably infinite case, one can think of E as of the set of natural numbers. However, the members of E need not be the natural numbers, since the signature of the model does not contain any arithmetic operations.

By *the axiomatic theory of stacks on* E we mean the (classical first-order) consequences of the universal closure of the following axioms in the (first-order) language corresponding to the above model:

$$\begin{array}{ll} \text{(Ax1)} & push(x,s) \neq empty, \\ \text{(Ax2)} & top(push(x,s)) = x, \\ \text{(Ax3)} & pop(push(x,s)) = s, \\ \text{(Ax4)} & s \neq empty \rightarrow push(top(s), pop(s)) = s. \end{array}$$

This theory is a proper subtheory of the true stack theory on E. In particular, the true stack theory, but not the axiomatic stack theory, admits the following *scheme of structural induction* for every first-order formula φ:

$$[\varphi(empty) \wedge \forall x \, \forall s \, (\varphi(s) \rightarrow \varphi(push(x,s)))] \rightarrow \forall s \, \varphi(s).$$

By *the true theory of stacks on the natural numbers* we mean the first-order theory of the model $(N, N^*; 0, suc, +, \times, top, pop, push, empty)$ with the usual meaning of the arithmetic operations on the set N of natural numbers. The Peano axioms for the sort of elements and axioms (Ax1–Ax4) for stacks form yet another formalization, which we will call the *axiomatic stack theory on the natural numbers*.

Let us point out that here we mean the Peano axioms with the scheme of elementary induction

$$[\varphi(0) \wedge \forall x \, (\varphi(x) \rightarrow \varphi(x+1))] \rightarrow \forall x \, \varphi(x)$$

for every first-order formula φ. By Peano Arithmetic we mean the (classical) first-order consequences of the usual axioms for $0, suc, +, \times$ with the scheme of elementary induction.

3. Reachable algebras.

DEFINITION 3.1. *An algebra is called* reachable *if it is generated from it's underlying constants by finitely many successive applications of the algebra's operations.*

An algebra $\mathcal{A}$ is generated by a subset X of the carrier of $\mathcal{A}$ (a set of generators) if all the elements of $\mathcal{A}$ are the values of Σ-terms under some valuation of variables into X. Σ is the signature of $\mathcal{A}$. Now $\mathcal{A}$ is reachable iff it is generated by the empty set of generators.

Equivalently, an algebra is reachable if each element of its universe (carrier sets) is the denotation of a closed (ground) term in the (first-order) signature of the algebra (cf. the notion of a Herbrand model in model-theory). The reachable algebra of natural numbers, the only reachable model of Peano Arithmetic, is usually called the *standard model*. One says the reachable algebras have "no junk" property: they contain all data you can talk about in the algebra's formal language and nothing more. This does not imply, of course, another useful property of "no confusion" (or "unambiguity," or "the unique name") that different terms denote distinct data items, unless those terms are forced to be identified by the specification axioms.

Of course, reachable algebras need not be programmable in general. For instance, the nonprogrammable algebra $(N, 0, suc, f)$ with nonrecursive f mentioned above is reachable. In this paper we show that for a sufficiently rich theory its reachable

models can be distinguished in the class of programmable models by a single first-order axiom. This is based on a deep result of Tennenbaum [17] that *every recursive model of Peano Arithmetic (with the first-order induction scheme) is standard.*

Roughly, a first-order theory T is sufficiently rich in this context if Peano Arithmetic can be recursively interpreted in T and T has the finite sequence coding property. Let us recall the necessary notions of mathematical logic.

For the concept of an interpretation we refer, for instance, to Shoenfield [15]. We say that Peano Arithmetic can be interpreted in T whenever there are formulas $I_N(\cdot)$, $I_0(\cdot)$, $I_S(\cdot,\cdot)$, $I_{Add}(\cdot,\cdot,\cdot)$, and $I_{Mult}(\cdot,\cdot,\cdot)$ in the language of T such that T proves all the Peano axioms relativized to the interpretation I. An interpretation defines a model of Peano Arithmetic within every model of T: I_N defines its carrier, while the other formulas define the operations (or their graphs) of arithmetic, respectively. An interpretation is recursive if it defines a recursive model of Peano Arithmetic within every recursive model of T.

Given a theory with an interpretation of (enough of) arithmetic, one can think of sequences indexed by the natural numbers of the interpretation, i.e., by the elements of the carrier of the interpretation. Then one can talk about possible coding of initial segments of such sequences by single elements. In the following definition, intuitively, *BETA* is a uniform projection (decoding) function: for a sequence $a_0, a_1, \ldots, a_l$, there is a code u such that $BETA(u, j) = a_j$ for each $j \leq l$. Let us stress once more though that the "length" of the sequence and the "indexing subscripts" are numbers only in the sense of the interpretation of arithmetic.

DEFINITION 3.2. *We say that a theory T with an interpretation of arithmetic has* the finite sequence coding property *if there is a binary function BETA definable in T such that*

for every formula $A(x, y, parameters)$ in the language of T, if T proves
"for each x of the arithmetic interpretation universe I_N, there exists exactly one y such that $A(x, y, parameters)$,"
then T also proves the following:
"for each l of I_N, there exists a u such that for each $j \leq l$, $A(j, BETA(u, j), parameters)$."

Models of a theory with the finite sequence coding property should be compared with the concept of arithmetical universes of Harel [6]. Harel's arithmetical universes are equipped with a copy of the standard model of arithmetic, whereas we consider algebras equipped with a copy of just a model of Peano Arithmetic. The latter turns out to be standard under the assumptions of the following theorem.

THEOREM 3.3. *Let T be a first-order theory over a finite language such that*

1. *Peano Arithmetic can be recursively interpreted in T and*
2. *T has the finite sequence coding property.*

Then there is a single sentence σ_{reach} in the language of T such that every recursive model of T satisfying σ_{reach} is reachable.

Proof. To prove the above theorem we take the following sentence as σ_{reach}:

$$\forall x\, \exists u\, (u \text{ codes a path of reaching } x \text{ from the generators}).$$

More precisely, though still informally, the formula under the two quantifiers says the following: u codes a sequence of values in a given model M such that x is the last element of the sequence and each element in the sequence is either a denotation of a constant in M or can be obtained from earlier values in the sequence by application of one of the operations of M. Then consider a recursive model M of T. M contains a

recursive model of Peano Arithmetic. The latter has to be standard by Tennenbaum's result [17]. Therefore, if σ_{reach} holds in M, then M is reachable. □

Since the above sentence σ_{reach} holds in the standard model of the theory of stacks on the natural numbers (which contains Peano Arithmetic and clearly has the sequence coding property) the above theorem gives us the following corollary.

COROLLARY 3.4. *Every recursive model of the true theory of stacks on natural numbers is reachable.*

It now follows easily that every programmable model of the true stack theory on natural numbers is isomorphic to $(N, N^*; 0, suc, +, \times; top, pop, push, empty)$. Can anyone supply a better, while so friendly, specification of this data type?

The message can now be put forward as follows. For any sufficiently rich specification, if one restricts its semantics to programmable models only, then in order to ensure reachability, which in many cases means uniqueness, one simply needs to throw *implicitly* or *explicitly* the appropriate σ_{reach} into the specification. Given a sufficiently rich specification, in order to verify that all its programmable models are reachable it is necessary and sufficient to check whether the appropriate σ_{reach} is derivable from this specification. We have just applied this result to the true stack theory on natural numbers.

4. Nonreachable programmable algebras. Theorem 3.3 indicates that computability of a model of a sufficiently rich specification implies its reachability. In this section we provide examples of specifications (theories) that are not rich enough and admit recursive nonreachable models.

As an easy warm-up example consider the usual specification axioms for successor:

$$succ(x) \neq 0,$$
$$succ(x) = succ(y) \rightarrow x = y.$$

Any linear order of type $\omega + (\omega^* + \omega)$ provides a model of these axioms in the obvious way (with no claim of induction). Clearly, this model is recursive but nonreachable.

There are several ways of constructing a recursive nonreachable model of the true stack theory (as well as of its axiomatic mutations) on a finite set of elements. The present research has evolved from a definition of such a model written as a program in [11].

Here we give details of one possible construction of a recursive nonreachable model of the axiomatic stack theory on a finite E.

Let us first recall the Gödel's recursive function β such that for any natural numbers $a_1, \ldots, a_n$, there exists an $a \in N$ with $\beta(a, i) = a_i$, for each $i = 1, \ldots, n$, and with $\beta(a, 0) = n$. Moreover, $\beta(a, i) \leq a \dot{-} 1$. One can introduce the following sequence coding function:

$$SC(a_1, \ldots, a_n) \quad \stackrel{df}{=} \quad \mu x\, (\beta(x, 0) = n \wedge \beta(x, 1) = a_1 \wedge \cdots \wedge \beta(x, n) = a_n).$$

Define also the length function $lh(a) = \beta(a, 0)$, and the projection functions $(a)_i = \beta(a, i)$, for $i = 1, \ldots, lh(a)$. For $n = 0$, $\beta(0, i) = 0$, and so for the empty sequence ε we have $SC(\varepsilon) = 0$. We also need

$$Seq(a) \quad \stackrel{df}{\equiv} \quad a \textit{ is a code of a sequence of length } lh(a).$$

The reader is referred to [15] for further details.

Now we define the model. Let $\{0,1\}$ be its carrier set of the element sort. The reader can easily modify this construction to any finite set of elements. Let its stack carrier be the union $S \cup S_0 \cup S_1$ of the following three sets of 0–1 sequences (finite or infinite):

- S is the set of all the finite 0–1 sequences;
- S_0 is the set of all the infinite 0–1 sequences stabilizing on 0;
- S_1 is the set of all the infinite 0–1 sequences stabilizing on 1.

Their union is recursive since they can be viewed as copies of the following sets of finite 0–1 sequences preceded by one of the markers 0, 1, or 2 and by their lengths. We use 0 as a marker for S_0, 1 for S_1, and 2 for S. Look at S_0 as the set of all finite sequences with 1 at the end, followed by infinitely many 0's. Allow the empty sequence there too—although it has no 1 at the end, it is needed to cover the case of the infinite sequence consisting only of 0's. Such infinite sequences can be coded up by pairs $\langle 0, a\rangle$ with sequence numbers a. The length $lh(a)$ indicates the position of the last 1. $lh(a) = 0$ indicates that there is no 1 at all. Similarly, deal with S_1 by indicating the position of the last 0 or indicating there is no 0 at all. Formally:

$$S \cup S_0 \cup S_1 = \{\langle \delta, a\rangle \mid \delta = 0,1,2 \wedge Seq(a) \wedge \forall 1 \le i \le lh(a)\,((a)_i = 0 \vee (a)_i = 1) \\ \wedge\ (lh(a) > 0 \to (a)_{lh(a)} \neq \delta)\}.$$

Define $empty = \langle 2, SC(\varepsilon)\rangle$, and for each $\delta = 0,1,2$ and each sequence code a of $(a_1, a_2, \ldots, a_{lh(a)})$ define the operations

- $top(\langle \delta, a\rangle) = \begin{cases} a_1 & \text{if } lh(a) > 0, \\ 0 & \text{if } lh(a) = 0 \text{ and } \delta = 0, \\ 1 & \text{if } lh(a) = 0 \text{ and } \delta = 1, \\ \text{undefined} & \text{otherwise.} \end{cases}$

- $pop(\langle \delta, a\rangle) = \begin{cases} \langle \delta, SC(a_2, \ldots, a_{lh(a)})\rangle & \text{if } lh(a) > 0, \\ \langle \delta, a\rangle & \text{if } lh(a) = 0 \text{ and } \delta \neq 2, \\ \text{undefined} & \text{otherwise.} \end{cases}$

- $push(e, \langle \delta, a\rangle) = \begin{cases} \langle \delta, SC(e, a_1, \ldots, a_{lh(a)})\rangle & \text{if } lh(a) > 0 \text{ or } \delta \neq e, \\ \langle \delta, a\rangle & \text{if } lh(a) = 0 \text{ and } \delta = e. \end{cases}$

LEMMA 4.1. *$(\{0,1\}, S \cup S_0 \cup S_1; top, pop, push, empty, 0, 1)$ is a recursive nonreachable model of the axiomatic stack theory.*

Proof. The proof has an easy verification. □

5. Nonreachable programmable models of decidable theories. In this section we show that there exists a nonreachable recursive model of the true stack theory on any given finite set E. We get it as an application of a general method (Corollary 5.4). The argument below relies heavily on the following theorem (known in the folklore of the field).

THEOREM 5.1 (the computable model existence theorem). *Every consistent decidable theory has a recursive model.*

Proof. One can "effectivize" Henkin's proof of the completeness theorem for the classical first-order logic. Some details are given in [7] and related results in [9]—for the reader's convenience we sketch a full proof in the appendix. The key step is an effective version of the celebrated Lindenbaum–Tarski theorem: every decidable consistent theory has a decidable complete extension in the same language. □

DEFINITION 5.2. *A theory is* reachably unambiguous *if all its reachable models are elementarily equivalent.*

It is easy to see that many interesting and useful theories are reachably unambiguous, e.g., Peano Arithmetic and Presburger Arithmetic. This is also the case for the stack theory in any of the above formalizations. In all these cases the theory has just one reachable model up to an isomorphism.

COROLLARY 5.3. *If T is a reachably unambiguous, incomplete, and decidable theory, then T has a recursive nonreachable model.*

Proof. In the algorithm of completion of T following Tarski [16], one can keep control on which of either σ or its negation $\neg\sigma$ gets into the completion, for a sentence σ such that neither $T \vdash \sigma$ nor $T \vdash \neg\sigma$. Since T is incomplete and reachably unambiguous, there is a sentence σ_0 such that neither $T \vdash \sigma_0$ nor $T \vdash \neg\sigma_0$, although σ_0 holds in every reachable model of T. Thus, a recursive model of the completion of T with $\neg\sigma_0$, built as in the proof of Theorem 5.1 above, cannot be reachable. □

COROLLARY 5.4. *Suppose T is a first-order theory. Let $c_0, c_1, c_2, \ldots$ be all the closed terms of the language of T, and let c be a new individual constant. If $T \cup \{\, c \neq c_n \mid n \text{ natural}\}$ is a consistent decidable theory, then T has a recursive nonreachable model.*

Proof. The proof is obvious by Theorem 5.1. □

We establish some results concerning decidability of the true stack theory in order to apply Corollary 5.4.

THEOREM 5.5. *Given a finite set E, the true stack theory on E is decidable.*

Proof. We show an interpretation of the true stack theory on E in Presburger Arithmetic (the first-order theory of $(N, suc, +, 0)$) which is well known to be decidable; see [13]. In fact, we show a somewhat stronger result by defining in $(N, suc, +, 0)$ an isomorphic copy of the model $(E, E^*; top, pop, push, empty, \{e\}_{e\in E})$.

Let k be the cardinality of E. We define the two-sorted carrier of our interpretation and its operations as follows with the usual notation for operations and relations definable in Presburger Arithmetic. ($\underline{k}$ denotes the kth numeral, i.e., $\underline{k}$ abbreviates $\underbrace{suc(\ldots suc(}_{k \text{ times}}\underline{0})\ldots)$).

- $U_E^I(e) \stackrel{df}{\equiv} e = \underline{0} \vee e = suc(\underline{0}) \vee \cdots \vee e = \underline{k-1}$;
- $U_S^I(s) \stackrel{df}{\equiv} s \geq \underline{k}$;
- $empty^I \stackrel{df}{=} \underline{k}$;
- $push^I(e, s) \stackrel{df}{=} \underline{k} \times (s - \underline{k}) + e + \underline{k} + 1$;
- $top^I(s) \stackrel{df}{=} (s - \underline{k} - 1) \bmod \underline{k}$, for $s > \underline{k}$;
- $pop^I(s) \stackrel{df}{=} [(s - \underline{k} - 1) \operatorname{div} \underline{k}] + \underline{k}$, for $s > \underline{k}$.

The idea behind this construction is that the stacks are coded up by numbers in their k-adic expansions, shifted up by adding $k + 1$ to make disjoint room for the elements and for the empty stack. The top^I operation returns the last digit of the expansion which is just the remainder of dividing by k. Similarly, pop^I cuts off the last digit. This is legal here since division by a given numeral is expressible in Presburger Arithmetic. Consequently, the decidability procedure for Presburger Arithmetic does the job for the true stack theory on E. □

The above proof gives immediately the following complexity results inherited from Presburger Arithmetic. The double exponential upper bound is due to Ferrante and Rackoff [3]. There exists a decision procedure and a constant c such that it takes

at most $2^{2^{cn}}$ deterministic single-tape Turing machine space to decide a sentence of length n. This gives deterministic time upper bound $2^{2^{2^{dn}}}$; see also Fischer and Rabin [5]. Precise complexity bounds for the true theory of stacks, which we have established recently beyond the scope of the present paper, will appear elsewhere.

LEMMA 5.6. *Let TST denote the true stack theory on a given finite set E. Then the theory $TST \cup \{c \neq c_n \mid n \text{ natural}\}$ is decidable, where $c_0, c_1, c_2, \ldots$ are all the closed terms of the stack sort of the language of TST, and c is a new individual constant of the stack sort.*

Proof. We reduce the decidability problem for this theory to the decidability of Presburger Arithmetic (denoted by PrA below). Let T denote the extended theory $TST \cup \{c \neq c_n \mid n \text{ natural}\}$. By a reduction we mean an effective mapping f associating to each sentence σ of the language $L(T)$ a sentence $f(\sigma)$ of the language $L(PrA)$ such that $T \vdash \sigma$ iff $PrA \vdash f(\sigma)$. Thus a decision procedure whether σ is a theorem of T will be reduced to whether $f(\sigma)$ is a theorem of Presburger Arithmetic. This will imply the decidability of T since Presburger Arithmetic is decidable.

For each σ of $L(T)$, we define $f(\sigma) = \exists y \forall x[x > y \longrightarrow I\sigma(x/c)]$, where $I\sigma$ is the interpretation of σ in Presburger Arithmetic as introduced in the proof of Theorem 5.5, separately for each E, and $I\sigma(x/c)$ is the substitution of x for c in $I\sigma$. We show that f is a reduction, i.e., $T \vdash \sigma$ iff $PrA \vdash f(\sigma)$.

($\Longrightarrow$) Suppose $T \vdash \sigma$. At most finitely many axioms of the form $c \neq c_n$ may appear in a proof of σ in T. Let l be the greatest number (subscript) of a constant c_n appearing in this proof. Then $TST \cup \{c \neq c_0, c \neq c_1, c \neq c_2, \ldots, c \neq c_l\} \vdash \sigma$. Thus

$$TST \vdash [c \neq c_0 \wedge c \neq c_1 \wedge c \neq c_2 \wedge \cdots \wedge c \neq c_l \rightarrow \sigma].$$

Since c is a new constant, there are no axioms concerning c in TST. Therefore

$$TST \vdash \forall x\,[x \neq c_0 \wedge x \neq c_1 \wedge x \neq c_2 \wedge \cdots \wedge x \neq c_l \rightarrow \sigma(x/c)],$$

and so

$$PrA \vdash \forall x\,[x \neq Ic_0 \wedge x \neq Ic_1 \wedge x \neq Ic_2 \wedge \cdots \wedge x \neq Ic_l \rightarrow I\sigma(x/c)].$$

In the standard model of Presburger Arithmetic the latter implies $\forall x[x > c_m \longrightarrow I\sigma(x/c)]$, where c_m is the greatest of Ic_0, Ic_1, Ic_2,..., Ic_l. Here by Ic_i we denote the interpretation of the constant term c_i. Hence $\exists y \forall x\,[x > y \rightarrow I\sigma(x/c)]$ holds in the standard model of Presburger Arithmetic. By completeness of Presburger Arithmetic we get $PrA \vdash \exists y \forall x\,[x > y \rightarrow I\sigma(x/c)]$.

($\Longleftarrow$) Suppose $PrA \vdash \exists y \forall x\,[x > y \rightarrow I\sigma(x/c)]$. Let m be such a number that $\forall x\,[x > m \rightarrow I\sigma(x/c)]$ holds in the standard model of PrA. Hence,

$$PrA \vdash \forall x\,[x \geq \underline{k} \wedge x \neq Ic_0 \wedge x \neq Ic_1 \wedge x \neq Ic_2 \wedge \cdots \wedge x \neq Ic_l \rightarrow I\sigma(x/c)],$$

where k is the cardinality of E and l is large enough for all the numerals from $\underline{k}$ to $\underline{m}$ to occur among Ic_0, Ic_1, Ic_2,..., Ic_l. Therefore

$$PrA \vdash [c \geq \underline{k} \wedge c \neq Ic_0 \wedge c \neq Ic_1 \wedge c \neq Ic_2 \wedge \cdots \wedge c \neq Ic_l \rightarrow I\sigma(c)],$$

where c is a new constant. Hence

$$TST \ \cup \ \{c \neq c_0 \wedge c \neq c_1 \wedge c \neq c_2 \wedge \cdots \wedge c \neq c_l\} \vdash \sigma(c)$$

for a new constant c of the stack sort. We get $T \vdash \sigma(c)$, since T contains

$$TST \cup \{c \neq c_0 \wedge c \neq c_1 \wedge c \neq c_2 \wedge \cdots \wedge c \neq c_l\}. \qquad \square$$

COROLLARY 5.7. *The true stack theory on a given finite set E has a recursive nonreachable model.*

Proof. The proof follows from Lemma 5.6 and Corollary 5.4. $\square$

COROLLARY 5.8. *The axiomatic stack theory on any finite E with the (first-order) scheme of structural induction has a recursive nonreachable model.*

Proof. It follows immediately from the above, since this theory is contained in the true stack theory on E. $\square$

REMARK 5.9. *Presburger Arithmetic has a recursive nonreachable model.*

Proof. It follows from Corollary 5.4 by a similar argument as in the proof of Lemma 5.6. This has been known to the experts although never published. $\square$

6. Conclusion. In general a first-order specification (axiomatization) has too many models. There have been several approaches in order to restrict the semantics of specifications to intended models only. The initial algebra approach proposed by [4] in the mid 1970s has particularly attracted a lot of attention. In this paper we propose and advocate the programmable algebra approach, i.e., to consider the programmable algebras as the only models. That is, by a specification we mean a pair (Σ, T), where Σ is a signature and T is a list of axioms in the classical first-order logic (with equality) of signature Σ, and the models of such a specification under the proposed semantics are all computable Σ-algebras that satisfy the axioms T. Furthermore,

- we give a criterion assuring that any specification which satisfies the criterion has only reachable models. Often this means just one model up to isomorphism. This is the case of the axiomatic theory of stacks on natural numbers.
- We give some examples of specifications that do not satisfy the criterion and admit programmable models which are not reachable. This is the case of both axiomatic and true stack theory on any given finite set E.
- We give a rather powerful method of constructing programmable nonreachable models.

These results can be easily generalized to provide a criterion that ensures that computable models of a theory are generated by a set of generators whenever this set is definable by a first-order formula. For instance, we may always indicate a subset of sorts and consider the carriers of these sorts as such a definable set of generators. This slightly more general version would be needed, for instance, to deal with stack theories on infinite sets of elements, which then can be considered over a finite signature, without constants for elements.

We also show decidability of the true stack theory on a finite set of elements, a result of interest on its own. One can use it, for instance, to conclude that Peano Arithmetic is not interpretable in the theory of stacks on a finite set of elements.

We do not know if the axiomatic stack theory is decidable. It remains as an intriguing open question. We conjecture that given a set E, the true stack theory on E is contained in the axiomatic stack theory on E with the scheme of structural induction. (Since the other inclusion is obvious, this would mean that the true and axiomatic theories are equivalent in this case.)

7. Appendix. In this appendix we provide a proof of Theorem 5.1: every decidable consistent theory has a recursive (computable) model. In view of this paper, it

seems to be of considerable revitalized interest for use in the area of algebraic specification of software systems, their semantics, validation, and systematic development. It provides a source of supply of programmable algebras to various purposes. The result has been in the folklore for many years. Some variant of it was stated in [9] without proof and another one (a bit stronger than our formulation) in [7] with a sketch of the proof idea.

As throughout this paper, we assume the reader is familiar with the usual formalization of classical first-order logic and its semantics; see, for example, [15]. All the theories considered will be formalized in classical first-order logic with countably many symbols. For simplicity we give the result and our proof here for the single-sorted case.

Recall that a theory T is called *decidable* if T is a recursive set (of Gödel numbers) of sentences, i.e., if there is an effective procedure deciding in a finite number of steps whether a given sentence is a theorem of T or not. Let us recall also that a theory T is *complete* if for every sentence σ exactly one of the following holds: either $\sigma \in T$ or $\neg\sigma \in T$. For two theories T and T', we say that T' is an extension of T if T' contains T. By $L(T)$ we denote the language of T.

To prove the theorem we follow Henkin's proof of the completeness theorem for classical first-order logic with appropriate modifications. Let T be a consistent decidable theory. We extend T to another theory T' such that T' is decidable, complete, and Skolemized. The latter means it has a term witness for each existential statement derivable in this theory. The *canonical structure* will be constructed on the terms of the language of T'. Its interpretations of relation and function symbols will be defined according to what T' knows about them. We then prove the canonical structure is a model of T'. Thus it is a model of T too, since T is a subtheory of T'. The model turns out recursive because T' is a decidable theory. That is, the construction of the model is described recursively by the story of T'. To this end we need the following two lemmas. The first of them can be thought of as an effectivized Lindenbaum theorem.

LEMMA 7.1. *Every decidable consistent theory has a complete decidable and consistent extension in the same language.*

Sketch of proof (Lindenbaum–Tarski). Let T be a decidable consistent theory. Let $\varphi_0, \varphi_1, \varphi_2, \ldots$ be a recursive enumeration of all sentences of the language of T. Let T_0 be T itself. Given decidable theory T_n, define k_n as the least k such that $\neg\varphi_k$ is not a theorem of T_n. Then let T_{n+1} be the theory that results from adding φ_{k_n} to T_n as a new axiom. T_{n+1} is decidable, since T_{n+1} proves σ iff T_n proves $\varphi_{k_n} \to \sigma$. Now let T' be the theory which has as its axioms all of the axioms of these theories T_n, $n \geq 0$. One can show that T' is consistent, complete, and decidable. □

LEMMA 7.2 (Skolemization). *If T is a consistent decidable theory, then there is a consistent decidable extension T' of T with the language $L(T')$ containing $L(T)$ and such that if a sentence $\exists x\varphi(x)$ is a theorem of T, then there is a term t in $L(T')$ with $\varphi(t)$ belonging to T'.*

Proof. Let T be a consistent decidable theory. We construct by recursion a sequence of languages L_0, L_1, L_2, ... and a sequence of theories T_0, T_1, T_2, ... At each step we extend the language with a new constant and adjoin one new axiom to the theory. First, let us enumerate recursively all the existential sentences of $L(T)$: $\exists x\varphi_n(x)$, n natural. We take off with $L_0 = L(T)$ and $T_0 = T$. Now suppose we are given a consistent decidable theory T_n in $L(T_n)$. If $\exists x\varphi_n(x)$ is derivable in T_n, then we take L_{n+1} to be L_n plus a new constant c_n, and T_{n+1} to result from T_n by adding $\varphi_n(c_n)$ as an axiom. Otherwise, we take $L_{n+1} = L_n$ and $T_{n+1} = T_n$. Clearly, new

theory T_{n+1} is consistent.

To show that in the nontrivial case T_{n+1} is decidable we reduce the decidability problem for this theory to decidability of T_n. As in the proof of Lemma 5.6, here by a reduction we mean an effective mapping f associating with each sentence σ of the language $L(T_{n+1})$ a sentence $f(\sigma)$ of the language $L(T_n)$ such that $T_{n+1} \vdash \sigma$ iff $T_n \vdash f(\sigma)$. Thus a decision procedure whether σ is a theorem of T_{n+1} will be reduced to that of whether $f(\sigma)$ is a theorem of T_n. This will mean decidability of T_{n+1}, since T_n is decidable, by the inductive hypothesis. We take $f(\sigma) \stackrel{df}{=} \exists x[\varphi_n(x)\&\sigma(x/c_n)]$. Given a sentence σ in $L(T_{n+1})$ and a proof d of σ in T_{n+1}, we notice that the axiom $\varphi_n(c_n)$ can occur at most finitely many times in d. We can obtain a proof d' of $\exists x[\varphi_n(x)\&\sigma(x/c_n)]$ in T_n out of d as follows. Replace each occurrence of the axiom $\varphi_n(c_n)$ in d by $\exists x\varphi_n(x)$. Then introduce the prefix $\exists x[\varphi_n(x)\&\ldots]$ to each member (step) of d containing c_n, possibly renaming the other variables. Finally, substitute each occurrence of c_n with x. The other way around is proved similarly. Thus f is a reduction.

To complete the proof, let T' be the union of all the theories T_n, $n \geq 0$. Clearly, T' is consistent. It is decidable as a recursive union of recursive sets. □

Proof of Theorem 5.1. Let T be a consistent decidable theory. As the first step we construct a consistent, decidable, complete, and Skolemized extension T^* of T. To this end we construct recursively a sequence L_0, L_1, L_2, ... of languages and a sequence T_0, T_0', T_1, T_1', T_2, T_2', ... of theories as follows. We start off with $L_0 = L(T)$ and $T_0 = T$ and set T_0' to be an extension of T provided by Lemma 7.1. Suppose we are given a consistent and decidable theory T_n in language $L_n = L(T_n)$. We take $L_{n+1} = L(T')$ and $T_{n+1} = T'$ where T' is an extension of T_n provided by Lemma 7.2. Clearly, T_{n+1} is consistent and decidable. Now we take the union of all of them:

$$L^* = \bigcup_{n \geq 0} L_n$$

and

$$T^* = \bigcup_{n \geq 0} T_n = \bigcup_{n \geq 0} T_n'.$$

T^* is decidable, consistent, complete, and Skolemized.

As for the rest of the proof we only use a standard construction of the canonical model for T^* (see, e.g., [15], pp. 44–46). We represent the resulting equivalence relations on the natural numbers. Equivalence classes can be identified with their least (number) representatives. The resulting model is computable since T^* is decidable. □

Acknowledgments. The third author would like to thank Richard Kaye for a number of inspiring conversations. Thanks to an anonymous referee who suggested to improve the paper by including a proof of Theorem 5.1 as an appendix.

REFERENCES

[1] F. Bauer and H. Wössner, *Algorithmic Language and Program Development*, Springer-Verlag, Berlin, Heidelberg, New York, 1982.

[2] J.A. Bergstra and J.V. Tucker, *Initial and final algebra semantics for data type specifications: Two characterization theorems*, SIAM J. Comput., 12 (1983), pp. 366–387.

[3] J. Ferrante and C. Rackoff, *A decision procedure for the first order theory of real addition with order*, SIAM J. Comput., 4 (1975), pp. 69–76.
[4] J.A. Goguen, J.W. Thather, and E. Wagner, *An initial algebra approach to the specification, correctness and implementation of abstract data types*, in Current Trends in Programming Methodology, R. Yeh, ed., Prentice-Hall, Englewood Cliffs, NJ, 1978, pp. 80–149.
[5] M.J. Fischer and M.O. Rabin, *Super-Exponential Complexity of Presburger Arithmetic*, SIAM-AMS Proc. 7, AMS, Providence, RI, 1974, pp. 27–41.
[6] D. Harel, *First Order Dynamic Logic*, Lecture Notes in Comput. Sci. 68, Springer-Verlag, Berlin, New York, 1979.
[7] V.S. Harizanov, *Pure computable model theory*, in Handbook of Recursive Mathematics, Yu. L. Ershov, S.S. Goncharov, A. Nerode, J.B. Remmel, and V.W. Marek, eds., Stud. Logic Found. Math. 138, North-Holland, Amsterdam, 1998.
[8] S. Kahrs, D. Sannella, and A. Tarlecki, *The definition of Extended ML: A gentle introduction*, Theoret. Comput. Sci., 173 (1997), pp. 445–484.
[9] T.S. Millar, *Foundations of recursive model theory*, Ann. Math. Logic, 13 (1978), pp. 45–72.
[10] R. Milner, R. Harper, and M. Tofte, *The Definition of Standard ML*, MIT Press, Cambridge, MA, 1991.
[11] G. Mirkowska and A. Salwicki, *The algebraic specifications do not have the Tennenbaum property*, Fund. Inform., 28 (1996), pp. 141–152.
[12] G. Mirkowska and A. Salwicki, *Algorithmic Logic*, PWN, Warsaw, Poland, and D. Reidel, Dordrecht, the Netherlands, 1987.
[13] M. Presburger, *Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen in welchem die Addition als einzige Operation hervortritt*, in Comptes Rendus du 1er Congrès des Mathematiciens des Pays Slaves, Warszawa, 1929, pp. 92–101.
[14] D. Sannella and A. Tarlecki, *Mind the Gap! Abstract Versus Concrete Models of Specifications*, Lecture Notes in Comput. Sci. 1113, Springer-Verlag, Berlin, 1996, pp. 114–134.
[15] J. Shoenfield, *Mathematical Logic*, Addison-Wesley, Reading, MA, 1967.
[16] A. Tarski, A. Mostowski, and R. Robinson, *Undecidable Theories*, North-Holland, Amsterdam, 1953.
[17] S. Tennenbaum, *Non-Archimedean models of arithmetic*, Notices Amer. Math. Soc., 1959, p. 270.
[18] M. Wirsing, *Algebraic specification*, in Handbook of Theoretical Computer Science, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, pp. 676–788.